Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Abstraction Techniques for the Analysis and Synthesis of Critical Cyber-Physical System Architectures

PhD Dissertation

Kristóf Marussy

Thesis supervisor:
István Majzik, PhD

Budapest, 2023

Kristóf Marussy
http://home.mit.bme.hu/~marussy/

January 16, 2023

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

H-1117 Budapest, Magyar tudósok körútja 2.

Declaration of own work and references

I, Kristóf Marussy, hereby declare that this dissertation, and all results claimed therein are my own work, and rely solely on the references given. All segments taken word-by-word, or in the same meaning from others have been clearly marked as citations and included in the references.

Nyilatkozat önálló munkáról, hivatkozások átvételéről

Alulírott Marussy Kristóf kijelentem, hogy ezt a doktori értekezést magam készítettem és abban csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerint, vagy azonos tartalomban, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Budapest, 2023. január 16.

Marussy Kristóf

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Dr. István Majzik, for his guidance and patience.

I am thankful to all my former and present colleagues in the Critical Systems Research Group, especially Prof. András Pataricza and Dr. Zoltán Micskei for their hard work in guiding the research group. I would like to thank all my co-authors and colleagues: Aren A. Babikian, Márton Búr, Krisztián Buza, Piroska Buzáné Kis, Boqi Chen, Dániel Darvas, Máté Földiák, Bence Graics, Ákos Hajdu, Csaba Hajdu, Attila Klenik, Júlia Koller, Anqi Li, Chuning Li, Brett H. Meyer, Vince Molnár, Simon József Nagy, Ladislav Peska, Oszkár Semeráth, Gábor Szárnyas, Zoltán Szatmári, Dániel Szekeres, Tamás Bartha, Miklós Telek, Nenad Tomašev, Dániel Varró, András Vörös, and others at Department of Measurement and Information Systems, the Department of Computer Science and Information Theory, and the Department of Networked Systems and Services at Budapest University of Technology and Economics, as well as the Department of Electrical and Computer Engineering at McGill University. I was fortunate to work with many talented students, including Márk Ángyán, Dóra Cziborová, Márton Marcell Golej, Zhekai Jiang, Benedek Juhász, Tímea Molnár, Inez Anna Papp, Dávid Dorián Pete, Zsuzsanna Randóti, and Dorottya Szabó.

Parts of this work were performed in the MTA-BME Lendület Cyber-Physical Systems Research Group. This work was partially supported by

- ARTEMIS-JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP project;
- the ÚNKP-16-2-I and ÚNKP-18-3-I New National Excellence Programs of the Ministry of Human Capacities;
- the ÚNKP-21-3-II New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund;
- the EFOP-3.6.2-16-2017-00013 and EFOP-4.2.1-16-2017-00021 grants of the European Union, co-financed by the European Social Fund;
- Project no. 2018-1.3.1-VKE-2018-00040 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.3.1-VKE funding scheme;
- Project no. 2019-1.3.1-KK-2019-00004 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.3.1-KK funding scheme;
- Schnell László Foundation; and
- the 2022 Amazon Research Award “Graph Solver as a Service”.

I am grateful to my hosts Prof. Vittorio Cortellessa, Romina Eramo, and Michele Tucci for the research visit to the University of L’Aquila. I would also like to acknowledge the Student Research Trainee program of McGill University. I would like to thank the researchers and developers who provided guidance in interpreting their research results or assisted with using their software tools: Clément Ballabriga, Julien Forget, Andrew S. Miner, and Martin Sicks.

I am also grateful to our industry collaborators: IncQuery Group GmbH, Péter Györke and thyssenkrupp Hungary Kft., Péter Lantos and Prolan Zrt., and AbsInt GmbH.

Finally, but most importantly, I would like to extend my deepest gratitude towards parents for their continuous support and encouragement.

Köszönetnyilvánítás

Mindenekelőtt szeretnék köszönetet mondani konzulensemnek, Dr. Majzik Istvánnak az fáradhatatlan iránymutatásáért és türelméért.

Hálás vagyok továbbá minden korábbi és jelenlegi munkatársamak a Kritikus Rendszerek Kutatócsoportból, különösen Prof. Pataricza Andrásnak és Dr. Micskei Zoltánnak a csoport vezetéséért végzett áldozatos munkájukért. Szeretném továbbá megköszönni az összes társszerzőmnek és munkatársamnak a közös munkát: Aren A. Babikiannak, Búr Mártonnak, Buza Krisztiánnak, Buzáné Kis Piroskának, Boqi Chennek, Darvas Dánielnek, Földiák Mátéának, Graics Bencének, Hajdu Ákosnak, Hajdu Csabának, Klenik Attilának, Koller Júliának, Anqi Linek, Chuning Linek, Brett H. Meyernek, Molnár Vincének, Nagy Simon Józsefnek, Ladislav Peskának, Semeráth Oszkárnak, Szárnyas Gábornak, Szatmári Zoltánnak, Szekeres Dánielnek, Bartha Tamásnak, Telek Miklósnak, Nenad Tomaševnek, Varró Dánilnek, Vörös Andrásnak, és még sok másnak a Budapesti Műszaki és Gazdaságtudományi Egyetem Méréstechnikai és Információs Rendszerek Tanszékéről, Számítástudományi és Információelméleti Tanszékéről és Hálózati Rendszerek és Szolgáltatások Tanszékéről, valamint a McGill Egyetem Electrical and Computer Engineering Tanszékéről. Szerencsém volt számos rendkívül tehetséges hallgatóval is együtt dolgoznom: Ángyán Márkkal, Cziborová Dórával, Golej Márton Marcellel, Zhekai Jianggal, Juhász Benedekkel, Molnár Tímeával, Papp Inez Annával, Pete Dávid Doriánnak, Radnóti Zsuzsannával és Szabó Dorottyával.

A disszertációban bemutatott munka egy részét az MTA-BME Lendület Kiberfizikai Rendszerek Kutatócsoportban végeztem. A munkát ezen felül támogatta

- az ARTEMIS-JU és a Nemzeti Kutatási, Fejlesztési, és Innovációs Alap az R5-COP projekt keretében,
- az Emberi Erőforrások Minisztériuma ÚNKP-16-2-I and ÚNKP-18-3-I Új Nemzeti Kiválóság Programjai,
- a Innovációs és Technológiai Minisztérium ÚNKP-21-3-II Új Nemzeti Kiválóság Programja a Nemzeti Kutatási, Fejlesztési, és Innovációs Alap forrásából,
- az Európai Unió EFOP-3.6.2-16-2017-00013 és EFOP-4.2.1-16-2017-00021, melyet részben az Európai Szociális Alap finanszírozott,
- a 2018-1.3.1-VKE-2018-00040 számú projekt a Nemzeti Kutatási, Fejlesztési, és Innovációs Alap támogatásával lett végrehajtva a 2018-1.3.1-VKE pályázat finanszírozásával,
- a 2019-1.3.1-KK-2019-00004 számú projekt a Nemzeti Kutatási, Fejlesztési, és Innovációs Alap támogatásával lett végrehajtva a 2019-1.3.1-KK pályázat finanszírozásával,
- a Schnell László Alapítvány, és
- a 2022 Amazon Research díj: „Graph Solver as a Service”.

Szeretnék köszönetet mondani Prof. Vittorio Cortellessának, Romina Eramonak és Michele Tuccinak, hogy vendégül láttak a L’Aquilai Egyetemen. Hálás vagyok továbbá a kanadai McGill Egyetem kutatói gyakornoki programjának. Szeretnék köszönetet mondani a kutatóknak és fejlesztőknek, akik segítettek az eredményeik értelmezésében és a szoftvereszközeik használatában: Célment Ballabrigának, Juliet Forget-nek, Andrew S. Minernek és Martin Sicksnek.

Hálás vagyok ipari partnereinknek is, akikkel együtt dolgozhattam: az IncQuery Group GmbH-nak, Györke Péternek és a thyssenkrupp Hungary Kft.-nek, Lantos Péternek és a Prolan Zrt.-nek, valamint az AbsInt GmbH-nak.

Végül, de nem utolsó sorban, szeretném megköszönni szüleimnek kitartó támogatásukat és bátorításukat.

Summary

Cyber-physical systems (CPS) are smart systems that include highly interconnected digital, analog, physical, and human components. They are increasingly prevalent in safety critical applications, such as infrastructure for self-driving cars and smart cities. This means that, in addition to their functional requirements, critical CPS have to satisfy stringent *extra-functional requirements*, including *quantitative* dependability, performance, and execution time requirements.

During the design of complex systems, the selected system architecture can have a profound impact on the satisfaction on the extra-functional requirements. In *architecture-based* analysis, specialized *analysis models* (e.g., stochastic models) are derived from candidate architectures by *view transformations* in order to compute quantitative extra-functional measures. The definition of appropriate transformations often requires integration of expertise from multiple viewpoints (e.g., dependability attributes of both hardware and software components).

The large (even possibly unbounded) number of *candidate architectures* makes manual analysis of all possible configurations intractable and necessitates automated *design space exploration*. While *(meta-)heuristic* approaches can handle a wide variety of extra-functional requirements, including those defined with the help of *view transformations*, they do not provide any formal guarantees for the completeness of exploration. In contrast, *logic solver based* approaches offer sound and complete reasoning, but problems with complex constraints, such as numerical constraints either cannot be expressed at all or only at a steep performance penalty.

This work aims to improve the support for extra-functional requirements in reasoning over architecture models. We aim to (1) facilitate analysis in the early stages of design by the use of *graph abstractions* to explicitly describe unknown attributes of architectures and decisions yet to be made, and (2) add support for extra-functional requirements to logic solvers while keeping their previous formal guarantees intact.

Firstly, we adopt and extend *partial models* as a common theoretical foundation for (1) and (2). We propose *4-valued partial models*, as well as partial models with *scope constraints* about model size and relationship multiplicities as graph abstractions for the analysis of architecture models.

Secondly, we provide *reasoning capabilities* over the proposed partial modelling formalisms. For (1), we propose a *fully compositional view transformation language* and *inconsistency-tolerant transformation engine* for 4-valued partial models. Moreover, for (2) we propose algorithms to reason about partial models with scope constraints, which can effectively reduce the design space to be explored.

Lastly, we propose concrete analyses based on the aforementioned modelling and reasoning capabilities. We propose a *mission automaton* formalism to describe reconfiguration strategies. This allows the construction of a *Phased-Mission System* of *stochastic Petri nets* to analyze the dependabiltiy and performability of a reconfigurable system. Additionally, we exploit scope-based numerical reasoning to tackle the *witness model synthesis* problem for the *worst-case execution time* (WCET) analysis of query-based runtime monitors for critical distributed systems. The highly data-driven nature of these programs posed a significant challenge for previous WCET analysis tools.

As added value, view transformations and mission automata enabled the architecture-based analysis of complex, reconfigurable systems. Incorporating scope propagations into a logic solver based model generator achieved a significant speedup in architecture candidate generation compared to state-of-the-art logic solver based techniques. The application of these techniques to WCET analysis allowed tighter WCET estimates and the handling WCET analysis tasks where previous methods were inapplicable.

The contributions of the thesis are demonstrated and evaluated on case studies adapted from industrial problems and collaborations. Tools developed in the context of the work are available as open-source software.

Összefoglaló

A *kiberfizikai rendszerek* (CPS) olyan intelligens rendszerek, amelyek nagymértékben összekapcsolt digitális, analóg és komponenseket tartalmaznak, emberekkel együttműködve. Az ilyen rendszerek egyre elterjedtebbek a biztonságkritikus alkalmazásokban, mint például az önvezető autókban és az intelligens városok infrastruktúrájában. Így a kritikus CPS-eknek a funkcionális követelményeken túl szigorú *extra-funkcionális követelményeknek* is meg kell felelniük, beleértve a *kvantitatív* megbízhatósági, teljesítmény- és végrehajtási idő követelményeket.

A komplex rendszerek tervezése során a kiválasztott rendszerarchitektúra nagymértékben befolyásolhatja az extra-funkcionális követelmények teljesülését. Az *architektúraalapú* elemzés során az architektúra javaslatokból *nézeti transzformációkkal* állítunk elő specializált *analízis modelleket* (pl. sztochasztikus modelleket) a kvantitatív extra-funkcionális mérőszámok kiszámítása érdekében. A transzformációk meghatározása gyakran több nézőpont szaktudásának integrálását igényli (pl. a hardver- és szoftverkomponensek megbízhatósági attribútumai).

Az *architektúra javaslatok* nagy (esetleg korlátlan) száma miatt az összes lehetséges konfiguráció kézi elemzése lehetetlen, így a *tervezési tér* automatizált bejárása szükséges. Bár a *(meta-)heurisztikus* stratégiák az extra-funkcionális követelmények széles skáláját képesek kezelni – beleértve a *nézet-transzformációk* segítségével meghatározottakat is –, nem nyújtanak formális garanciát a bejárás teljességére. Ezzel szemben a *logikai megoldókon* alapuló megközelítések helyes és teljes körű érvelést kínálnak, de az összetett kényszereket, például numerikus kényszereket vagy egyáltalán nem, vagy csak jelentős teljesítményveszteség mellett lehet kifejezni a bemeneti nyelvükön.

Értekezésem célja, hogy javítsa az extra-funkcionális követelmények támogatását az architektúramodellek feletti következtetés során. Célom, hogy (1) megkönnyítsem – a még meghozandó döntések explicit leírására *gráf absztrakciók* használatával – az elemzést a tervezés korai szakaszában, valamint, hogy (2) kiegészítsem a logikai megoldókat a korábbi formális garanciák megtartása mellett az extra-funkcionális követelmények támogatásával.

Egyrészt felhasználom és kibővítem a *parciális modell* formalizmust, mint az (1) és (2) feladatok közös elméleti alapját. Így a *4-értékű parciális modelleket* és a *darabszám kényszerekkel* ellátott parciális modelleket javasolom gráf absztrakcióként az architektúra modellek elemzéséhez.

Másrészt a javasolt részleges modellezési formalizmusok felett *érvelési technikákat* biztosítok. Az (1) feldathoz *teljesen komponálható nézeti transzformációs nyelvet* és *inkonzisztencia-tűrő transzformációs motort* javasolok a 4-értékű részleges modellek esetén. A (2) feladathoz algoritmusokat javasolok az érveléshez a darabszám kényszerekkel ellátott parciális modelleken.

Végül konkrét elemzéseket javasolok a fenti modellezési és következtetési technikák felhasználásával. Az újrakonfigurációs stratégiák leírására a *misszió automata* formalizmust javasolom, mely lehetővé teszi *többfázisú missziós rendszer* (Phased-Mission System, PMS) *sztochasztikus Petri-háló* modellek felépítését újrakonfigurálható rendszer szolgáltatásbiztonságának elemzésére. Továbbá felhasználtam a darabszám kényszeres érvelést, hogy megoldjam a *tanú modell szintézis* problémát a kritikus elosztott rendszerek lekérdezésalapú futásidejű monitorainak *leghosszabb futásidő* (Worst-Case Execution time, WCET) elemzéséhez.

A munka hozzáadott értéke, hogy a nézeti transzformációkkal és misszió automatákkal lehetővé teszi a komplex újrakonfigurálható rendszerek architektúraalapú elemzését. A darabszám kényszerek beépítése egy logikai megoldó alapú modellgenerátorba jelentős gyorsulást eredményez az architektúra javaslatok generálásában az eddigi legkorszerűbb logikai megoldó alapú technikákhoz képest. Ezen technikák alkalmazása élesebb WCET becsléseket eredményez, és olyan WCET elemzéseket is lehetővé tesz, ahol a korábbi módszerek nem alkalmazhatók.

Az értekezés kontribúcióit ipari problémákból és együttműködésekből adaptált esettanulmányokon keresztül mutatom be és értékelem. A munka során kifejlesztett eszközök nyílt forráskódú szoftverekként érhetők el.

Contents

Chapter 1

Introduction

1.1 Cyber-Physical Systems

Cyber-physical systems (CPS) are smart systems that include highly interconnected digital, analog, physical, and human components. They provide new functionalities that improve quality of life and enable technological advances in critical areas, such as personalized healthcare, traffic flow management, smart manufacturing, energy supply, autonomous vehicles, and intelligent buildings [NIST17].

The pervasive interconnectedness of computational and physical aspects, as well as the often critical roles these systems are employed lead to *specific characteristics* of CPS that go beyond traditional system and application design:

- CPS may be deployed in a *wide variety of configurations* and on *heterogeneous computing* and *communication platforms* and are often composed with other systems as part of a *System of Systems*. Engineering such systems needs a methodology to ensure the interoperability of components, manage the evolution of the requirements and the design, and deal with any harmful phenomena emerging from the complex interactions.

- The *potential impact* of CPS on the physical world, up to the possibility of massive economic damage or even the loss of life, places a heightened demand for trustworthy systems. These concerns result in the need for rigorously proven security, privacy, safety, reliability, and resilience. The systems may rely on runtime adaptation techniques to ensure that their dependability objectives are met in the face of changes in the operating environment and failure processes of the hardware [Epi+09; FB16; IW17].

- The interactions between CPS and their operating environment are often *time sensitive*. There may be hard real-time requirements placed on certain functions. For example, we must ensure that the latency between sensing and actuation remains limited for correct control loops behavior.

Therefore, CPS have to satisfy stringent *extra-functional requirements*, such as maintainability, reusability, extensibility, cost, security, privacy, safety, dependability, scalability, and performance, in addition to their functional requirements [Fel03]. *Model-driven engineering* has been widely applied as a methodology to facilitate CPS design in accordance with these goals, especially in the context of changing and evolving requirements and architectures [Vog+15].

1.1.1 Architecture-based analysis of extra-functional requirements

A large fraction of the extra-functional requirements is *quantitative*, i.e., they can be evaluated in a mathematically precise way by computing some *metric* on either the CPS design artifacts or an auxiliary *analysis model*. For complex metrics, the architecture models (e.g., block diagrams) of the CPS are usually not sufficient, and we have to adopt specific mathematical formalisms for constructing analysis models. Among these, stochastic models, such as *fault trees* [Xia+11; Gha+17; Get+18][c17], *Markov chains* [KB09], *queuing networks* [KR08], and *Generalized Stochastic Petri Nets* (GSPN) [BMM99; BDD04; LMC04; Ndi+16; CET18][c7; c12], serve as analysis models for reliability, availability, and performance metrics.

Manual creation of analysis models requires specialized expertise and meticulous work for each architectural alternative to be analyzed. For complex design spaces, a large (or possibly even infinite) amount of candidate architectures must be analyzed [CA05; KR11; GTC15]. Such large-scale manual analyses are infeasible. To alleviate this problem, *architecture based* analysis techniques, e.g., [BMM99; BDD04; FD16; KR08; Gha+17], have been developed, which rely on *model transformations* to automatically construct analysis models from the architectural design artifacts. However, these techniques can only handle static architecture models: the possible runtime changes have to be fully described in the analysis model, e.g., as failure processes and other random processes in the case of stochastic modeling.

To enable runtime monitoring and adaptation, the models@run.time paradigm [BBF09; Che+11b; Búr+20] facilitates the capture of runtime knowledge about the system and its environment as a continuously maintained model. Although this allows applying model-driven engineering techniques to create adaptation and reconfiguration strategies, the mathematically precise extra-functional analysis of such strategies raises a need for evaluating the extra-functional properties of dynamic (changing) architecture models [Cal+12].

The first research objective investigated in this thesis aims to address this issue by extending the architecture-based approach to the runtime adaptation strategies.

Research objective 1 Quantitative extra-functional analysis of dynamic system architectures

1.1.2 Automated generation of design candidates

The large number of possible system configurations each with highly varying extra-functional characteristics poses a major challenge to find the most suitable system architecture in an early design phase [NIST17]. Engineers can manually inspect only a handful subset of candidate architectures. Therefore, various automated *design space exploration* (DSE) tools have been developed *for system architecture synthesis* to assist in finding viable candidate architectures that satisfy all functional and extra-functional constraints while optimizing for a target objective. DSE tools tackling this challenge are broadly classified into two categories:

(Meta-)heuristic techniques, such as genetic algorithms or multi-objective optimization [Mar+10; Abd+14; GTC15; BZS18; Arc+18; FTW16], can support a wide variety of analyses directly inside the DSE process to derive near-optimal design candidates. However, they do not guarantee a complete (exhaustive) enumeration of the design space and the optimality of the generated candidates [Ker+13]. Therefore, engineers are not informed about the cause of failure (e.g., an *unsatisfiable core*) if the exploration fails to produce the desired candidates. Moreover, encoding hard constraints (which must be satisfied at all times) either requires approximations by custom soft constraints (which can be violated, but violating solutions are penalized), objective functions and mutation operators, or it could significantly degrade the performance or scalability of the exploration [SNP13; BZJ21].

Logic solver based DSE techniques (e.g., [Jac02; KJS10]) have guaranteed *soundness* and *completeness*. They usually allow encoding complex logical hard constraints and logical formulas or model queries [CCR07; KHG11; Ujh+15; SNV18] and may provide an explanation when the synthesis task is unsatisfiable. However, purely logical constraints cannot capture most extra-functional requirements that rely on an external numerical solver to analyze. Thus, solvers have to be specifically extended for optimization tasks, such as in [Li+14; BPF15] to handle both logical and numerical constraints. Unfortunately, for complex extra-functional analysis tasks, such optimizing solvers are often unavailable.

Recently, logic-solver based graph model generation has been suggested, which takes advantage of *partial modelling* [RSW04; RD06] to explicitly represent design decisions yet to be made in the internal representation of the solver as a graph model [SV17; SNV18]. This represents a potential for applying architecture-based extra-functional analysis directly in the logic solvers to generate design candidates with completeness guarantees while keeping expressiveness similar to that of (meta)heuristic techniques.

The second research objective in this thesis focuses on this issue:

Research objective 2 Synthesis of candidate system architectures with completeness guarantees according to quantitative extra-functional requirements and objective functions

1.1.3 Representing uncertainty and variability

Both **Research objective 1** and **Research objective 2** require a representation of CPS architectures to analyze and synthesize. Architecture modeling languages, such as SysML, Palladio [Mar+10], Æmilia [Arc+18], and domain specific languages rely on *graph models* to represent system architectures, configurations, and deployments on heterogeneous computing and communication infrastructures [Vog+15]. In this setting, *graph transformations* can express reconfigurations as endogenous (in-place) transformations of the architecture models, while exogenous transformations can automatically derive analysis models as target models from the source architecture models [Koz10].

However, both **Research objective 1** and **Research objective 2** pose challenges not yet tackled by existing graph model representations.

1.1.3.1 Representations for view transformations

In **Research objective 1**, (endogenous) transformations for applying reconfiguration occur at the same time as (exogenous) *view transformations* for maintaining analysis models. It is possible to execute the view transformation from scratch after each reconfiguration, but this can have significant performance costs, and leads to the loss of valuable *traceability* information that connects the prior and current versions of the analysis model. Therefore, an ideal *view transformation engine* [Ber+12a; CET18] is *reactive* (i.e. reacts to source model changes), target *incremental* (i.e., updates only affected target elements), *consistent* (i.e., continuously maintains a transformation relation between source and target models) and *validating* (i.e., the target model is a valid instance of the target language).

A validating engine that satisfies all well-formedness constraints of the target language is at odds with reactive and incremental execution: depending on the changes to the architecture model, it may be the case that no valid target model exists at some time [c7]. By semantically preserving inconsistencies in an *inconsistency-tolerant* knowledge base, a large fragment of the source and view models can be kept sync in case of such model editing operations. This allows preserving traceability links and provides hippocratic behavior (i.e. avoids the unnecessary deletion and recreation of elements).

Moreover, the development of the analysis model transformation usually requires the collaboration of experts from multiple domains (e.g., embedded systems, communications infrastructure, hardware reliability). Integrating their results requires combining *partial* information obtained by model transformations created according to multiple viewpoints to construct the complete analysis model [CNS12].

1.1.3.2 Representations for design-space exploration

Regarding **Research objective 2**, the need for *partial* and *inconsistency-tolerant* representations of CPS architectures is even more explicit.

By introducing a *fixed number* of explicit points of variability into a system model, such as the number of redundant component instances and the possible allocations of functions, a genotype vector for systems models can be constructed [Mar+10]. This enables the use of efficient meta-heuristic algorithms either on the level of architecture models [Ale+09; Mar+10; Li+11; BFK19] or directly on the level of analysis models [GTC15].

While the aforementioned approaches offer scalability due to the fixed-length, domain-specific genotype encoding, such encodings are not directly applicable for problems with a *variable number of objects and connections*, such as communication network topologies. To this end, logic solver based approaches for model synthesis [SV17; SNV18] rely on partial graph models to encode design decisions yet to be made as unknown aspects of the design candidate being worked on.

Conversely, representing inconsistencies that arise during the execution of decision procedures can pinpoint contradictions in the requirements.

Because existing approaches with partial graph models have limited support for evaluating extra-functional metrics [FFJ12], there is a need for extending partial graph model formalisms with such support for the quantitative extra-functional analysis of CPS architectures. The contributions presented in the thesis in connection with **Research objective 2** aim to provide reasoning capabilities over these extended partial model representations.

In summary, the following research objective arises as the common theoretical background for **Research objective 1** and **Research objective 2**:

Research objective 3 Represent uncertainty caused by design decisions yet to be made, runtime reconfigurations, as well as inconsistencies in complex system architectures explicitly

1.2 Background and challenges

In this section, we overview the state-of-the-art results connected to our research objectives and identify five *challenges* to be addressed later in this thesis.

1.2.1 State space explosion

As we discussed in Section 1.1.2, the large number of possible hardware architectures, software component allocations, and configurations poses a significant difficulty in the verification and synthesis of design candidates. Even if we consider only a single node (component) type and a single edge (link) type, there are $2^{10\times10} \approx 10^{30}$ possible graph models with 10 nodes. Real system design artifacts (e.g., SysML, Palladio, or domain-specific models) have both more node and edge types, as well as more components. Thus, the *design space* formed by possible graphs models can be much larger, or potentially infinite when we consider node attributes with continuous range (e.g., probabilities and failure rates).

Approaches based on *(meta-)heuristics* aim to tackle this problem by only exploring a fraction of the design space. The heuristics, such as genetic algorithms, are used to sample parts of the design space in an adaptive manner that are most likely to contain optimal solutions of the design challenge. In this case, the explorations task consists of various *hard* and *soft constraints* on the design candidates and one or more *objectives* (usually corresponding to extra-functional metrics) to be optimized. As a benefit, a wide variety of constraints and objectives can be supported, even in a *multi-objective* setting where *Pareto-optimal* [REJ09] solutions according to multiple objectives are sought.

1.2.1.1 Genotype-based approaches

By translating the design candidates into a specialized parametric *exploration representation*, such as a finite-dimensional vector, approaches like ArcheOpteryx [Ale+09], AQOSA [Li+11], PerOpteryx [Mar+10], EvoChecker [GTC15], and Rodes [Cal+17] can execute genetic algorithms on design candidates with a high probability of convergence to near-optimal solutions. For example, a fixed number of parameters might be introduced for the allocations of software components to the hardware platform and for the available component variants. However, in many graph-like problems, were the number of components and their interconnections are also variable, it is either impossible to introduce such a finite parametrization, or the required number of parameters (e.g., at least n^2 parameters for links in a graph with n nodes) grows infeasibly large.

1.2.1.2 Graph-based approaches

In contrast, graph-based techniques use graph transformations [Agr+02] or refactorings to generate candidate designs as graph models. They either rely on *model-based* search, where a graph model is being mutated, or *rule-based* search, where solutions are encoded as a sequence of graph transformation operations [Joh+19]. MOMoT [FTW16] and MDEOptimiser [BZS18] rely on the Henshin model transformation language [Are+10] for model-based exploration. Hard constraints pose a challenge for such approaches: they are either handled by relaxation into *soft* constraints or by encoding them in the transformation rules [BZJ21].

Viatra-DSE [Abd+14; HHV15] is a rule-based DSE tool that relies on the Viatra [Ujh+15] language. SHEPhERd [CMP15] and EASIER [Arc+18] aim to derive sequences of software architecture refactorings according to extra-functional criteria. Synthesizing long chains of model transformations might be challenging in the presence of hard constraints; e.g., the effectiveness of evolutionary *crossover* operators is diminished compared to *mutation* operators [Abd+14].

Challenge 1 How to represent design candidates and perform search and analysis in the extremely large design and configuration spaces arising from complex CPS?

1.2.2 Functional and structural constraints

In model-driven engineering, general-purpose or domain-specific *systems modeling languages* (e.g., UML, SysML, Palladio [BKR08]) are used to represent system architectures. Such languages are defined by (1) a *metamodel* to capture the vocabulary of language with classes and references, while (2) consistent (valid) systems models also have to satisfy *well-formedness* (WF) constraints.

We will assume that *first-order logic* (FOL) can uniformly formalize (a) the constraints associated with the systems modeling language, (b) additional design rules, and (c) functional requirements. For example, if the system modeling language can describe the physical and functional system architecture and their allocations, FOL constraints can capture whether all

functions are allocated to physical components in a valid manner. Thus, as a theoretical basis, we may use FOL signatures and logic formulas to capture metamodels and WF constraints, respectively.

Structural constraints, including *type hierarchy*, *type compliance*, *multiplicity* constraints, *inverse relations*, and the *containment hierarchy* are often imposed by model management frameworks to ensure that the model artifact remains serializable [Ste+09]. Several highly expressive languages exist for the specification of well-formedness constraints, e.g., OCL [OCL] and VIATRA [Ber+11].

As such, generating valid models *requires solving a logical program with the satisfaction of user-provided FOL formulas* as *hard constraints*, which is highly challenging even in the absence of extra-functional metrics [CCR07; KHG11; Var+18]. Typically, only a small fraction of the possible model candidates will satisfy the complex logical hard constraints.

FOL hard constraints often pose a challenge in (meta-)heuristic DSE approaches. Burdusel et al. [BZJ21] proposed the automated generation of transformation rules that preserve a limited class of WF constraints (multiplicity constraints). PLEDGE [SSB20] combines evolutionary search with logic solving to preserve WF constraints over object attributes. Other, more complex FOL constraints cannot be encoded and have to be relaxed into soft constraints.

Logic solver based techniques for FOL problems include constraint programming, such as in UMLtoCSP [CCR07] and DesertFD [ENS10], SAT solving, such as in Alloy [Jac02] and CoBaSA [MVS07], and Satisfiability Modulo Theories (SMT), such as in FORMULA [KJS10].

SAT and SMT solvers commonly rely on the Davis-Putnam-Logemann-Loveland (DPLL) [DLL62; Kat+16] and the abstract DPLL [NOT06; Bra+13] algorithms for an efficient, *complete* enumeration of the design space. However, scalability of logic solvers may be limited in the case of graph-like synthesis problems [SNV18].

Recently, partial modeling [RSW04; Ren06] based graph generation was proposed [SNV18] as an implementation of a DPLL decision procedure for graph models, which improves performance over SAT solver for graph model generation. Nevertheless, integrating partial modeling based DPLL with other *background theories* for SMT solving, e.g., for attribute constraints, remains challenging.

Challenge 2 How to ensure that functional and structural constraints are satisfied in architecture and analysis models?

1.2.3 Dependability analysis models for reconfigurable systems

Specifying and analyzing extra-functional metrics and requirements in an architecture-based manner may pose significant challenges in complex, adaptive and reconfigurable systems. Not only an analysis model has to be constructed from the system architecture automatically, but the analysis itself must also remain tractable.

Methods for the construction of stochastic analysis models are widespread in the evaluation of dependability (including reliability and availability), and performance metrics based on architecture models, especially for component-based design [Koz10; BMP12]. Underlying analysis formalisms include *fault trees* [JVB17; Xia+11; Gha+17; Get+18], *Markov chains* [KB09], *queuing networks* [KR08], and *Generalized Stochastic Petri Nets* (GSPN) [MPB02; BDD04; MB15; Ndi+16; CET20].

The *dependability attributes* and *failure processes* may be described in two ways: (a) They may be represented directly in the architecture model using, e.g., UML stereotypes [MPB02], MARTE annotations [Iqb+15] or the AADL Error Annex [JVB17; Ale+09]. Alternatively, (b) the transformation definition itself may encode the dependability attributes of the components (as analysis model fragments). To retain flexibility in the transformation definition and aggregate knowledge

for multiple domain experts, transformation definitions may be composed *sequentially* [Anj+14; Heg+16] or in *parallel* [CNS12; DXC11], where the various elementary transformations to be composed describe the dependability attributes of different parts of the system.

The analysis of adaptive systems poses further challenges. If the adaptations are *dynamically* synthesized in response to failures and events from the environment, *quantiative verification* must be employed at runtime [Cal+12] to verify the proposed adaptations. If the system uses a *static* adaptation strategy, an analysis model that includes the behaviors of all possible configurations of the system along with the strategy may grow so large that it makes the analysis intractable.

To alleviate this issue, *Phased-Mission System* (PMS) [MB99] models were developed, where *phases*, which are stochastic models (e.g., fault trees, Markov chains, GSPNs) describing the behaviors of system configurations, are connected by a *high-level* model describing the adaptation strategy. Numerical [SRA92; MB99] or combinatorial [Xin07; WT07] analysis methods can process phases one-by-one, connecting the individual models later according to the high-level model. Therefore, this formulation reduces both the memory and computation demands of extra-functional analysis.

However, to our best knowledge, while analysis models for individual phases can be constructed from architecture models by model transformations, no approach supports simultaneously deriving all phases and the high-level model for the PMS analysis of an adaptation strategy.

Challenge 3 How to compactly specify and analyze dependability measures for families of reconfigurable architecture model candidates?

1.2.4 Witness models for Worst-Case Execution Time Analysis

In embedded systems, runtime monitoring programs are integral components of the system that analyze events and execution traces [Bar+18] in order to detect potentially critical situations that violate a requirement. Task schedulability and real-time requirements demand the use of runtime monitors that adhere to *Worst-Case Execution Time* (WCET) constraints [Pik+10; HR02].

According to the models@run.time [BBF09; Che+11b] paradigm, runtime monitors are continuously executed on runtime *snapshots*, i.e., models describing the runtime state of the system. The *heavily data-driven* execution of such runtime monitors compared to the *low expressiveness* of traditional runtime monitoring solutions [Hav15] makes producing analysis models for safe and tight WCET estimation challenging [Búr+18; Har+19; DBB18]. This necessitates the use of *semantic-aware WCET estimation* [Mai+17] to consider limitations on input data (i.e., constraints restricting the possible runtime snapshots).

The *Implicit Path Enumeration Technique* (IPET) [LM97] constructs *Integer Linear Programs* (ILP) as WCET analysis models according to the *Control Flow Graph* (CFG) on the runtime monitor programs. The IPET ILP can accurately represent the *microarchitectural timing characteristics* of the target execution platform along with the control flow of the program.

The effectiveness of WCET analysis relies on precise program *flow facts* (e.g., loop bounds, infeasible paths). Although there are several algorithms to derive additional constraints on the program flow [Gus+06; Erm+07; CJ11; KKZ13; Lis14], there is still a significant manual effort needed to specify flow facts [Abe+15] to represent *domain-specific* semantic information about program inputs.

We identified three domain-specific analysis scenarios for runtime monitors: (i) WCET based on a single runtime snapshot, (ii) WCET based on the metamodel and well-formedness constraints of the runtime snapshots, and (iii) WCET based on a *partial* runtime snapshot. The latter enables reasoning when some information is fixed at design time (e.g., a railway network) while the rest of the model changes at runtime (e.g., the positions of trains).

Classical WCET analysis methods (without domain-specific analysis) can handle scenario (i) by *data flow analysis*. However, they require over-approximations based on *loops bounds* for scenario (ii), because they cannot take well-formedness constraints of runtime models into account. They fail to address scenario (iii), because data flow analysis cannot process partial data that without a pre-allocated memory layout.

In [Búr21][j5], Búr proposed *witness models* to tackle scenarios (ii) and (iii). The witness model is a concrete runtime snapshot that maximizes the estimated upper bound of the execution time of the runtime monitor. Thus, it provides a safe and tight estimate of the WCET. However, *synthesizing* the witness model based on the domain constraints remains challenging.

Challenge 4 How to synthesize witness models for domain-specific Worst-Case Execution Time evaluation?

1.2.5 Numerical reasoning with extra-functional attributes

The analysis of quantiative extra-functional properties requires *numerical* (quantitative) *reasoning* for evaluating *numerical constraints* over the attributes of the architecture models, as well as computing *extra-functional metrics* during synthesis.

In the case of (meta-)heurisic synthesis approaches, a concrete candidate architecture is available at all times during synthesis. Thus, analysis models can be constructed by model transformations and analyzed using analysis tools, such as in [Ale+09; Mar+10; Li+11; FTW16; BZS18].

In logic solvers, the supported metrics and objectives is limited by the input language and the supported *background theories* [Kat+16]. For supported theories, optimizing SMT solvers provide capabilities for finding globally optimal solutions [Li+14; BPF15]. The Nelson–Oppen procedure [NO79] is widely employed as means of combining background theories.

However, the scalability of traditional logic solvers may be limited in the case of graph-like synthesis problems [SNV18]. Tableau-based reasoning for graphs has been proposed in [Pen08; SLO17; ADW16], but these approaches lack the support for reasoning with extra-functional properties.

In abstract interpretation, *abstract domains* [CH78; SPV18] are used to reason about the (numerical) values of program variables and expressions. *Relational* abstract domains, such as *polyhedron abstraction* [CH78; BHZ08] can represent arbitrary numerical relationships between arbitrary program variables. Reasoning techniques for polyhedra include *Linear Programming* (LP) and *Integer Programming* (IP) solvers [Clp; Cbc], as well as specialized representations [Nin15].

In contrast, *non-relational* domains, such as *interval abstraction* reduce expressiveness by restricting each representable numerical constraint to only refer to a *single* variable, but offer more efficient reasoning e.g., by *interval arithmetic* [Kul09] and *propagators* [Nin15].

Even though partial modeling [RSW04; Ren06] based graph generation has been recently proposed [SNV18] as an implementation of a DPLL [Kat+16] decision procedure for graph models, the support for numerical reasoning remained lacking. There are two sources of quantitative values in a graph model: (i) objects of the model may be equipped with numerical attributes (e.g., elementary performance or dependability metrics), and (ii) the *size* or *cost* measures may depend on the number of objects.

For (i), the combination of abstract domains and partial models [Mag+07; MRS10; FFJ12] was proposed, especially for the *value analysis* of heap and pointer-based programs [APV09], where partial models represent possible program states. For (ii), the most closely related works are *structural counter abstraction* [Ban+13] for graph transformation systems and *model-based quantifier instantiation* [Rey+13] in SMT solvers. Representing models with numerical attributes where the model size (and thus the number of attribute values to be represented) is not known

Table 1.1: Research objectives and challenges

	Ch. 1	Ch. 2	Ch. 3	Ch. 4	Ch. 5
Research objective 1	○	●	●	○	○
Research objective 2	●	●	○	●	●
Research objective 3	○	○	●	○	●

in advance has been tackled by *summarized dimensions* [Gop+04].

However, the aforementioned techniques do not in themselves provide a sound and complete decision procedure for graph generation, and their use remains limited to program verification.

Partial modeling allows a sound and complete decision procedure for graph generation and synthesis by computing *under-* and *over-approximations* of the constraints to be satisfied [SV17] following the abstract DPLL [NOT06; Bra+13] framework. However, its use is currently limited to logical (FOL) well-formedness constraints. Thus, the use of numerical abstractions over (i) model attributes and (ii) model size, as well as the combination of various reasoning techniques in the context of graph models remains challenging.

Challenge 5 How to reason about the quantitative extra-functional aspects of complete or incomplete system architectures automatically and efficiently?

The relationships between our research objectives and the aforementioned challenges are summarized in Table 1.1.

1.3 Research method

My research method has been aligned with the best practices of software engineering research. **Research objectives 1–3** were motivated by generalizing industrial problems and case studies. The feasibility and applicability of the contributions were demonstrated by open source prototype implementations, while their scalability was evaluated by synthetic and real-world performance benchmarks. The prototypes are integrated with industrial and academic tools, such as EMF [Ste+09], VIATRA Query [Ujh+15], VIATRA Generator [SNV18], and PetriDotNet [j6].

In particular, the following benchmarks and case studies were used to motivate and evaluate the contributions:

The **Interferometry Mission Architecture Optimization** [Her+17] case study was adapted from the NASA Jet Propulsion Laboratory. It aims at deriving design candidates for interferometry missions with multiple satellites (including CubeSats and small satellites) communicating over radio links with each other and a ground station using an optimization approach based on model transformations.

The **Train Benchmark** [Szá+17] is an open source framework for measuring the performance of continuous model transformations, with a particular emphasis on the performance of incremental query reevaluation. It has been actively used within the model-driven engineering community as a cross-technology performance benchmark. It uses a domain-specific model of a railway system originating from the MOGENTES EU FP7 [MOG] project.

The **Model-Based Demonstrator for Smart and Safe Cyber-Physical Systems (MoDeS3)** [Vör+18b] showcases various challenges in intelligent safety-critical CPS. It extends the *structure model* of a railway system from the Train Benchmark with a *runtime model* [BBF09] that tracks the positions of trains, which enables detecting and correcting potentially unsafe situations by model-based runtime monitoring [Bar+18; Búr+20].

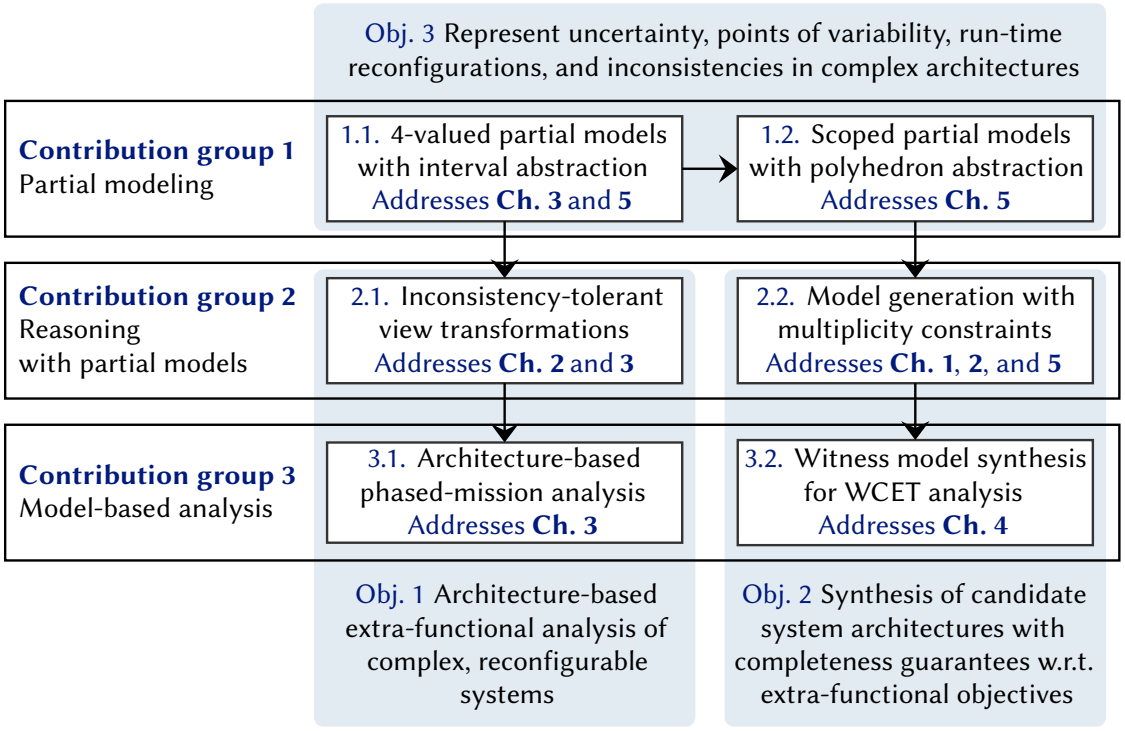


Figure 1.1: Overview of research objectives and contributions

Lastly, the **Flexible Manufacturing System (FMS)** [CT93] case study illustrates the performability analysis of an industrial CPS system using Petri nets.

1.4 Contribution overview

This thesis presents the contributions related to **Research objectives 1–3** in three groups. The groups are arranged starting from the theoretical foundations to concrete extra-functional analyses. Figure 1.1 shows the relationship between **Research objectives 1–3**, **Challenges 1–5**, and **Contributions 1–3**.

Contribution group 1 addresses **Research objective 3** by proposing novel *partial model* formalisms to represent graphs models with unknown or inconsistent information along with extra-functional attributes of model elements.

Contribution group 2 provides *structural* and *numerical reasoning* capabilities over the partial models to efficiently construct analysis models and provide *over-* and *under-approximation* of extra-functional metrics. Taking advantage of these capabilities, **Contribution group 3** presents two architecture analysis methods.

The contribution groups are split into two contributions each. *Contribution 2.1* provides reasoning capabilities with analysis models for dynamic architecture models, which are exploited in *Contribution 3.1* for the *phased-mission analysis* of reconfigurable CPS to address **Research objective 1**. Likewise, *Contribution 2.2* provides reasoning capabilities over systems of linear equations and the sizes of models, which are exploited in *Contribution 3.2* for the WCET analysis of *monitor query programs* to address **Research objective 2**.

Contribution group 1 I defined *4-valued* and *scoped* extensions of the partial graph model formalism for the quantitative extra-functional analysis of system architecture models with unknown properties and pending design decisions.

- *Contribution 1.1* I formalized *4-valued partial models* as a conceptual core to introduce *inconsistency-tolerance* into architecture models and other graph models [j2; c7; d21].
- *Contribution 1.2* I formalized *scoped partial models* as a conceptual core for model generation with linear numerical constraints, proposing the use of *polyhedron abstraction* to express constraints on model size, number of graph pattern matches, and costs [j1].

Contribution group 2 I proposed two reasoning techniques for deriving quantitative extra-functional metrics from partial models of systems and considering multiplicity constraints on architecture models.

- *Contribution 2.1* I proposed a fully compositional view transformation language and developed a reactive, incremental, validating, and inconsistency-tolerant view transformation engine for executing view transformations using 4-valued partial models [c7; d21].
- *Contribution 2.2* I proposed a model generation approach that combines a DPLL-like decision procedure based on partial modeling with LP and ILP background solvers to synthesize and optimize models according to objective functions defined as linear programs, such as *multiplicity constraints*, *total model size*, and *cost functions*, using scoped partial models [j1].

Contribution group 3 I proposed analysis methods for the reliability and worst-case execution time estimation based on reasoning with partial models.

- *Contribution 3.1* I proposed a model-based technique for automatically deriving phased-mission stochastic Petri net models for complex reconfigurable systems based on fully compositional view transformations [c12; l18; r19].
- *Contribution 3.2* I proposed an approach for finding *witness models* of worst-case execution times of query-based runtime monitor programs in critical embedded systems by model generation using scoped partial models with linear programming [j5].

The rest of this thesis is structured as follows:

- Chapter 2 presents *Contributions 1.1* and *1.2* after reviewing some preliminary notions for domain modeling. The introduced concepts are illustrated using the **Interferometry Mission** case study from NASA JPL. The corresponding formal proofs are relegated to Appendix A.
- Chapter 3 presents the view transformation language and engine which comprise *Contribution 2.1*. The applicability and effectiveness of the approach is demonstrated using the open-source **Train Benchmark**.
- Chapter 4 presents the graph generator, which comprises *Contribution 2.2*. The applicability and effectiveness of the approach is demonstrated using **Interferometry Mission** case study, as well as two other case studies from model-driven engineering. The corresponding formal proofs are relegated to Appendix B.
- Chapter 5 presents the architecture-based construction of Phased-Mission Systems, which comprises *Contribution 3.1*. The applicability and effectiveness of the approach is demonstrated using the **Flexible Manufacturing System** case study.
- Chapter 6 presents witness model synthesis for WCET analysis, which comprises *Contribution 3.2*. The applicability and effectiveness of the approach is demonstrated using the **MoDeS3** case study. The corresponding formal proofs are relegated to Appendix C.
- Finally, Chapter 7 summarizes the proposed contributions and highlights possible avenues for future work.

Chapter 2

Formalisms for partial models

Quality assurance of critical software-intensive systems frequently relies on the automated synthesis of test data to reduce conceptual gaps in the test cases. When testing domain-specific modeling tools, or autonomous cyber-physical systems in model-based systems and software engineering scenarios, test data takes the form of typed and attributed graph models. Automated model generators are key technologies to address the needs of such testing scenarios.

A model generator needs to derive consistent models where each model needs to satisfy (or deliberately violate) a set of constraints captured in the form of OCL constraints or graph predicates. Logic solvers (like SMT-solvers, SAT-solvers, CSP-solvers) have been frequently providing precise foundations for such model generators in tools like Alloy, USE, UMLtoCSP, Formula, etc. However, a recently emerging family of model generators, like PLEDGE [SSB20] or the Viatra Solver [SNV18], addresses the consistent model generation challenge directly on the level of graph models by sophisticated search strategies (like multi-objective optimization or SAT-solving algorithms) and powerful abstractions provided by 3-valued partial models and partial model refinement [Var+18]. To fine-tune the model generation process, iterative and incremental approaches [SVV16] are proposed where models obtained as output in a previous run can be used as inputs (e.g. required or forbidden model fragments) in subsequent runs.

Unfortunately, partial modeling tools are limited in their support for reasoning with (i) *inconsistent information* (such as *conflicts* in model merging scenarios) and with (ii) *numerical information* about the attributes of the number of objects in a partial model. While SMT solvers may rely on SMT *bakcground theories* [Kat+16] for reasoning about numerical variable and the Nelson–Oppen [NO79] procedure for the combination of theories, support for numerical information in partial modeling remains limited. The combination of abstract domains [CH78; SPV18] and partial models [Gop+04; Mag+07; MRS10; FFJ12] was proposed for *value analysis* of pointer-manipulating programs [APV09].

However, the aforementioned techniques do not in themselves provide a sound and complete decision procedure (as in graph generation) for partial models with numerical values. In particular, there is presently no support for reasoning about the *size* of the possible solutions represented by a partial model, even though related techniques like *structural counter abstraction* [BCK01; Var+06; KK08; KK06; Ban+13] and *model-based quantifier instantiation* [Rey+13] are in use for the analysis of graph transformation systems and in SMT solvers, respectively.

This chapter presents the following contributions:

- We propose *4-valued partial models* based on the first-order, inconsistency-tolerant Belnap–Dunn 4-valued logic [Bel77; KO17] as semantic framework for possible partial (*paracomplete*) or inconsistent (*paraconsistent*) graph models. In addition to highlighting inconsistencies and conflicts between constraints during graph generation, such model can also

represent *merge conflicts* during relationship-based model merge operations [CNS12][c7]. As a benefit over merely marking merge conflicts, the precise semantics Belnap–Dunn logic enable evaluating model queries and reasoning over inconsistent models without encountering the *principle of explosion* of ordinary (2-valued) logic.

- We integrate *interval abstraction* [Min04; Kul09; JM09] into 4-valued partial models (in a manner similar to [FFJ12]) to store and reason about numerical attribute values in graph models. As a key novelty, the *error* (↯) logic value of Belnap–Dunn logic enables uniform treatment of calculations that fail at runtime (e.g., due to division by zero) as conflicts during model generation, integrating attribute constraints into the partial model based model generation framework [j2; j3].
- We propose *scoped partial models*, which use *polyhedron abstraction* [CH78; BHZ08] to store and reason about *linear* constraints over the size of models, such as *type scope* [Jac02] constraints and linear cost functions. We formulate *model generation* [j1] and *model optimization* [j5] tasks with linear constraints and linear objective functions.

As a result, partial models with (i) conflicts and (ii) numerical information can be described in a mathematically precise intermediate language along with their (structural and numerical) well-formedness constraints in a technology-independent way. These formalisms can serve as (a) *problem representations* as the input of model generators, (b) *internal representations* for exploring state spaces of partial models, e.g., using the abstract DPLL [NOT06; Bra+13] algorithm as a search strategy, and (c) *output representations* highlighting conflicts between requirements or as partial solutions in iterative (multi-step) model generation. A concrete syntax for serializing 4-valued partial models serving in these roles was proposed in [j2].

We describe reasoning techniques for 4-valued partial models and for scoped partial models in Chapters 3 and 4, respectively.

We demonstrate the formalisms on a sequence of examples adapted from an architecture synthesis case study proposed by engineers at the NASA Jet Propulsion Lab.

The contents of this chapter are based on the journal papers [j1; j2].

2.1 Modelling and metamodels

A large set of industrial modeling tools (including e.g., Capella, Artop, Yakindu, Papyrus, etc.) use DSLs as conceptual foundation. The specification of a DSL typically starts from defining a *metamodel* (MM) and a set of *well-formedness constraints* (WF). A metamodel defines the main concepts and relations in a domain imposing the basic graph structure of *instance models*. WF constraints further restrict consistent (or valid) instance models of the language by defining additional design rules. In this paper, we use the Eclipse Modeling Framework (EMF) [Ste+09] metamodels and Viatra well-formedness constraints [VB07; Ujh+15] as a technical foundation for domain modeling, which is also used in those industrial tools above as well as in [Her+17]. Conceptually, the graph generation approach could be applied on other modeling formalisms too, e.g. UML Class Diagrams for defining the types and Object Constraint Language (OCL, [OCL]) for defining constraints as in [SAB09; SSB17].

> **Example 2.1** The metamodel for interferometry constellation missions is shown in Fig. 2.1 using an EMF notation. An InterferometryMission consists of communicating CommElements (as *EClasses*), which are equipped with CommSubsys subsystems (i.e., antennas with different radio frequencies) through their subsys *EReferences* for Ka, X, and UHF bands.
> Spacecraft of different sizes, including cube satellites Cube3U and Cube6U, as well as small satellites SmallSat, may carry interferometry Payloads (photo sensors), and must be able to

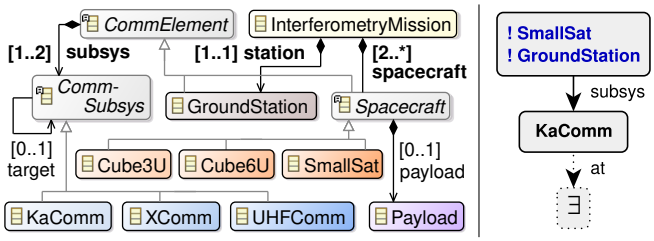


Figure 2.1: Example metamodel and WF constraint error pattern

reach the GroundStation via radio links (to send sensor data) denoted by the target references.

As a foundation for generating consistent models first, we need a precise formal framework to specify DSLs for which purpose we rely on [Sem+17; JLB11; SV17; Jac02; CCR07; FSC12].

2.1.1 Metamodels

The metamodel defines the main concepts and relations of the target domain.

Definition 2.1 (Metamodel) A *metamodel* corresponds to a *signature* $\langle \Sigma, \alpha \rangle$, where

- $\Sigma = \{\varepsilon, \sim, \mathsf{int}\} \uplus \Sigma_C \uplus \Sigma_R \uplus \Sigma_A \uplus \Sigma_F$ is the set of *logical symbols*;
- $\alpha \colon \Sigma \to \mathbb{N}$ is the *arity function*;
- ε is the *object existence symbol* with $\alpha(\varepsilon) = 1$;
- $\sim$ is the *object equality symbol* with $\alpha(\sim) = 2$;
- int is the *integer type symbol* with $\alpha(\mathsf{int}) = 1$;
- $\Sigma_C = \{C_1, \ldots, C_c\}$ is the set of *class symbols* with $\alpha(C_i) = 1$ for each $C_i \in \Sigma_C$;
- $\Sigma_R = \{R_1, \ldots, R_r\}$ is the set of *reference symbols* with $\alpha(R_i) = 2$ for each $R_i \in \Sigma_R$;
- $\Sigma_A = \{A_1, \ldots, A_a\}$ is the set of *attribute symbols* with $\alpha(A_i) = 2$ for each $A_i \in \Sigma_A$; and
- $\Sigma_F = \{F_1, \ldots, F_f\}$ is the set of *predicate symbols*.

To represent an EMF metamodel, we add a class symbol $C \in \Sigma_C$ for each *EClass*, a reference symbol $R \in \Sigma_R$ for each *EReference*, and an attribute symbols $A \in \Sigma_A$ for each *EAttribute*. The int symbol corresponds to the *EInt* datatype.

This formalism, in accordance with the EMF standard, handles references as relations: edges do not have identities and parallel edges of the same *EReference* are not allowed. Since our current work focuses on model generation for the structural part of graph models (i.e. nodes/objects and edges/links), we omit the detailed handling of attributes, which could be introduced similarly.

We also permit predicate symbols $F \in \Sigma_F$ of arbitrary arity $\alpha(F)$ to represent the results of model queries.

Additionally, we introduce partial modelling specific concepts: a unary predicate ε denoting the existence of an object (in a normal model, each object is existing), and a binary predicate $\sim$ denoting the equivalence of objects (in a normal model, all objects are different from each other).

A metamodel also imposes several structural constraints on instance models to enforce syntactic consistency for model manipulation or model persistence operations:

1. *Type Hierarchy (TH)* expresses that a more specific (child) class has every structural feature of the more general (parent) class;
2. *Type Compliance (TC)* requires that for any relation $R(o, t)$, its source and target objects o and t must have compliant types;
3. *Abstract (ABS)*: If a class is defined as abstract, it is not allowed to have direct instances;
4. *Multiplicity (MUL)* of structural features can be limited with upper and lower bound in the form of "lower..upper";
5. *Inverse (INV)* states that two parallel references of opposite direction always occur in pairs.

6. *Containment (CON)*: Instance models in EMF are expected to be arranged into a containment hierarchy, which is a directed tree along relations marked in the metamodel as containment (e.g., subsys or payload). The containment hierarchy is particularly relevant for serialization purposes.

2.1.2 Well-formedness constraints

In many industrial modeling tools, domain-specific WF constraints are defined by *error predicates* captured either as OCL invariants [OCL] or as graph patterns [NNZ00; VB07]. A major practical subclass of such constraints can be formalized using first-order logic with transitive closure [SV17; SNV18], which can be efficiently evaluated by underlying query engines like [Ujh+15] to validate models, or formally analyzed by model generators [Sem+17] to synthesize well-formed models. Later in this chapter, we will extend the logic below for specific modelling use-cases.

Definition 2.2 (Basic syntax of first-order graph predicates) A graph predicate φ is defined over a Σ vocabulary of a metamodel and an infinite set of (object) variables $\{v_1, v_2, \ldots\}$ using the following grammar rules:

$$\begin{aligned}
\varphi ::= {} & \mathsf{C}(v) \mid \mathsf{R}(v_1, v_2) && \textit{type and reference pred.}\\
\mid {} & v_1 \sim v_2 && \textit{equivalence}\\
\mid {} & \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 && \textit{logic connectives}\\
\mid {} & \exists v\colon \varphi \mid \forall v\colon \varphi && \textit{quantified expression}\\
\mid {} & \mathsf{F}^+(v_1, v_2) && \textit{transitive closure}
\end{aligned}$$

Informally, assuming that error patterns $\varphi_1, \ldots, \varphi_n$ are defined for a domain, a model is consistent (or valid), if it does not satisfy any error predicates $\varphi_i(v_1, \ldots, v_m)$, i.e.

$$\neg\exists v_1, \ldots, v_m\colon \varphi_i(v_1, \ldots, v_m) = \forall v_1, \ldots, v_m\colon \neg\varphi_i(v_1, \ldots, v_m).$$

Example 2.2 Error predicates $\varphi_1, \ldots, \varphi_8$ in the *satellite* case study capture the following design rules of interferometry missions.

- A CommElement may only have a single transmitting subsys (the other subsys, if present, may only receive):
$$\varphi_1(e) \coloneqq \exists c_1, c_2\colon \mathsf{subsys}(e, c_1) \wedge \mathsf{subsys}(e, c_2) \wedge c_1 \neq c_2 \wedge \big(\exists t\colon \mathsf{target}(c_1, t)\big) \wedge \big(\exists t\colon \mathsf{target}(c_2, t)\big).$$
- The GroundStation can only receive and may not have outgoing communication links:
$$\varphi_2(g) \coloneqq \exists c, t\colon \mathsf{GroundStation}(g) \wedge \mathsf{subsys}(g, c) \wedge \mathsf{target}(c, t).$$
- At least two different Spacecrafts must have the interferometry Payload configured:
$$\varphi_3 \coloneqq \forall s_1, s_2\colon \neg\big(\exists p\colon \mathsf{payload}(s_1, p)\big) \vee \neg\big(\exists p\colon \mathsf{payload}(s_2, p)\big) \vee s_1 = s_2.$$
- All Spacecraft must have a communication path (transitive closure of radio links) to the GroundStation:
$$\varphi_{\mathsf{link}}(s_1, s_2) \coloneqq \exists c_1, c_2\colon \mathsf{subsys}(s_1, c_1) \wedge \mathsf{subsys}(s_2, c_2) \wedge \mathsf{target}(c, s_2),$$
$$\varphi_4(s) \coloneqq \mathsf{Spacecraft}(s) \wedge \big(\forall g\colon \neg\mathsf{GroundStation}(g) \vee \neg\mathsf{link}^+(s, g)\big).$$
- There may be no communication loops, i.e., communication paths from a CommElement to itself:
$$\varphi_5(e) \coloneqq \mathsf{link}^+(e, e).$$

- CommSubsystems can only communicate if they use the same frequency band:

$$\varphi_6(c_1, c_2) \coloneqq \mathsf{target}(c_1, c_2) \wedge \neg\big(\mathsf{KaComm}(c_1) \wedge \mathsf{KaComm}(c_2)\big) \\ \wedge \neg\big(\mathsf{XComm}(c_1) \wedge \mathsf{XComm}(c_2)\big) \wedge \neg\big(\mathsf{UHFComm}(c_1) \wedge \mathsf{UHFComm}(c_2)\big).$$

- Cube3U satellites can only cross-link (send data to another satellite) using an UHFComm transmitter, but can only communicate with the GroundStation using a XComm transmitter:

$$\varphi_7(s) \coloneqq \exists c_1, c_2, e\colon \mathsf{Cube3U}(s) \wedge \mathsf{subsys}(s, c_1) \wedge \mathsf{subsys}(e, c_2) \wedge \mathsf{target}(c_1, c_2) \\ \wedge \neg\big(\mathsf{UHFComm}(c_1) \wedge \mathsf{Spacecraft}(e)\big) \\ \wedge \neg\big(\mathsf{XComm}(c_1) \wedge \mathsf{GroundStation}(e)\big).$$

- Only a SmallSat or the GroundStation may be configured with a KaComm subsystem:

$$\varphi_8(e) \coloneqq \big(\exists s\colon \mathsf{subsys}(e, s) \wedge \mathsf{KaComm}(s)\big) \wedge \neg\mathsf{SmallSat}(e) \wedge \neg\mathsf{GroundStation}(e).$$

The error predicate φ_8 is depicted on the right side of Figure 2.1 as a graph pattern using the graphical syntax of the GROOVE graph transformation tool [Ren+12].

Since the structural constraints on metamodels can be formalized as WF constraints [Sem+17] using the graph predicate language of [SV17; SNV18], we can evaluate both kinds of constraints uniformly with first-order logic. However, as structural constraints are prevalent in modeling tasks, in the following, we will exploit their special structure, especially that of *MUL* and *CON* constraints, to speed up model generation by numerical reasoning, while retaining full support for arbitrary WF constraints.

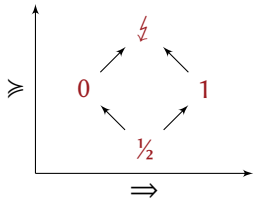
2.1.3 Type scopes

To guide model generation towards more relevant models in a domain, type scopes are frequently used to specify the number of required elements of each type (class). For example, Alloy [Jac02] introduces *scope bounded* analysis for relational specifications. For larger models, prescribing lower and upper bounds may ensure realistic distribution of types in auto-generated test cases and benchmarks.

Type scope constraints define lower and upper bounds for the number of instances generated for a specific class. A *lower type scope constraint* $L_i \leq \mathsf{C}_i$ and an *upper type scope constraint* $\mathsf{C}_i \leq U_i$ respectively assert that there are at least L_i and at most U_i instances of the class C_i (where $L_i, U_i \in \mathbb{N}$). We require that a generated model must satisfy the conjunction of all scope constraints of a given type.

Test and benchmark generation tasks require models of some finite size n, whereas for proving the inconsistency of modeling languages, cases up to a small size n are checked according to the small scope assumption [Jac02]. Therefore, we assume the existence of an upper bound n on the number of objects in the generated models, which can be seen as a type scope bound on a common supertype of all types.

Our formulation of type scopes extends the notation of scopes introduced in Alloy [Jac02], which supports only upper ($\mathsf{C}_i \leq U_i$) and exact limits ($\mathsf{C}_i = E_i$) (but not lower bounds). Alloy also limits type scopes and type hierarchy. If a type scope is specified for a class C_i, its supertypes cannot have a type scope. Scopes in Alloy cannot express problems where the sums of (upper) type scope bounds do not coincide with the number of objects ($\sum_i U_i > n$), because the model size n can only be specified as a type scope bound on the common supertype of all types. Therefore, these upper scope constraints and all lower scope constraints need to be formulated as additional constraints instead.



$$(X \geqslant Y) \Leftrightarrow [(X = ½) \vee (X = Y) \vee (Y = ⚡)]$$
$$(X \Rightarrow Y) \Leftrightarrow [(X = 0) \vee (X = Y) \vee (Y = 1)]$$

(a) Lattice of logical values

X	$\neg^4 X$
0	1
1	0
½	½
⚡	⚡

$\vee^4$	0	1	½	⚡
0	0	1	½	⚡
1	1	1	1	1
½	½	1	½	1
⚡	⚡	1	1	⚡

$\wedge^4$	0	1	½	⚡
0	0	0	0	0
1	0	1	½	⚡
½	0	½	½	0
⚡	0	⚡	0	⚡

$\oplus$	0	1	½	⚡
0	0	⚡	0	⚡
1	⚡	1	1	⚡
½	0	1	½	⚡
⚡	⚡	⚡	⚡	⚡

(b) Logic connectives and information merge

Figure 2.2: Belnap–Dunn 4-valued logic

Example 2.3 Given the constraints $30 \leq$ Spacecraft, Spacecraft ≤ 50, and SmallSat ≤ 15 for our running example, generated models may contain between 30 and 50 Spacecrafts. Moreover, at most 15 of these spacecrafts can be SmallSats.

2.2 4-valued partial models with attributes

2.2.1 Belnap–Dunn 4-valued logic

In this work, we utilize 4-valued logic to explicitly represent unfinished, partial (*paracomplete*) models, as well as errors and inconsistencies (*paraconsistency*) arising during the evaluation of computations over such models. This section provides semantic foundations for our specification language based on the *inconsistency-tolerant* Belnap–Dunn 4-valued logic [Bel77; KO17]. In addition to reasoning about design decisions yet to be made (i.e., currently unknown outcomes) and contradiction arising during e.g., model merging, 4-valued logic allows catching runtime errors caused by undefined arithmetic operations with attributes, such as division by zero [MW12].

Belnap–Dunn 4-valued logic contains the usual false 0 and true 1 truth values, the *unknown* ½ value introduced for unspecified or unknown properties, and the *inconsistent* ⚡ value that signals inconsistencies and computation failures. The subset $\{0, 1, ½\}$ of logic values can express partial, but consistent information. Conversely, the subset $\{0, 1, ⚡\}$ expresses possibly inconsistent, but complete information.

Two partial orders can be defined over 4-valued logic values (Figure 2.2a). The *information order*, denoted by $\geqslant$, expresses the gathering of information as new facts are learned during the refinement of partial models. Facts with ½ logical value can be set to either 1 or 0, while a change to ⚡ signifies an inconsistency or failure.

The second partial order is the *implication order*, denoted by $\Rightarrow$, which serves as a generalization of logical implication.

The *information merge* operator $\oplus$ merges 4-valued truth values where contradictory information results in ⚡. Other operations on 4-valued truth values $\neg^4$, $\vee^4$, and $\wedge^4$ are extensions of the usual logic operators $\neg$, $\vee$, and $\wedge$. Their truth tables (see Figure 2.2b) coincide with their classical counterparts for $\{0, 1\}$ inputs.

Semantically, ½ truth value represents potential 1 or 0 (or ⚡) values, and the semantic is chosen to cover all of those options. On the other hand, ⚡ is often unintuitive, but it allows the precise and explicit localization of inconsistencies within models [Bel77; CNS12]. For example,

we may see that if $X = \lightning$ and $Y = ½$, then $X \vee^4 Y = 1$, because the *only* way for our logical inference to result in a consistent truth value is to eventually learn that Y is 1. Should Y turn out to be 0, the inconsistent $\lightning$ value will be propagated.

2.2.2 Interval arithmetic

We use interval arithmetic to represent numerical attribute values, including unfinished or inconsistent ones. A closed, possibly infinite interval $iv \in \mathbb{IV} \subsetneq 2^{\mathbb{N}}$ of integers denotes a set of possible numerical values. The empty interval $\emptyset \in \mathbb{IV}$ denotes a missing value or a result of failed computation.

The operators $+^\sharp$, $-^\sharp$, $\cdot^\sharp$, $/^\sharp$, $\uparrow^\sharp$, $\Sigma^\sharp$, $\min^\sharp$ and $\max^\sharp$ refer to the interval arithmetic [Kul09] versions of the usual + (addition), − (subtraction), · (multiplication), / (division), ↑ (exponentiation), Σ (summation), min and max operations over integers, respectively. The ⊔ symbol denotes the *join* (i.e., the smallest interval containing both intervals) of two intervals, while ∩ denotes interval intersection.

2.2.3 Partial models

In this paper, we introduce the concept of *4-valued partial models* as an extension of partial models proposed in [SNV18]. The goal of partial models is to explicitly represent uncertainty in models, thus a single partial model represents a set of potential (traditional) instance models. We combine two techniques to capture uncertainty in a partial model.

First, *4-valued logic* is used to explicitly represent uncertain *structural properties* of models with the ½ truth value in accordance with [RSW04; Var+18; SV17]. From a formal perspective, this technique implements *predicate abstraction* on graph models [FFJ12].

Secondly, *numerical information* is attached to the partial model to represent attribute values with *interval abstraction*. Since intervals are a *non-relational abstraction* [Min04; JM09], the partial model does not store any information about the relationships between different attribute values. This facilitates fast local model querying and reasoning by evaluation numerical constraints with interval arithmetic. Nevertheless, reasoning over the relationships of attribute values still remains possible by invoking external solvers [c9; j3].

The fourth $\lightning$ (error) truth value can explicitly mark constraint violations and conflicting information in models, for example, during model merge [CNS12]. For attribute values, the empty interval $\emptyset$ can be used analogously to signify lack of any valid numerical value.

Definition 2.3 A *4-valued partial model* over a metamodel signature $\langle \Sigma, \alpha \rangle$ is a tuple $P = \langle O_P, \mathcal{I}_P, \mathcal{V}_P \rangle$, where

- O_P is a finite set of *individuals* (i.e., objects);
- $\mathcal{I}_P$ gives a *4-valued interpretation* $\mathcal{I}_P(\varsigma) \colon O_P^{\alpha(S)} \to \{1, 0, ½, \lightning\}$ for each symbol $\varsigma \in \Sigma$;
- $\mathcal{V}_P \colon O_P \to \mathbb{IV}$ gives a *numeric value interpretation* for all individuals.

Informally, the meaning of the structural 4-valued interpretation $\mathcal{I}_P$ is as follows:

- *Node (class) symbols:* $\mathcal{I}_P$ gives a 4-valued interpretation of each class symbol C_i in Σ: $\mathcal{I}_P(\mathrm{C}_i) \colon O_P \to \{1, 0, ½, \lightning\}$ that describes whether it is true, false, unspecified, or inconsistent if an object is an instance of a class C_i.
- *Integer type symbol:* $\mathcal{I}_P$ gives a 4-valued interpretation $\mathcal{I}_P(\mathrm{int}) \colon O_P \to \{1, 0, ½, \lightning\}$ of int: $\mathcal{I}_P(\mathrm{int})(x) = 1$ and ½ means that x is surely or possibly a *data object* representing an integer value, respectively. The rest of the individuals in the model are *domain objects*. In EMF terminology, data objects correspond to *EInt* instances, while domain objects are *EObject* instances.

- *Attribute symbols:* $\mathcal{I}_P$ gives a 4-valued interpretation of each attribute symbol A_j in Σ: $\mathcal{I}_P(\mathsf{A}_j) \colon O_P \times O_P \to \{1, 0, ½, ↯\}$, where $\mathcal{I}_P(\mathsf{A}_j)(x, y) = 1$, 0, $½$, and $↯$ mean that it is true, false, unknown, or inconsistent whether the value of the A_j attribute of the x domain objects is represented by the y data object. The numeric value interpretation $\mathcal{V}_P(y)$ gives the interval of possible integer attribute values that the domain object y can represent.
- *Predicate symbols:* $\mathcal{I}_P$ gives a 4-valued interpretation of each predicate symbol F_j in Σ: $\mathcal{I}_P(\mathsf{A}_j) \colon O_P \times O_P \to \{1, 0, ½, ↯\}$, where $\mathcal{I}_P(\mathsf{F}_j)(x_1, \ldots, x_{\alpha(\mathsf{F}_j)}) = 1$, 0, $½$, and $↯$ mean that it is true, false, unknown, or inconsistent whether the predicate represented by F_j holds for the tuple of objects $\langle x_1, \ldots, x_{\alpha(\mathsf{F}_j)}\rangle$. Later in this chapter, we will associate a logic formula φ_j with each predicate symbol F_j. Thus, the interpretation $\mathcal{I}_P(\mathsf{F}_j)(x_1, \ldots, x_{\alpha(\mathsf{F}_j)})$ can store the (prescribed) value of the evaluation of the formula φ_j after its free variables are bound to the objects $x_1, \ldots, x_{\alpha(\mathsf{F}_j)}$.
- *Existence symbol:* $\mathcal{I}_P$ gives a 4-valued interpretation $\mathcal{I}_P(\varepsilon) \colon O_P \to \{1, 0, ½, ↯\}$ of the existence symbol ε: $\mathcal{I}_P(\varepsilon)(x) = 1$ and $½$ means certain or possible existence of object x. In particular, $\mathcal{I}_P(\varepsilon)x = ½$ turns x into an *optional* object that can be freely included in or omitted from the model.
- *Equality symbol:* $\mathcal{I}_P$ also gives 4-valued interpretation $\mathcal{I}_P(\sim) \colon O_P \times O_P \to \{1, 0, ½, ↯\}$ of equality symbol $\sim$: $\mathcal{I}_P(\sim)(x, y) = 1$, 0, $½$, and $↯$ mean that it is true, false, unknown, or inconsistent whether x and y are equal. In particular, self-loops with $\mathcal{I}_P(\sim)(x, x) = ½$ turns x a *multi-object* that can be *split* into multiple, non-equal objects to compactly represent a number of concrete model objects. Conversely, potentially equal objects $\mathcal{I}_P(\sim)(x, y) = ½$ can be *merged* into single objects.

In the context of model generation, we restrict the possible combination of those predicates to exclude inconsistent and irrelevant constructs that are not productive as intermediate states of model generation. However, such inconsistent constructs can still arise during model merge operations when there are merge conflicts.

Structural regularity criteria avoid unnecessarily storing nonexistent objects and take care of the expected properties of object equality $\sim$.

Definition 2.4 A 4-valued partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P\rangle$ is *structurally regular* if

- there are no *nonexistent objects*, i.e., if $o \in O_P$, then $\mathcal{I}_P(\varepsilon)(o) \neq 0$;
- $\sim$ is *reflexive*, i.e., if $o \in O_P$, then $\mathcal{I}_P(\sim)(o, o) \neq 0$;
- $\sim$ is *symmetric*, i.e., if $o_1, o_2 \in O_P$, then $\mathcal{I}_P(\sim)(o_1, o_2) = \mathcal{I}_P(\sim)(o_2, o_1)$; and
- there are no *object merges*, i.e., if $o_1, o_2 \in O_P$ and $\mathcal{I}_P(\sim)(o_1, o_2) \neq 0$, then $o_1 = o_2$.

Numerical regularity criteria ensure that attribute values are represented by value objects.

Definition 2.5 A scoped partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P\rangle$ is *numerically regular* if

- *domain objects* have no numerical value, i.e., if $o \in O_P$ and $\mathcal{I}_P(\mathsf{int})(p) \leqslant 0$, then we have $\mathcal{V}_P(o) = \emptyset$; and
- *value objects* have at least one possible numerical value, i.e., if $o \in O_P$ and we have $\mathcal{I}_P(\mathsf{int})(p) = 1$, then $\mathcal{V}_P(o) \neq \emptyset$.

Definition 2.6 A 4-valued partial model is *regular* if it is both structurally regular and numerically regular.

2.2.4 Refinement and concretization

We carry out model generation along a sequence of refinement steps that derive new partial models by increasing their size but gradually reducing the level of uncertainty in each model while continuously checking (an approximated version of) well-formedness constraints. Thus, we introduce the formal concept of refinement for 4-valued partial models which simultaneously refines both the 4-valued logic structure and the numerical attribute values.

Definition 2.7 The relation $ref \subseteq O_P \times O_Q$ is a *refinement* from the 4-valued partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{V}_P \rangle$ to the scoped partial model $Q = \langle O_Q, \mathcal{I}_Q, \mathcal{V}_Q \rangle$ over the same signature $\langle \Sigma, \alpha \rangle$, written as $P \succcurlyeq_{ref} Q$, if

VR1. no surely existing object disappears, i.e., if $p \in O_P$ and $\mathcal{I}_P(\varepsilon)(p) \preccurlyeq 1$, then there is some successor $q \in O_Q$ such that $\langle p, q \rangle \in ref$ and $\mathcal{I}_P(\varepsilon)(p) \succcurlyeq \mathcal{I}_Q(\varepsilon)(q)$;
VR2. no possibly existing objects are created from scratch, i.e., if $q \in O_Q$ and $\mathcal{I}_Q(\varepsilon)(q) \succcurlyeq 1$, then there is some predecessor $p \in O_P$ such that $\langle p, q \rangle \in ref$ and $\mathcal{I}_P(\varepsilon)(p) \succcurlyeq 1$;
VR3. *ref* obeys the *information ordering of 4-valued interpretations*, i.e., if $\varsigma \in \Sigma \setminus \{\varepsilon\}$ and $\langle p_1, q_1 \rangle, \dots, \langle p_{\alpha(\varsigma)}, q_{\alpha(\varsigma)} \rangle \in ref$, then $\mathcal{I}_P(\varsigma)(p_1, \dots, p_{\alpha(\varsigma)}) \succcurlyeq \mathcal{I}_Q(\varsigma)(q_1, \dots, q_{\alpha(\varsigma)})$; and
VR4. *ref* obeys the *information ordering of numerical value interpretations*, i.e., if $\langle p, q \rangle \in ref$, then $\mathcal{V}_P(p) \supseteq \mathcal{V}_Q(q)$.

We will write $P \succcurlyeq Q$ if we have $P \succcurlyeq_{ref} Q$ for some refinement *ref*.

If a 4-valued partial model M only contains 1 and 0 logic values and singleton attribute values (i.e., each attribute value is a single integer), then M represents a traditional (concrete) *instance model*. This corresponds to resolving all uncertainties while still having no inconsistency in the model. We can use concrete partial models to represent completed system architectures where there are no further design decisions to be made.

Definition 2.8 A regular 4-valued partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{V}_P \rangle$ is *concrete* if

- $\mathcal{I}_P$ contains 1 and 0 values only, i.e., for each $\varsigma \in \Sigma \cap \Gamma$ and $o_1, \dots, o_{\alpha(\varsigma)} \in O_P$, we have $\mathcal{I}_P(\varsigma)(o_1, \dots, o_{\alpha(\varsigma)}) \in \{1, 0\}$; and
- each value object has a single possible numerical, i.e., if $o \in O_P$ and $\mathcal{V}_P(o) = \{k\}$ for some $k \in \mathbb{Z}$.

Because the refinement of 4-valued partial models is transitive, we may obtain concrete model M from the *initial* partial model P_0 after a chain of refinements $P_0 \succcurlyeq P_1 \succcurlyeq \cdots \succcurlyeq M$. Such a refinement chain will be constructed during model generation or synthesis.

Conversely, in model merge operations, chains of refinements composed of *merge functions* will be created to include information from the individual model fragments to be merged, as well as to ensure that the concrete model obtain after merging satisfies the metamodel constraints.

Proposition 2.9 Refinement of 4-valued partial models is transitive, i.e., if P, Q, R are regular partial models, $P \succcurlyeq_{ref_1} Q$, and $Q \succcurlyeq_{ref_2} R$, then $P \succcurlyeq_{ref_3} R$ for some refinement ref_3.

Note that if a partial model contains an inconsistent $\lightning$ logic value or a data object with an empty $\emptyset$ set of represented attribute values, obtaining a concrete model by refinement is only possible if the affected objects can be removed outright.

2.2.5 Compatibility and inconsistency of partial models

As for logic to define well-formedness constraints and graph predicate, we extend the basic logical language from Definition 2.2 with numerical expressions over attribute values.

The resulting semantics contain not only the evaluation $[\![\varphi]\!]^P_Z$ for logic formulas φ, but also an evaluation $(\!|\mu|\!)^P_Z$ for numerical expressions. Similarly to the storage of information in partial models, we use 4-valued logic abstraction to encode the results of logic formula evaluation while relying on interval abstraction for the results of numerical expressions.

Definition 2.10 The *semantics* $[\![\varphi]\!]^P_Z \in \{1, 0, ½, ↯\}$ of a logic formula φ and $(\!|\mu|\!)^P_Z \in \mathbb{IV}$ of a numerical expression μ with free variables $V = \{v_1, \ldots, v_k\}$ over the 4-valued partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{V}_P \rangle$, the signature $\langle \Sigma, \alpha \rangle$, and the variable binding $Z : V \to O_P$ are inductively defined as

$$
\begin{aligned}
[\![\varsigma(v_1, \ldots, v_{\alpha(\varsigma)})]\!]^P_Z &= \mathcal{I}_P(\varsigma)(Z(v_1), \ldots, Z(v_{\alpha(\varsigma)})) \text{ for all } \varsigma \in \Sigma \setminus \{\varepsilon, \sim\},\\
[\![v_1 \sim v_2]\!]^P_Z &= \mathcal{I}_P(\sim)(Z(v_1), Z(v_2)),\\
[\![\neg\varphi_1]\!]^P_Z &= \neg^4 [\![\varphi_1]\!]^P_Z,\\
[\![\varphi_1 \vee \varphi_2]\!]^P_Z &= [\![\varphi_1]\!]^P_Z \vee^4 [\![\varphi_2]\!]^P_Z,\\
[\![\varphi_1 \wedge \varphi_2]\!]^P_Z &= [\![\varphi_1]\!]^P_Z \wedge^4 [\![\varphi_2]\!]^P_Z,\\
[\![\exists v : \varphi_1]\!]^P_Z &= \bigvee_{o \in O_P}^4 \mathcal{I}_P(\varepsilon)(o) \wedge^4 [\![\varphi_1]\!]^P_{Z, v \mapsto o},\\
[\![\forall v : \varphi_1]\!]^P_Z &= \neg^4 [\![\exists v : \neg\varphi_1]\!]^P_Z,\\
[\![\varsigma^+(v_1, v_2)]\!]^P_Z &= \left[\!\!\left[\varsigma(v_1, v_2) \vee \bigvee_{i=1}^{|O_P|} \exists u_1 : \cdots \exists u_i : \varsigma(v_1, u_1) \wedge \bigwedge_{j=1}^{i-1} \varsigma(u_j, u_{j+1}) \wedge \varsigma(u_i, v_2) \right]\!\!\right]^Z_P\\
&\qquad\qquad \text{for all } \varsigma \in \Sigma \setminus \{\sim\} \text{ with } \alpha(\varsigma) = 2,\\
[\![\mu \in iv]\!]^P_Z &= \begin{cases} 1 & \text{if } (\!|\mu|\!)^P_Z \neq \emptyset \text{ and } (\!|\mu|\!)^P_Z \subseteq iv, \\ 0 & \text{if } (\!|\mu|\!)^P_Z \neq \emptyset \text{ and } (\!|\mu|\!)^P_Z \cap iv = \emptyset, \end{cases} \quad \begin{array}{ll} ↯ & \text{if } (\!|\mu|\!)^P_Z = \emptyset, \\ ½ & \text{otherwise,} \end{array}\\
(\!|\mathit{literal}|\!)^P_Z &= \{\mathit{literal}\} \text{ for all } \mathit{literal} \in \mathbb{N},\\
(\!|v|\!)^P_Z &= \mathcal{V}_P(Z(v)),\\
(\!|\mu_1 \langle op \rangle \mu_2|\!)^P_Z &= (\!|\mu_1|\!)^P_Z \langle op \rangle^\sharp (\!|\mu_2|\!)^P_Z \text{ for } \langle op \rangle = +, -, \cdot, /, \text{ or } \uparrow.
\end{aligned}
$$

We will use Φ_Σ to denote the set of all possible logic formulas over the signature $\langle \Sigma, \alpha \rangle$. If $[\![\varphi]\!]^P_Z \preccurlyeq 1$, then Z is a *match* of φ in P.

The newly added $\mu \in iv$ operator allows us to check whether the result of a numerical expression lies in a given interval. This also allows expressions equality ($\mu = k$) and inequality ($\mu \leq k$ and $\mu \geq k$) constraints by setting $iv = \{k\}$, $(-\infty, k]$, and $[k, \infty)$, respectively.

Numerical expressions are formed with the usual $+$, $-$, $\cdot$, $/$, and $\uparrow$ numerical operators, which are interpreted in accordance with interval arithmetic rules. They also contain *literal* expressions, standing for singleton interval, and references to first-order variables v, standing for the numerical value $\mathcal{V}_P(o)$ of the individual o bound to the variable v.

Example 2.4 The logic formula $\forall v_2 : \neg\mathsf{missionTime}(v_1, v_2) \wedge v_1 \in [0; 10]$ evaluates to 1 if the value of the missionTime attribute of the individual bound to the variable $v1$ surely lies between 0 and 10.

Notice that, on a concrete 4-valued model M, the result $[\![\varphi]\!]^M_Z$ of a logic formula is either 1, 0, or ↯, while the result $(\!|\mu|\!)^M_Z$ of a numerical expression is either a singleton interval $\{k\}$ for some $k \in \mathbb{Z}$, or the empty interval $\emptyset$. Empty intervals and ↯ logic values as introduced when and invalid arithmetic operation (e.g., division by zero) happens according to interval arithmetic rules, or the result of an invalid operation is compared with another interval.

The semantics of both logic formulas and numerical expressions obey 4-valued partial model

refinement. If more information is incorporated into the partial model, the results of expression evaluation get more precise as well. In particular, refining a partial model might change the results of formulas from ½ to 1 or 0 and tighten the interval bounds of the results of numerical expressions.

Lemma 2.11 The semantics of expressions obeys partial model refinement, i.e., if $P = \langle O_P, \mathcal{I}_P, \mathcal{V}_P \rangle$ and $Q = \langle O_Q, \mathcal{I}_Q, \mathcal{V}_Q \rangle$ are 4-valued partial models over the signature $\langle \Sigma, \alpha \rangle$, $P \succcurlyeq_{ref} Q$ for some refinement ref, φ is a logic formula (resp. μ is a numerical expression) with free variables $V = \{v_1, \ldots, v_k\}$, $Z \colon V \to O_P$ and $Y \colon V \to O_Q$ are variable bindings, and $\langle Z(v), Y(v) \rangle \in ref$ for all $v \in V$, then $[\![\varphi]\!]^P_Z \succcurlyeq [\![\varphi]\!]^Q_Y$ (resp. $(\!|\mu|\!)^P_Z \supseteq (\!|\mu|\!)^Q_Y$).

To incorporate the definitions of graph predicates, we define a respective *theory* (set of axioms). The theory associates each predicate symbol $\mathsf{F}_j \in \Sigma_\mathsf{F}$ with a logic formula that provides its meaning. As a convenience feature, the theory can also contain *error formulas*, so we do not have to add a fresh predicate symbol F_j for every well-formedness constraint.

Definition 2.12 A *theory* for the signature $\langle \Sigma, \alpha \rangle$ is a pair $\mathcal{T} = \langle d, \mathcal{E} \rangle$, where

- $d \colon \Sigma_\mathsf{F} \to \Phi_\Sigma$ is the *predicate definition function*, which assigns a formula $f(\mathsf{F}_j)$ with free variables $\{v_1, \ldots, v_{\alpha(\mathsf{F}_j)}\}$ to each predicate symbol $\mathsf{F}_j \in \Sigma_\mathsf{F}$; and
- $\mathcal{E} \subseteq \Phi_\Sigma$ is the set of *error formulas*.

As such, we can focus only on semantic interpretations of partial models which are compatible with the actual definitions. For example, if a particular logic formula computes a 4-valued value as a result, then the interpretation of the respective predicate symbol $\mathsf{F}_j \in \Sigma_\mathsf{F}$ stored in the partial model should actually retrieve the respective value.

Definition 2.13 A 4-valued partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{V}_P \rangle$ over the signature $\langle \Sigma, \alpha \rangle$ is *compatible* with the theory $\mathcal{T} = \langle d, \mathcal{E} \rangle$, if

- for all $\mathsf{F}_j \in \Sigma_\mathsf{F}$ and $o_1, \ldots, o_{\alpha(\mathsf{F}_j)} \in O_P$, we have

$$[\![d(\mathsf{F}_j)]\!]^P_{v_1 \mapsto o_1, \ldots, v_{\alpha(\mathsf{F}_j)} \mapsto o_{\alpha(\mathsf{F}_j)}} \oplus \mathcal{I}_P(\mathsf{F}_j)(o_1, \ldots, o_{\alpha(\mathsf{F}_j)}) \neq \lightning ; \text{ and}$$

- for all $\varphi \in \mathcal{E}$ with free variables $V = \{v_1, \ldots, v_m\}$ and for all valuations $Z \colon V \to O_P$, we have $[\![\varphi]\!]^P_Z \succcurlyeq 0$.

If some of these constraints are violated, we say that P is *inconsistent* with the theory $\mathcal{T}$.

Note that error patterns can be substituted with predicate symbols without a loss of expressiveness. We may remove the error formula $\varphi_i \in \mathcal{E}$ from $\mathcal{E}$ and instead add a new predicate symbol F_i to Σ with $d(\mathsf{F}_i) = \varphi_i$. If φ_i has free variables $V = \{v_1, \ldots, v_m\}$ then we may set $\mathcal{I}_P(\mathsf{F}_i)(o_1, \ldots, o_k)$ for each $o_1, \ldots, o_k \in O_P$ to ensure that P is compatible with the modified theory if and only if it was compatible with the original one.

2.2.6 Model generation problems

The generation of consistent (concrete) models is driven by a series of refinement and concretization steps where the level of uncertainty in partial models is gradually reduced. When all uncertainties are resolved and there is still no inconsistency in the model then a concrete model is obtained.

Next, we formally define the task of (consistent) model generation along partial models. Given a model generation task, a *complete model generator* outputs some model $Q \in \mathit{solutions}(P, T)$ if

solutions(P, T) is non-empty. Otherwise, it provides a proof of the unsatisfiability of the task.

Definition 2.14 Given a 4-valued partial model $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{V}_P \rangle$ over the signature $\langle \Sigma, \alpha \rangle$ and a theory $\mathcal{T} = \langle d, \mathcal{E} \rangle$ over the same signature, the solutions of the corresponding *model generation problem*,

$$\mathit{solutions}(P, \mathcal{T}) = \{M \mid P \succcurlyeq M, M \text{ is compatible with } \mathcal{T}\},$$

are the concrete refinements of P that are compatible with $\mathcal{T}$.

If a partial model P is inconsistent, then its refinements Q ($P \succcurlyeq Q$) cannot be concrete. Hence, when searching for a concrete, consistent refinement of P, model generators can abandon inconsistent partial models Q without compromising the completeness of model generation [SNV18].

2.3 Scoped partial models

We introduce the concept of *3-valued scoped partial models* as an extension of partial models proposed in [SNV18] to precisely keep track of the *size* of the partial model and the *number of represented graph predicate matches*. This allows reasoning about scope constraints and *linear cost functions* (computed as a linear combination of predicate match counts) efficiently.

Firstly, use *3-valued logic* is used to explicitly represent uncertain *structural properties* of models with a third ½ (unspecified or unknown) truth value (besides 1 and 0, which stand for *true* and *false*) in accordance with [RSW04; Var+18; SV17]. This is a subset of the *4-valued logic* used in Section 2.2 with the inconsistent ↯ logic value removed.

Secondly, *quantitative information* is attached to the partial model in the form of a *system of linear inequalities* to precisely represent the *known (or required) size* of the models and predicate match sets. Later, we use partial models as states of model generation to represent intermediate solutions with uncertain parts denoted with truth-value ½ and its size. The resulting *polyhedron abstraction* [BHZ08] can express relationships between arbitrary linear combinations of match set sizes, i.e., it is a *fully relational* abstract domain [JM09]. We exploit this property to efficiently reason about type scope constraints over *type hierarchies*, which is a feature not supported by the type scopes implemented in Alloy [Jac02].

2.3.1 Systems of linear inequalities and linear programs

First, we review some notations for systems of linear inequalities and ILP problems that will be used in this work.

Definition 2.15 Let $\mathcal{X} = \{\mathfrak{x}_1, \ldots, \mathfrak{x}_{|\mathcal{X}|}\}$ be a finite set of *linear equation variables*. Then $\mathcal{S} = \left\{\sum_{\mathfrak{x}_j \in \mathcal{X}} a_{i,j} \cdot \mathfrak{x}_j \leq y_i\right\}_{i=1}^{|\mathcal{S}|}$ is a *system of linear inequalities* on $\mathcal{X}$.

While Definition 2.15 only considers linear constraints with the $\leq$ operator. We will occasionally use constraints also with the $\geq$ and $=$ operators, which can be transformed into $\leq$ constraints in a straightforward manner by rearranging (and, in the case of $=$ constraints, splitting into two separate inequality constraints).

Definition 2.16 Let functions $k \colon \mathcal{X} \to \mathbb{N}$ be *valuations*. A valuation k is a *solution* of $\mathcal{S}$ (written as $k \vDash \mathcal{S}$) if $\sum_{\mathfrak{x}_j \in \mathcal{X}} a_{i,j} \cdot k(\mathfrak{x}_j) \leq y_i$ for all inequalities $\sum_{\mathfrak{x}_j \in \mathcal{X}} a_{i,j} \cdot \mathfrak{x}_j \leq y_i$ in $\mathcal{S}$. The system of linear equations $\mathcal{S}_1$ *entails* $\mathcal{S}_2$ (written as $\mathcal{S}_1 \vDash \mathcal{S}_2$) if, for any solution $k \vDash \mathcal{S}_1$, we also have $k \vDash \mathcal{S}_2$.

Note that we restrict our attention to *nonnegative* valuations (i.e., the value $k(\mathfrak{x})$ of each linear equation variable $\mathfrak{x} \in \mathcal{X}$ is a natural number), since we will use linear equation variables to represent model size and the number of graph predicate matches only.

Definition 2.17 An *integer linear program* (ILP) is of the form $\max \sum_{\mathfrak{x}_i \in \mathcal{X}} c_i \cdot \mathfrak{x}_i$ *subject to* $\mathcal{S}$, where $\mathcal{S}$ is a system of linear equations.

While ILP with both maximization (max) and minimization (min) objectives can be considered, we restrict our attention to max objectives w.l.o.g., since any min objective can be transformed into a max objective after multiplication by -1.

Definition 2.18 Let $g(k) = \sum_{\mathfrak{x}_i \in \mathcal{X}} c_i \cdot k(\mathfrak{x}_i)$ denote the *cost* of valuation k. The valuation k^* is an *optimal solution* of the ILP if $k^* \vDash \mathcal{S}$ and $g(k^*) \geq g(k)$ for all $k \vDash \mathcal{S}$, i.e., its cost is maximal.

2.3.2 Scoped signatures and partial models

We define *scoped signatures* that specify a subset $\Gamma_C \uplus \Gamma_R \uplus \Gamma_F \subsetneq \Sigma$ of class, reference, and predicate symbols that are subject to *fully relational abstraction*. The distinction between Σ and Γ allows us to selectively apply *polyhedron abstraction* using linear inequalities to only a subset of the relations in the model. This enables powerful abstraction for logical symbols where the added precision can greatly aid in model analysis and generation, while we can avoid the maintenance of linear constraints for logical symbols where they do not improve the precision of the abstraction or pose too large of an overhead.

For the sake of brevity, we omit the handling of attribute symbols A_j and the integer type symbol int. Therefore, in a scoped partial model, all objects are domain objects.

However, to enable the tracking of arbitrary linear inequalities alongside the scope partial model, we introduce *numerically tracked symbols* $\Sigma \not\ni \mathsf{X}_j \in \Gamma_X$ that are only considered by polyhedron abstraction and not by 3-valued logic abstraction.

Definition 2.19 (Scoped signature) A *scoped signature* is a tuple $\langle \Sigma, \Gamma, \alpha \rangle$, where

- $\Sigma = \{\varepsilon, \sim\} \uplus \Sigma_C \uplus \Sigma_R \uplus \Sigma_F$ is the set of *logical symbols*;
- $\Gamma = \{\varepsilon\} \uplus \Gamma_C \uplus \Gamma_R \uplus \Gamma_F \uplus \Gamma_X$ is the set of *numerical symbols*;
- $\alpha \colon \Sigma \uplus \Gamma_X \to \mathbb{N}$ is the *arity function*;
- ε is the *object existence symbol* with $\alpha(\varepsilon) = 1$;
- $\sim$ is the *object equality symbol* with $\alpha(\sim) = 2$;
- $\Sigma_C = \{\mathsf{C}_1, \ldots, \mathsf{C}_c\}$ is the set of *class symbols* with $\alpha(\mathsf{C}_i) = 1$ for each $\mathsf{C}_i \in \Sigma_C$;
- $\Sigma_R = \{\mathsf{R}_1, \ldots, \mathsf{R}_r\}$ is the set of *reference symbols* with $\alpha(\mathsf{R}_i) = 2$ for each $\mathsf{R}_i \in \Sigma_R$;
- $\Sigma_F = \{\mathsf{F}_1, \ldots, \mathsf{F}_f\}$ is the set of *predicate symbols*;
- $\Gamma_C \subseteq \Sigma_C$, $\Gamma_R \subseteq \Sigma_R$, and $\Gamma_F \subseteq \Sigma_F$ are the sets of *numerically tracked* class, relation, and predicate *symbols*, respectively; and
- $\Gamma_X = \{\mathsf{X}_1, \ldots, \mathsf{X}_x\}$ is the set of *auxiliary variable symbols*.

Scoped partial models provide a system of linear inequalities $\mathcal{S}_P$ alongside the 3-valued logic interpretation $\mathcal{I}_P$. Compared to Definition 2.3, we omit the numeric value interpretation $\mathcal{V}_P$, since we only consider domain objects as part of the model.

Definition 2.20 (Scoped partial model) A *scoped partial model* over a signature $\langle \Sigma, \Gamma, \alpha \rangle$ is a tuple $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \rangle$, where

- O_P is a finite set of *individuals* (i.e., objects);
- $\mathcal{I}_P$ gives a *3-valued interpretation* $\mathcal{I}_P(\varsigma) \colon O_P^{\alpha(S)} \to \{1, 0, ½\}$ for each symbol $\varsigma \in \Sigma$; and

- $\mathcal{S}_P$ is a *system of linear inequalities* over the linear inequality variables $\mathcal{X}_P = \{\hat{\gamma}(o_1, \ldots, o_{\alpha(\gamma)}) \mid \gamma \in \Gamma, o_1, \ldots, o_{\alpha(\gamma)} \in \mathcal{O}_P\}$.

For any symbol $\varsigma \in \Sigma \cap \Gamma$, we can consider $\hat{\varsigma}(o_1, \ldots, o_{\alpha(\sigma)})$ as the linear inequality variable corresponding to the number of ς relationships represented by the individuals $\langle o_1, \ldots, o_{\alpha(\sigma)}\rangle$ in the partial model as follows:

- For each class symbol $\mathsf{C} \in \Sigma$ and individual $o \in \mathcal{O}_P$, the variable $\hat{\mathsf{C}}(o)$ corresponds to the number of objects of type C represented by o.
- Likewise, for each reference symbol $\mathsf{R} \in \Sigma$ and objects $o_1, o_2 \in \mathcal{O}_P$, the variable $\hat{\mathsf{R}}(o_1, o_2)$ corresponds to the number of links of type R from objects represented by o_1 to objects represented by o_2.
- For k-ary predicate symbol $\mathsf{F} \in \Sigma$ (i.e., $\alpha(\mathsf{F}) = k$) and objects $o_1, \ldots, o_k \in \mathcal{O}_P$, the variable $\hat{\mathsf{F}}(o_1, \ldots, o_k)$ corresponds to the number of matches of the predicate F where the ith argument of the match is represented by the individual o_i for each $i = 1, \ldots, k$.
- Lastly, we interpret the variables $\hat{\varepsilon}(o)$ associated with the existence symbol ε as the number of objects represented by the individual o.
- Because the 3-valued interpretation $\mathcal{I}_P(\sim)$ of the equality symbol $\sim$ governs the merging and splitting of partial model individuals, it already involved in determining the number of objects represented in a partial model. Thus, we refrain from providing a linear inequality variable interpretation $\hat{\sim}$ for it by ensuring that $\sim \notin \Gamma$.

Auxiliary symbols $\mathsf{X} \in \Gamma_{\mathsf{X}}$ give rise to linear equation variables with no corresponding logical relations in the model. Nevertheless, they are still associated with a tuple $\langle o_1, \ldots, o_{\alpha(\gamma)}\rangle$ of objects which help in identifying the variable during model refinement. Auxiliary variables with $\alpha(\gamma) = 0$ are global variables, which remain unaffected by refinement.

For 0-ary logical symbols $\varsigma \in \Sigma$ (i.e., $\alpha(\varsigma) = 0$), we will write $\mathcal{I}_P(\varsigma)$ instead of $\mathcal{I}_P(\varsigma)()$. Likewise, for $\gamma \in \Gamma$ with $\alpha(\gamma) = 0$, we will write $\hat{\gamma}$ instead of $\hat{\gamma}()$. Thus, 0-ary auxiliary variable symbols give rise to named *global linear equation variables* in $\mathcal{S}_P$. We will use the notation $\hat{o} = \hat{\varepsilon}(o)$ to refer to the linear inequality variable corresponding to the *number of concrete model objects* represented by the individual $o \in \mathcal{O}_P$.

In the context of model generation, we restrict the possible combination of those predicates to exclude inconsistent and irrelevant constructs that are not productive as intermediate states of model generation.

Definition 2.21 A scoped partial model $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P\rangle$ is *structurally regular* if

- there are no *nonexistent objects*, i.e., if $o \in \mathcal{O}_P$, then $\mathcal{I}_P(\varepsilon)(o) \neq 0$; and
- there are no *object merges*, i.e., if $o_1, o_2 \in \mathcal{O}_P$ and $\mathcal{I}_P(\sim)(o_1, o_2) \neq 0$, then $o_1 = o_2$.

The *numerical regularity criteria* of scoped partial models ensures consistency of the 3-valued interpretation $\mathcal{I}_P$ and the scopes $\mathcal{S}_P$.

Definition 2.22 A scoped partial model $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P\rangle$ is *numerically regular* if

- $\mathcal{S}_P$ is satisfiable;
- surely false relationships represent exactly 0 instances of the relationship, i.e., for each $\varsigma \in \Sigma \cap \Gamma$ and $o_1, \ldots, o_{\alpha(\varsigma)} \in \mathcal{O}_P$,

$$\mathcal{I}_P(\varsigma)(o_1, \ldots, o_{\alpha(\varsigma)}) = 0 \quad \Longrightarrow \quad \mathcal{S}_P \vDash \hat{\varsigma}(o_1, \ldots, o_{\alpha(S)}) = 0.$$

- surely existing individuals must represent at least one object, i.e., for each $o \in \mathcal{O}_P$,

$$\mathcal{I}_P(\varepsilon)(o) = 1 \quad \Longrightarrow \quad \mathcal{S}_P \vDash \hat{o}$$

Table 2.1: Explanation for existence and self-equivalence predicates

$\varepsilon(x)$	$x \sim x$	Description	Symbol	Regularity criteria
1	1	concrete object	[1..1]	$\mathcal{S}_P \vDash \widehat{x} = 1$
½	1	uncertain, concrete	[0..1]	$\mathcal{S}_P \vDash \widehat{x} \leq 1$
1	½	multi-object	[1..*]	$\mathcal{S}_P \vDash \widehat{x} \geq 1$
½	½	uncertain, multi	[0..*]	unrestricted

- individuals that are surely equal to themselves cannot represent multiple objects, because they cannot be split further, i.e., for each $o \in O_P$,

$$\mathcal{I}_P(\sim)(o, o) = 1 \quad \Longrightarrow \quad \mathcal{S}_P \vDash \hat{o} \leq 1.$$

Table 2.1 summarizes the possible cases of uncertain existence and self-equivalence permitted by Definition 2.22. Individuals that may represent multiple concrete model objects are *multi-objects*, while individuals with uncertain existence are *uncertain* elements of the partial model.

In contrast with 4-valued partial models, for the most part, we will restrict our attention to scoped partial models that are both structurally and numerically regular.

Definition 2.23 A scoped partial model is *regular* if it is both structurally regular and numerically regular.

2.3.3 Refinement and concretization

Analogously to 4-valued partial model refinement in Definition 2.7, we introduce the refinement of scoped partial models to gradually incorporate information into the models.

Definition 2.24 The relation $ref \subseteq O_P \times O_Q$ is a *refinement* from the scoped partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ to the scoped partial model $Q = \langle O_Q, \mathcal{I}_Q, \mathcal{S}_Q \rangle$ over the same signature $\langle \Sigma, \Gamma, \alpha \rangle$, written as $P \succcurlyeq_{ref} Q$, if

SR1. no surely existing object disappears, i.e., if $p \in O_P$ and $\mathcal{I}_P(\varepsilon)(p) = 1$, then there is some successor $q \in O_Q$ such that $\langle p, q \rangle \in ref$ and $\mathcal{I}_P(\varepsilon)(p) \succcurlyeq \mathcal{I}_Q(\varepsilon)(q)$;

SR2. no possibly existing objects are created from scratch, i.e., if $q \in O_Q$ and $\mathcal{I}_Q(\varepsilon)(q) \neq 0$, then there is some predecessor $p \in O_P$ such that $\langle p, q \rangle \in ref$ and $\mathcal{I}_P(\varepsilon)(p) \neq 0$;

SR3. ref obeys the *information ordering of 3-valued interpretations*, i.e., if $\varsigma \in \Sigma \setminus \{\varepsilon\}$ and $\langle p_1, q_1 \rangle, \ldots, \langle p_{\alpha(\varsigma)}, q_{\alpha(\varsigma)} \rangle \in ref$, then $\mathcal{I}_P(\varsigma)(p_1, \ldots, p_{\alpha(\varsigma)}) \succcurlyeq \mathcal{I}_Q(\varsigma)(q_1, \ldots, q_{\alpha(\varsigma)})$; and

SR4. $\mathcal{S}_Q$ *refines the numerical information* in $\mathcal{S}_P$, i.e., $\mathcal{S}_Q \vDash \mathcal{S}_P[ref]$, where $\mathcal{S}_P[ref]$ is obtained from $\mathcal{S}_P$ by replacing the occurrences of all variables $\hat{\gamma}(p_1, \ldots, p_{\alpha(\gamma)})$ with the sums $\sum_{\langle p_1, q_1 \rangle, \ldots, \langle p_{\alpha(\gamma)}, q_{\alpha(\gamma)} \rangle \in ref} \hat{\gamma}(q_1, \ldots, q_{\alpha(\gamma)})$.

We will write $P \succcurlyeq Q$ if we have $P \succcurlyeq_{ref} Q$ for some refinement ref.

In a concrete scoped partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ (i.e., a scoped *instance model*), the system of linear inequalities $\mathcal{S}_P$ must reflect the number of relationship instances corresponding to tuples of objects surely belonging to classes, references, and predicate symbols in P. Therefore, $k(\hat{\varsigma})(o_1, \ldots, o_{\alpha(\varsigma)})$ is entirely determined by $\mathcal{I}_P(\varsigma)(o_1, \ldots, o_{\alpha(\varsigma)})$ for all $\varsigma \in \Sigma \cap \Gamma$, $o_1, \ldots, o_{\alpha(\varsigma)} \in O_P$, and bindings $k \colon \mathcal{X}_P \to \mathbb{N}$ that satisfy $k \vDash \mathcal{S}_P$. However, for an auxiliary variable symbol $\mathsf{X} \in \Gamma_{\mathsf{X}}$, satisfying bindings k might have (depending on the constraints in $\mathcal{S}_P$) many possible values of $\hat{\mathsf{X}}(o_1, \ldots, o_{\alpha(\mathsf{X})})$.

Definition 2.25 A regular scoped partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ is *concrete* if

- $\mathcal{I}_P$ contains 1 and 0 values only, i.e., for each $\varsigma \in \Sigma \cap \Gamma$ and $o_1, \ldots, o_{\alpha(\varsigma)} \in O_P$, we have $\mathcal{I}_P(\varsigma)(o_1, \ldots, o_{\alpha(\varsigma)}) \in \{1, 0\}$; and
- the linear inequality interpretation $\mathcal{S}_P$ coincides with the 3-valued interpretation $\mathcal{I}_P$, i.e., for each $\varsigma \in \Sigma$ and $o_1, \ldots, o_{\alpha(\varsigma)} \in O_P$,

$$\mathcal{I}_P(\varsigma)(o_1, \ldots, o_{\alpha(\varsigma)}) = 1 \implies \mathcal{S}_P \vDash \hat{\varsigma}(o_1, \ldots, o_{\alpha(\varsigma)}) = 1,$$
$$\mathcal{I}_P(\varsigma)(o_1, \ldots, o_{\alpha(\varsigma)}) = 0 \implies \mathcal{S}_P \vDash \hat{\varsigma}(o_1, \ldots, o_{\alpha(\varsigma)}) = 0.$$

Unlike Proposition 2.9, we can only prove transitivity of scoped partial model refinement form *regular* models. This does not become a limitation in practice, because model generation can proceed through a chain of regular model refinements $P_0 \succcurlyeq P_1 \succcurlyeq \cdots \succcurlyeq M$ (i.e., $P_0, P_1, \ldots, M$ are all regular) to obtain some concrete model M.

Proposition 2.26 Refinement of regular scoped partial models is transitive, i.e., if P, Q, R are regular partial models, $P \succcurlyeq_{ref_1} Q$, and $Q \succcurlyeq_{ref_2} R$, then $P \succcurlyeq_{ref_3} R$ for some refinement ref_3.

2.3.4 Scope constraints

Let the *count aggregation* $\#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\varsigma)})]^P_Z$ denote the number of $\gamma \in \Gamma$ relationships in the scoped partial model P that match the *mask* $\bar{v}_1, \ldots, \bar{v}_{\alpha(\varsigma)}$. Each $\bar{v}_i$ is either a variable v_j or a *wildcard* $*$. In the former case, we are counting γ tuples where the ith element is the individual $Z(v_i) \in O_P$ bound to the variable v_i. In the latter case, no restrictions are placed on the counted tuples.

The result of evaluating $\#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\varsigma)})]^P_Z$ is not a concrete number, but a linear combination $\sum_{\mathfrak{x}_i \in \mathcal{X}_P} c_i \cdot \mathfrak{x}_i$ of linear inequality variables. Therefore, such expressions cannot be used in computations directly, but only in linear inequalities.

Example 2.5 The expression $\#[\mathrm{target}(v_1, v_2)]^P_Z$ refers to the number of target references represented in the partial model P between the objects $Z(v_1)$ and $Z(v_2)$. It may be the case $\mathcal{S}_P \vDash \#[\mathrm{target}(v_1, v_2)]^P_Z \geq 2$ if at least one of $Z(v_1)$ and $Z(v_2)$ are multi-objects, which can be *split* to obtain multiple target links. In particular, the expression $\#[\varepsilon(v_1)]^P_Z$ refers to the number of objects represented by the multi-object $Z(v_1)$.
The expression $\#[\mathrm{target}(v_1, *)]^P_Z$ refers to the number of target links starting from $Z(v_1)$, while $\#[\mathrm{target}(*, *)]^P_Z$ refers to the number of target represented in P overall. Similarly, $\#[\mathrm{SmallSat}(*)]^P_Z$ refers to the number of SmallSat instances represented in the model.

As a special case, in any valuation $k \vDash \mathcal{S}_M$ associated with a concrete model M, Definition 2.25 ensures that $\#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\varsigma)})]^P_Z = \sum_{\mathfrak{x}_i \in \mathcal{X}_P} c_i \cdot \mathfrak{x}_i$ evaluated to a single quantity $N = \sum_{\mathfrak{x}_i \in \mathcal{X}_P} c_i \cdot k(\mathfrak{x}_i)$ that depends only on M but not k. The quantity N is precisely the number of γ links present in the model that match the mask $\bar{v}_1, \ldots, \bar{v}_{\alpha(\varsigma)}$, since concrete models may contain no multi-objects.

Definition 2.27 Let $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ be a scoped partial model over the signature Σ, Γ, α, let $V = \{v_1, \ldots, v_m\}$ be a finite set of *variables*, and let $Z : V \to O_P$ be a *variable binding*. The function $resolve^P_Z : (V \cup \{*\})^n \to 2^{O^n_P}$ is defined as

$$resolve^P_Z(\bar{v}_1, \ldots, \bar{v}_n) = \{\langle o_1, \ldots, o_n \rangle \in O^n_P \mid \bar{v}_i = * \text{ or } o_i = Z(\bar{v}_i) \text{ for all } i = 1, \ldots, n\},$$

If $\gamma \in \Gamma$, we define $\#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\varsigma)})]^P_Z = \sum_{\langle o_1, \ldots, o_{\alpha(\varsigma)} \rangle \in resolve^P_Z(\bar{v}_1, \ldots, \bar{v}_{\alpha(\varsigma)})} \hat{\gamma}(o_1, \ldots, o_{\alpha(\varsigma)})$ as the linear combination of linear inequality variables corresponding to the number of γ

relationships in P that match the *mask* $\langle \bar{v}_1, \dots, \bar{v}_{\alpha(\varsigma)} \rangle$ under the variable binding Z. We say that the mask $\langle \bar{v}_1, \dots, \bar{v}_n \rangle$ is Z-*focused* if $\bar{v}_i = *$ or $\mathcal{I}_P(\sim)(Z(\bar{v}_i), Z(\bar{v}_i)) = 1$ for all $i = 1, \dots, n$.

Linear inequalities formed by count aggregations are preserved under the refinement of regular scoped partial models. In particular, this allows us to add such constraints to the linear inequalities $\mathcal{S}_P$ associated with a partial model P while making sure that they will hold for any concrete model $M \preccurlyeq P$ obtained by refinement.

Masks which are focused, i.e., where none of the objects bound to the variables can be split during refinement, are of interest because they represent accurate lower bounds in addition to accurate upper bounds.

Lemma 2.28 If $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ and $Q = \langle \mathcal{O}_Q, \mathcal{I}_Q, \mathcal{S}_Q \rangle$ are regular scoped partial models over the signature $\langle \Sigma, \Gamma, \alpha \rangle$, $P \mathrel{\rotatebox{90}{$\Vdash$}}_{ref} Q$ for some refinement ref, $V = \{v_1, \dots, v_m\}$ is a set of variables, $Z \colon V \to \mathcal{O}_P$ and $Y \colon V \to \mathcal{O}_Q$ are variable bindings, $\langle Z(v), Y(v) \rangle \in ref$ for all $v \in V$, and $\gamma, \delta \in \Gamma$, then

a. if $\mathcal{S}_P \vDash \#[\gamma(\bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)})]_Z^P \leq U$, then $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)})]_Y^Q \leq U$.
b. if $\mathcal{S}_P \vDash \#[\gamma(\bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)})]_Z^P \geq L$, and $\langle \bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)} \rangle$ is Z-focused, then $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)})]_Y^Q \geq L$; and
c. if $\mathcal{S}_P \vDash \#[\gamma(\bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)})]_Z^P \leq \#[\delta(\bar{v}'_1, \dots, \bar{v}'_{\alpha(\delta)})]_Z^P$, and $\langle \bar{v}'_1, \dots, \bar{v}'_{\alpha(\delta)} \rangle$ is Z-focused, then $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)})]_Y^Q \leq \#[\delta(\bar{v}'_1, \dots, \bar{v}'_{\alpha(\delta)})]_Y^Q$.

2.3.5 Compatibility and inconsistency of scoped partial models

As for logic to define well-formedness constraints and graph predicate, we extend the basic logical language from Definition 2.2 with numerical expressions over count aggregations.

The resulting semantics describe the evaluation $[\![\varphi]\!]_Z^P$ for logic formulas φ according to 3-valued logic abstraction. Compared to Definition 2.10, we do not add a dedicated evaluation operator for numerical expressions. Instead, we introduce the operators $\mathsf{count}\,\gamma(\bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)}) \leq U$ and $\mathsf{count}\,\gamma(\bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)}) \geq L$ directly into the syntax of logic expressions.

Definition 2.29 The *semantics* $[\![\varphi]\!]_Z^P$ of a logic formula φ with free variables $V = \{v_1, \dots, v_k\}$ over the regular scoped partial model $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$, the signature $\langle \Sigma, \Gamma, \alpha \rangle$, and the variable binding $Z \colon V \to \mathcal{O}_P$ is inductively defined as

$$
\begin{aligned}
[\![\varsigma(v_1, \dots, v_{\alpha(\varsigma)})]\!]_Z^P &= \mathcal{I}_P(\varsigma)(Z(v_1), \dots, Z(v_{\alpha(\varsigma)})) \text{ for all } \varsigma \in \Sigma \setminus \{\varepsilon, \sim\},\\
[\![v_1 \sim v_2]\!]_Z^P &= \mathcal{I}_P(\sim)(Z(v_1), Z(v_2)),\\
[\![\neg\varphi_1]\!]_Z^P &= \neg^3 [\![\varphi_1]\!]_Z^P,\\
[\![\varphi_1 \vee \varphi_2]\!]_Z^P &= [\![\varphi_1]\!]_Z^P \vee^3 [\![\varphi_2]\!]_Z^P,\\
[\![\varphi_1 \wedge \varphi_2]\!]_Z^P &= [\![\varphi_1]\!]_Z^P \wedge^3 [\![\varphi_2]\!]_Z^P,\\
[\![\exists v \colon \varphi_1]\!]_Z^P &= \bigvee_{o \in \mathcal{O}_P}^3 \mathcal{I}_P(\varepsilon)(o) \wedge^3 [\![\varphi_1]\!]_{Z, v \mapsto o}^P,\\
[\![\forall v \colon \varphi_1]\!]_Z^P &= \neg^3 [\![\exists v \colon \neg\varphi_1]\!]_Z^P,
\end{aligned}
$$

$$[\![\varsigma^+(v_1,v_2)]\!]^P_Z = \Big[\!\Big[\varsigma(v_1,v_2) \vee \bigvee_{i=1}^{|O_P|} \exists u_1 \colon \cdots \exists u_i \colon \varsigma(v_1,u_1) \wedge \bigwedge_{j=1}^{i-1} \varsigma(u_j,u_{j+1}) \wedge \varsigma(u_i,v_2)\Big]\!\Big]^Z_P$$
$$\text{for all } \varsigma \in \Sigma \setminus \{\sim\} \text{ with } \alpha(\varsigma) = 2,$$

$$[\![\mathsf{count}\,\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)}) \leq U]\!]^P_Z = \begin{cases} 1 & \text{if } \mathcal{S}_P \vDash \#[\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})]^P_Z \leq U, \\ 0 & \text{if } \mathcal{S}_P \vDash \#[\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})]^P_Z \geq U+1, \\ & \quad \text{and } \langle \bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)}\rangle \text{ is } Z\text{-focused}, \\ ½ & \text{otherwise,} \end{cases}$$

$$[\![\mathsf{count}\,\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)}) \geq L]\!]^P_Z = \begin{cases} 1 & \text{if } \mathcal{S}_P \vDash \#[\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})]^P_Z \geq L \\ & \quad \text{and } \langle \bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)}\rangle \text{ is } Z\text{-focused}, \\ 0 & \text{if } \mathcal{S}_P \vDash \#[\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})]^P_Z \leq L-1, \\ ½ & \text{otherwise.} \end{cases}$$

We will use $\Phi_{\Sigma,\Gamma}$ to denote the set of all possible logic formulas over the signature $\langle \Sigma, \Gamma, \alpha\rangle$.

The evaluation of logic expressions w.r.t. the inclusion of new information into scoped partial models by refinement is only monotonic for regular scoped partial models. Therefore, we must take care during model generation to only process regular models when pruning the search space according to the 3-valued logic evaluation of logical expressions.

Lemma 2.30 The semantics of expressions obeys partial model refinement, i.e., if $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P\rangle$ and $Q = \langle O_Q, \mathcal{I}_Q, \mathcal{S}_Q\rangle$ are regular scoped partial models over the scoped signature $\langle \Sigma, \Gamma, \alpha\rangle$, $P \succcurlyeq_{ref} Q$ for some refinement ref, φ is a logic formula with free variables $V = \{v_1,\ldots,v_k\}$, $Z \colon V \to O_P$ and $Y \colon V \to O_Q$ are variable bindings, and $\langle Z(v), Y(v)\rangle \in ref$ for all $v \in V$, then $[\![\varphi]\!]^P_Z \succcurlyeq [\![\varphi]\!]^Q_Y$.

We may define theories and compatibility for scoped partial models analogously to Definitions 2.12 and 2.13 for 4-valued partial models.

Definition 2.31 A *theory* for the scoped signature $\langle \Sigma, \Gamma, \alpha\rangle$ is a pair $\mathcal{T} = \langle d, \mathcal{E}\rangle$, where

- $d \colon \Sigma_{\mathsf{F}} \to \Phi_{\Sigma,\Gamma}$ is the *predicate definition function*, which assigns a formula $f(\mathsf{F}_j)$ with free variables $\{v_1,\ldots,v_{\alpha(\mathsf{F}_j)}\}$ to each predicate symbol $\mathsf{F}_j \in \Sigma_{\mathsf{F}}$; and
- $\mathcal{E} \subseteq \Phi_{\Sigma,\Gamma}$ is the set of *error formulas*.

When checking for compatibility, we may omit the case for the ↯ logic values, since scoped partial models cannot be paraconsistent.

Definition 2.32 A 4-valued partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{V}_P\rangle$ over the scoped signature $\langle \Sigma, \Gamma, \alpha\rangle$ is *compatible* with the theory $\mathcal{T} = \langle d, \mathcal{E}\rangle$, if

- for all $\mathsf{F}_j \in \Sigma_{\mathsf{F}}$ and $o_1,\ldots,o_{\alpha(\mathsf{F}_j)} \in O_P$, we have
$$[\![d(\mathsf{F}_j)]\!]^P_{v_1\mapsto o_1,\ldots,v_{\alpha(\mathsf{F}_j)}} \succcurlyeq \mathcal{I}_P(\mathsf{F}_j)(o_1,\ldots,o_{\alpha(\mathsf{F}_j)}); \text{ and}$$
- for all $\varphi \in \mathcal{E}$ with free variables $V = \{v_1,\ldots,v_m\}$ and for all valuations $Z \colon V \to O_P$, we have $[\![\varphi]\!]^P_Z \succcurlyeq 0$.

If some of these constraints are violated, we say that P is *inconsistent* with the theory $\mathcal{T}$.

2.3.6 Model generation and optimization problems

We may define scoped model generation problems analogously to Definition 2.14 in the case of 4-valued partial models.

Definition 2.33 Given a scoped partial model $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ over the scoped signature $\langle \Sigma, \Gamma, \alpha \rangle$ and a theory $\mathcal{T} = \langle d, \mathcal{E} \rangle$ over the same signature, the solutions of the corresponding *model generation problem*,

$$solutions(P, \mathcal{T}) = \{M \mid P \succcurlyeq M, M \text{ is compatible with } \mathcal{T}\},$$

are the concrete refinements of P that are compatible with $\mathcal{T}$.

In the model generation problem, the well-formedness constraints of the domain are encoded in the theory $\mathcal{T}$, while the system of linear equations $\mathcal{S}_P$ associated with the *initial partial model* P may encode any desired type scope constraints.

The additional numerical information present in the system of linear equations $\mathcal{S}_P$ associated with scoped partial models allows us to consider *model optimization* problems, where a given numerically tracked symbol $\mathsf{X}^* \in \Gamma_\mathsf{X}$ serves are the *objective function*. In the following, we define maximization problems, but minimization problems can be defined analogously.

Definition 2.34 The *cost* of the scoped partial model $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ over the scoped signature $\langle \Sigma, \Gamma, \alpha \rangle$ with respect to the numerically tracked symbol $\mathsf{X}^* \in \Gamma_\mathsf{X}$, where $\alpha(\mathsf{X}^*) = 0$, is the solution of the ILP $cost_{\mathsf{X}^*}(P) = \max_{k \vDash \mathcal{S}_P} k(\hat{\mathsf{X}}^*)$.

Since $\Sigma \not\ni \mathsf{X}^* \in \Gamma_\mathsf{X}$, $k(\mathsf{X}^*)$ can have many distinct values for different $k \vDash \mathcal{S}_P$ even if P is consistent. Due to Condition SR4, we have $cost_{\mathsf{X}^*}(P) \geq cost_{\mathsf{X}^*}(Q)$ for regular models P and Q whenever $P \succcurlyeq Q$. The inequality can be strict, i.e., $cost_{\mathsf{X}^*}(P) > cost_{\mathsf{X}^*}(Q)$ if the refinement discards possible solution $\mathcal{S}_P$ from $\mathcal{S}_Q$.

Now we are ready to define the model optimization problem.

Definition 2.35 The *model optimization problem* with the initial scoped partial model $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ over the scoped signature $\langle \Sigma, \Gamma, \alpha \rangle$, the theory $\mathcal{T}$, and the *cost symbol* $\mathsf{X}^* \in \Gamma_\mathsf{X}$, where $\alpha(\mathsf{X}^*) = 0$, which is written as

$$\max \mathsf{X}^* \textit{ subject to } \langle P, \mathcal{T} \rangle,$$

has the set of *optimal solutions*

$$optimal(P, \mathcal{T}, \mathsf{X}^*) = \{M \in solutions(P, \mathcal{T}) \mid cost_{\mathsf{X}^*}(M) \geq cost_{\mathsf{X}^*}(M') \text{ for all } M' \in solutions(P, \mathcal{T})\}.$$

2.4 Related work

The *Meta-Object Facility* (MOF) [MOF] is a conceptual core for model-driven engineering standardized by the Object Management Group. It is widely used in the specification general-purpose modeling languages, such as the *Unified Modeling Langauge* (UML) [UML]. The Eclipse Modeling Framework (EMF) [Ste+09] relies on *Ecore*, an implementation of the *Essential MOF* (EMOF) fragment of MOF as a technical foundation for metamodeling and model interchange.

The majority of industrial modeling toolchains rely on either MOF, UML, or EMF to capture models, as well as the associated *Object Constraint Language* (OCL) [OCL] for model querying and expressing well-formedness constraints. EMF and OCL supports a wide variety of user

defined and built-in data types [OCL, Section A.2] for numerical attributes. OCL is at least as expressive as first-order logic; it has been translated to first-order logic [BKS02; Soe+10; Sem+17] and CSP [CCR07; Gon+12].

However, MOF-based approaches lack native support of unknown aspects of models (*paracompleteness*) and inconsistency-tolerance (*paraconsistency*). Even though all types in OCL have a special bottom ($\bot$) value, and all types except OclInvalid have a special null (ε) value, these are unsuitable for paracomplete or paraconsistent modeling. The behavior of null corresponds to the *absence* of any value instead of the *potential* for some (yet to be specified value) [OCL, Section A.2.1.3]. Likewise, bottom is always propagated on error and thus cannot be used indicate the root cause of an error.

To alleviate this issue, standardized models may be extended with *annotations*, such as MAVO (*May*/*Abstract*/*Variable*/*Open world*) [Sal+15] for incompleteness or *merge conflict* annotations [SE06; CNS12] for inconsistency. Reasoning with such models has to take annotations into account, which can increase computational overhead considerably [SV17; WA21].

Alternatively, *flexible modeling* frameworks were developed [HS17], e.g. Muddle [Kol+13], FlexiMeta [Hil16], and JSMF [SB16], which may support different *phases of flexibility* (i.e., less information may be present in the model during early stages of development) and *level of conformance* (i.e., some constraints may be omitted for some instance models). However, they do not offer reasoning capabilities over unknown aspects or inconsistencies.

Fluid [VP03; Mez+19] and *multi-level metamodeling* [AK01] aim to support the gradual construction of models and solve challanges of expressiveness [DGC14] by relaxing the strict hierarchy between metamodels and models. *Potency-based* [Küh18] systems, e.g., DeepJava [KS07], MetaDepth [LG10], and Melanee [AG16], allow the definition of an arbitrary number of metalevels, while *level-blind* systems, e.g., DeepTelos [JN16] and VMTS [UTM18], do not impose levels as restriction on instantiation relations between *clabjects* (i.e., concepts with both a *classifier* and an *object facet*).

Inheritance between classifiers and instantiation between classifiers and objects are *refinement relationships* [TM15; Mez+19] between clabjects of a multi-level model. As a key difference from partial modelling, where refinement occurs *between (partial) models*, such refinement relationship occur *within a (multi-level) model* between modeled concepts. Thus, partial model based analysis remains applicable to changes (i.e., chains of refinements) to a model during analysis, while multi-level refinement can only be used to analyze concepts of a single model.

Feature models [Sch+07] as especially prevalent in software product line engineering and the specification of potential design decisions in variant-rich and reconfigurable systems. Automated analysis is available for the instantiation and consistency analysis of feature model [BSC10]. The analysis of real-world feature models is often easy, because they only contain few difficult logical constraints [MWC09; Lia+15]. Clafer [Bak+13] unifies feature models and class diagrams for additional expressive power. However, sound and complete analysis is only available for a subset of the language [Wec+18]. Similarly to scoped partial models, the analysis in [Wec+18] also relies on linear programming to bound the size of potential instance models.

Ontological approaches, e.g., the *Ontological Modeling Language* [Wag+22] from NASA JPL, as well as *KerML* and *SysML v2* [SysML2] from OMG, are gaining popularity in systems engineering. Partial information about the system design can be incorporated into an ontological *knowledge base* thanks to the *open world* assumption. The specification of well-formedness constraints using *description Logics* (see, e.g., [KSH12] for an introduction) allows the use of efficient decision procedures for reasoning about uncertainty and consistency of models, albeit at a reduced expressive power compared to full First-Order Logic.

Object Role Modeling (see, e.g., [Hal18]) approach to ontology modeling [Spy05] provides a facility for inconsistency-tolerance: while *alethic* (necessary) constraints must be satisfied at all times, *deontic* (obligatory) constraints may be violated by the knowledge base.

2.5 Conclusions

In this chapter, we proposed two extensions to the *partial modeling* [RSW04; Ren06; Var+18] formalism to support the extra-functional analysis of complex system architecture models.

The first proposed extension, *4-valued partial models* [c7; j2] incorporate 4-valued Belnap–Dunn logic [Bel77; KO17] to describe *paraconsistent* (inconsistency-tolerant) models in addition to the *paracomplete* (uncertain) aspects supported by existing partial modeling techniques. We also integrated *interval abstraction* [Min04; Kul09; JM09] to support abstraction over quantitative *attribute values* of model elements. Chapter 3 provides reasoning capabilities for 4-valued partial models, which are subsequently exploited in Chapter 5 to construct Phased-Mission stochastic analysis models for reconfigurable systems.

The second proposed extension, *scoped partial models* [j1] incorporate *polyhedron abstraction* [CH78; BHZ08] to support *model size* and *multiplicity* constraints in partial models. Chapter 4 provides reasoning capabilities for scoped partial models, which significantly improve the scalability of graph model generation.

We defined *model generation* [j1] problems for both extended formalisms and *model optimization* [j5] for scoped partial models. Chapters 4 and 6 rely on such problems to enable the synthesis of interferometry mission architecture models and test inputs for industrial modeling tools, as well as *witness models* [j5] for WCET analysis, respectively.

Chapter 3

Fully compositional view transformations

Complex industrial toolchains used for designing cyber-physical systems frequently depend on various models on different levels of abstraction. Abstract models [Bru+17] can be derived and synchronized by view model transformations upon changes of one or more underlying source models.

View synchronization challenges are addressed by using either *general purpose model transformation tools* (e.g. ATL [Jou+08; MTD17], ETL [KPP], Henshin [Are+10], Viatra [Var+16]), *bidirectional model synchronization* (e.g. various TGG tools [Sch95; Lau+12; GHL14; GK07] and QVTr [QVT]), or dedicated *view transformation* techniques (e.g. View TGGs [JKS06; Anj+14], Active Operations [Bea+10], Viatra Views [Deb+14], QuEST [GDM14]).

To tackle complex scenarios, view model transformations are desirably *defined in a compositional way* to reuse existing transformations without further changes. While sequential composition (chaining) is widely supported, existing tools need to impose major restrictions in case of parallel composition (merging) of target views. An ideal (forward only) *view transformation engine* is *reactive* (i.e. reacts to source model changes), target *incremental* (i.e. updates only affected target elements), *consistent* (i.e. continuously maintains a transformation relation between source and target models) and *validating* (i.e. the target model is a valid, materializable instance of the target language).

Currently, there is a significant trade-off in existing tools between the expressiveness and compositionality of the view transformation language, and the level of support for desirable features of the view transformation engine. On the one hand, fully reactive behavior is a challenge in itself supported by only few tools (e.g. [Var+16; Bea+10; MTD17]), while incrementality, consistency and validity is provided at the same time for very restrictive transformation languages. Practical model transformation engines frequently fail to restore consistency between models [Ste14].

Our main contribution in this chapter is a unidirectional view transformation approach with a (1) a *fully compositional view transformation language*, and (2) a *reactive, incremental, validating and inconsistency-tolerant transformation engine*. The view transformation language explicitly reuses the Viatra Query Language [Ujh+15] to declaratively capture relevant source and view patterns by following the principles of ramification [Küh+10]. Moreover, *inconsistency-tolerant partial models* (a generalization of partial models of [FSC12; Var+18]) provide the conceptual core of the transformation engine.

The transformation engine reacts to aggregated changes of the source model observed in the result set of graph queries (hence *reactive*), then it builds and maintains a partial model as a knowledge base with traceability links. Once the partial view model becomes a valid instance of

the target metamodel (i.e. relevant aggregated changes are observed in the knowledge base, and structural constraints are respected), the target view model is *incrementally updated by providing a corresponding change* (e.g. model delta, notification or API call). Our engine is *inconsistency tolerant* in the sense that inconsistencies are semantically persisted in the internal knowledge base. This allows to keep a large fragment of the source and view models in sync in case of inconsistent source changes and provides hippocratic behavior (i.e. avoids the unnecessary deletion and recreation of elements).

The transformation engine is implemented as a prototype tool[1] and integrated into the open source Viatra transformation framework [Ber+15]. Moreover, we carry out an initial scalability evaluation by adapting an existing view model transformation from an industrial research project (aiming to carry out dependability evaluation of automotive designs) to the open Train Benchmark [Szá+17]. Artifacts related to this chapter are also available in [d21].

The contents of this chapter are based on the conference paper [c7].

3.1 An overview of compositional view transformations

A view transformation $Trg = tr(Src_1, \ldots, Src_k)$ aims to derive a target view model Trg as an abstraction of a set of source models $Src_1, \ldots, Src_k$. A tr is a mapping from source(s) to target models typically with loss of information.

Moreover, in a typical view synchronization scenario, each target change is causally dependent on some (aggregate) change of the source model (e.g. a model delta or notification upon model update). This causal dependence can be captured by a *match* of a *view transformation rule* in the source model which triggers the simultaneous creation of respective target elements together with some traceability links between source and target elements.

A motivating example The running example of the paper is adapted from an industrial project where formal dependability analysis of automotive models were carried out by composing two view transformations: (1) $PN = tr_1(Aut)$ maps automotive component models Aut to stochastic Petri nets PN [Ajm+94], and (2) $PN = tr_2(Dep)$ is a reusable mapping [BMM99; MPB02] from a domain-independent dependability model Dep to stochastic Petri nets. The target Petri net model is defined as the (parallel) composition of the two transformations $PN = tr_1 \parallel tr_2(Aut, Dep)$ calculated over the two input models.

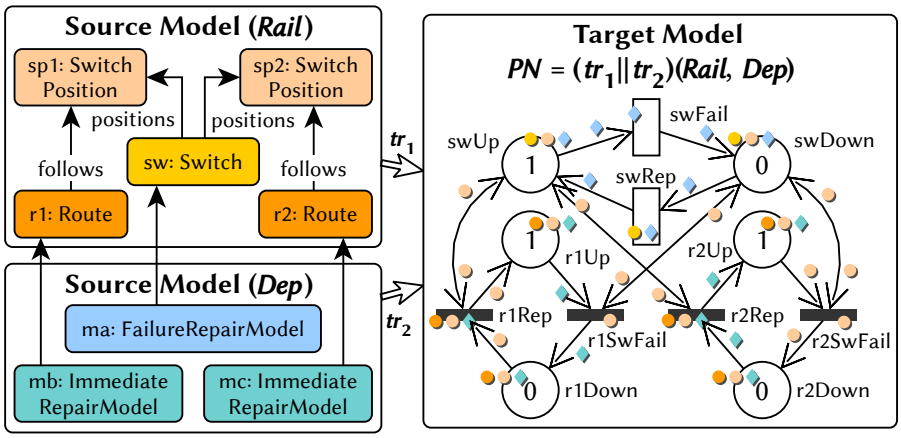
Due to IP restrictions of automotive models, we present the challenge using a public model of railway networks developed as part of the Train Benchmark [Szá+17], a cross-technology macrobenchmark of graph-based model query tools. A sample source and target model are shown along with the traceability links in Fig. 3.1, while some transformation rules will be illustrated later in Fig. 3.6.

3.1.1 Levels of compositional definitions

To categorize the levels of compositionality in view transformations, let us assume the existence of two view transformations, tr_1 and tr_2 and a single source model Src to simplify the discussion. Transformations tr_1 and tr_2 can be composed in different ways.

In many practical scenarios [Anj+14; Heg+16], chaining of view transformations is necessitated, which is a *sequential composition* of transformations $Trg = tr_2 \circ tr_1(Src) = tr_2(tr_1(Src))$ where tr_2 takes the output of tr_1 as its source model, and the target model of this transformation chain is the subsequent result of tr_2. The definition of sequential composition is supported in several tools [Mel+05; Heg+16].

[1] Available under the Eclipse License 1.0 at `https://github.com/ftsrg/viewmodel`



Trace (tr_1 : *Rail* → *PN*): sw ↦ {swUp, swDown, swRep, swFail}, r1 ↦ {r1Up, r1Down, r1Rep}, r2 ↦ {r2Up, r2Down, r2Rep}, sp1 ↦ {swUp, swDown, r1Up, r1Down, r1Rep, r1SwFail}, sp2 ↦ {swUp, swDown, r2Up, r2Down, r2Rep, r2SwFail}
Trace (tr_2 : *Dep* → *PN*): ma ↦ {swUp, swDown, swRep, swFail}, mb ↦ {r1Up, r1Down, r1Rep}, mc ↦ {r2Up, r2Down, r2Rep}

Markers denote the trace of the nodes; multiple markers mean objects merged from both view transformations.

Figure 3.1: Source and target models with traceability links.

Given two existing view transformations $Trg_1 = tr_1(Src)$ and $Trg_2 = tr_2(Src)$, another relevant aspect is *parallel composition* $Trg = tr_1 \parallel tr_2(Src) = tr_1(Src) \oplus tr_2(Src)$ where the target model is derived by merging (or gluing) the results of transformations tr_1 and tr_2 both applied on the same source model *Src*. If the two transformations are independent, the target model is the union of the individual transformations, otherwise the aggregated result can be computed e.g. by category-theoretical foundations [CNS12; DXC11; Ehr+99]. Below, we briefly categorize the *major assumptions for parallel composition* $tr_1 \parallel tr_2$ used in existing transformation tools.

1. In the *independent* case, each target object is fully defined by a single rule in one transformation, thus a union of target elements can be taken without merge, i.e. $Trg = tr_1(Src) \cup tr_2(Src)$. Otherwise, a new transformation tr_3 needs to be written manually.

2. In the *serializable* case, the parallel composition is turned into a sequential composition where one transformation (e.g. tr_1) is taken as-is (called *primary*) while the other transformation tr_2 (called *secondary*) needs to be manually changed to tr'_2, i.e. $tr_1 \parallel tr_2(Src) = tr'_2(tr_1(Src), Src))$ or $tr'_1(tr_2(Src), Src))$.

   a) Certain transformation languages (e.g. ATL [Jou+08]) *restrict* primary rules, i.e. at most one serialization $tr'_2(tr_1(Src), Src))$ or $tr'_1(tr_2(Src), Src))$ can exist. In ATL, outgoing references of an object can only be defined in a primary rule (to ensure multiplicity constraints in the target language), thus a static check will prevent serializing the transformations in the wrong way.

   b) Other serializable view transformation approaches [Ber+15; Heg+16] are *unrestricted* to allow both serializations $tr'_2(tr_1(Src), Src))$ and $tr'_1(tr_2(Src), Src))$, but one of the transformations still needs to be adapted to take the output of the other.

3. *Fully compositional* view transformation approaches allow to compose tr_1 and tr_2 as $tr_1 \parallel tr_2(Src) = tr_1(Src) \cup_? tr_2(Src)$ without changing the transformations by using some model merge operator $\cup_?$ to weave the target models of individual transformations into a joint result.

a) In *ID-based* $tr_1 \parallel tr_2(Src) = tr_1(Src) \cup_{\text{ID}} tr_2(Src)$ composition, rules assign the same ID to objects that need to be merged in the final target model. The ID attribute can be selected from the metamodel intrusively [QVT] or added by *augmentation* [Küh+10].

b) *Relation-based* $tr_1 \parallel_g tr_2(Src) = tr_1(Src) \cup_{g(Src)} tr_2(Src)$ composition can mark unrelated objects constructed separately by transformations tr_1 or tr_2 to be merged. The merge operation is a parameter, i.e. it can be specified as a categorial colimit with a suitable *reference* or *connection* model [Eng+97; SE06; DXC11], by *direct mappings* [CNS12], or by *graph bisimulation* [BFS00].

3.1.2 Properties of view transformation engines

A view transformation engine $Out^{(i)} = exec(tr, In^{(i)})$ repeatedly executes a transformation tr at a given logical time point i on an input $In^{(i)}$ (which can be the source model $Src^{(i)}$ or a delta $\Delta^{(i)}_{Src}$) to derive an output $Out^{(i)}$ (the target model $Trg^{(i)}$ or a delta $\Delta^{(i)}_{Trg}$) while (a) maintaining the consistency relation $Trg = tr(Src)$ between the source and target models and (b) keeping the target model a valid instance of the target language ($Trg \vDash MM_T$).

1. A *batch* engine takes the entire source model at any step: $Out^{(i)} = exec(tr, Src^{(i)})$. A *delta-based* engine takes a model change as input, but it executes on-demand: $Out^{(i)} = exec(tr, Src^{(i-1)}, \Delta^{(i)}_{Src})$. A *reactive* engine executes in response to source model changes by receiving deltas as model notifications: $Out^{(i)} = exec(tr, \Delta^{(i)}_{Src})$ [Ber+12a; MTD17]. Delta-based and reactive engines load the source model as a large delta upon initialization.

2. An *incremental* engine updates only those target elements which are affected by a specific source model change, that is $\Delta^{(i)}_{Trg} = Out^{(i)} = exec(tr, In^{(i)})$, thus the new target model is obtained by applying this delta: $Trg^{(i)} = Trg^{(i-1)} + \Delta^{(i)}_{Trg}$. A *non-incremental* engine derives the new target model from scratch:
$Trg^{(i)} = Out^{(i)} = exec(tr, In^{(i-1)})$.

3. A *consistent* engine continuously enforces consistency (correctness) constraints between source and target elements: if $Out^{(i)} = exec(tr, In^{(i)})$ then $Trg^{(i)} = tr(Src^{(i)})$. A *non-consistent* engine does not guarantee these constraints if the transformation rules are conflicting with each other (e.g. in case of a specific source change).

4. A *validating* engine derives the view model as a valid instance model of the target metamodel (or viewtype) where all metamodel constraints (e.g. aggregation, multiplicity) are satisfied: if $Trg^{(i)} = exec(tr, In^{(i)})$ then $Trg^{(i)} \vDash MM_T$. Checking these structural constraints of the target metamodel is out of scope for a *non-validating* engine, thus $Trg^{(i)} \nvDash MM_T$. A validating engine can be used for both materialized and virtualized viewtypes [Bru+17].

Fully compositional view transformations need to face the conceptual challenge that while enforcing the consistency between the source and target models, one may easily violate the structural constraints imposed by a metamodeling framework like EMF [Ste+09].

But for state-of-the-art transformation tools, these properties can only be achieved together by imposing major restrictions on the expressiveness of the transformation language, otherwise at least one of these properties will not be provided .

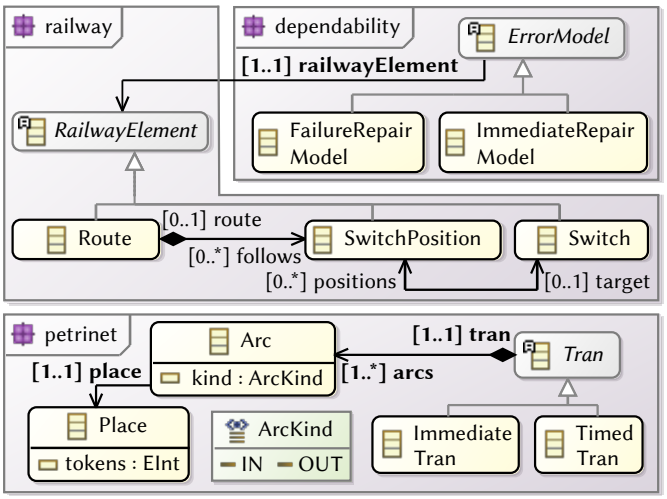


Figure 3.2: Two source and one target metamodels

3.2 Modelling and partial models

Our view transformation technique builds on 4-valued, i.e., inconsistency-tolerant partial models (see Section 2.2) which store inconsistent and unknown information in models by generalizing the merging of inconsistent and incomplete views in conceptual models [SE06].

For simplicity, we will omit attributes and predicate symbols from the discussion. Therefore, we will assume that

- the signature $\langle \Sigma, \alpha \rangle$ that is associated with the *metamodels* involved in the view transformation contains no attribute or predicate symbols, i.e., $\Sigma_A = \Sigma_F = \emptyset$; and
- every object is a domain object, i.e., $\mathcal{I}_P(\text{int})(p) = 0$ and $\mathcal{V}_P(p) = \emptyset$ for all objects $p \in O_P$ in any partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{V}_P \rangle$.

Nevertheless, the theory and the implementation can be easily extended to support attributes. In fact, the view transformation tool accompanying this chapter is able to construct view models that contain attributes.

Since each object in the partial model always has the same $\mathcal{V}_P$ value, from now on, we will omit $\mathcal{V}_P$ from the notation of partial models. Instead, we will simply write $P = \langle O_P, \mathcal{I}_P \rangle$.

3.2.1 Metamodels and structural constraints

In our running example, Fig. 3.2 defines two source (railway and dependability) and one target metamodels (petrinet).

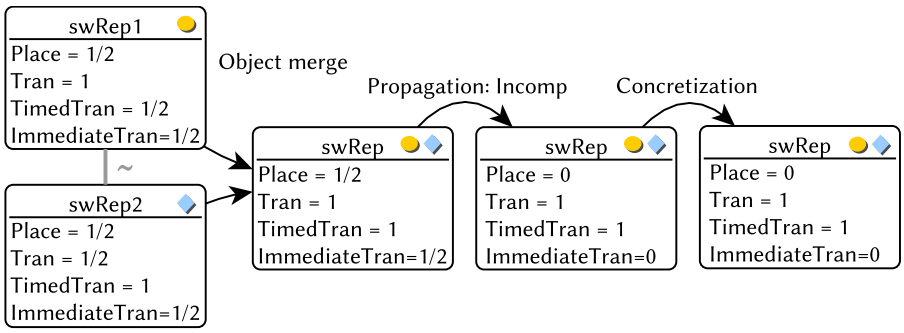
Metamodeling tools impose *structural constraints* on instance models to enforce a basic structure. In the Eclipse Modeling Framework (EMF) [Ste+09], violating such a structural constraint would prevent the materialization (saving) of a model.

Recall from Section 2.1.2 that EMF enforces the following structural constraints:

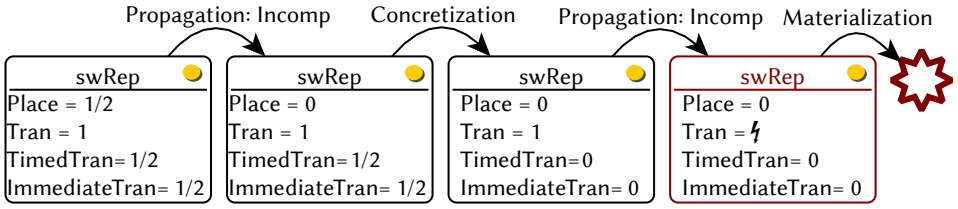
Type hierarchy. A metamodel defines a type system by *supertype* relations and *abstract* classes. For each object o, there shall be a single class C, where (i) C is non-abstract, and (ii) o is an instance of C' when C' is a supertype of C. In the petrinet metamodel in Fig. 3.2, an abstract Tran is either an ImmediateTran or a TimedTran.

Type compliance. The metamodel restricts the classes C_1, C_2 of objects at the ends of a reference R: $\forall o_1, o_2 \colon R(o_1, o_2) \Rightarrow C_1(o_1) \wedge C_2(o_2)$. For example, the target of a tran reference has to be an instance of the class Tran.

Multiplicity constraints are placed on upper bounds on the number of references adjacent to an object: $\forall o, o_1, o_2 \colon R(o, o_1) \wedge R(o, o_2) \Rightarrow o_1 \sim o_2$. For example, an Arc can have only one tran.



(a) Partial model refinement by merge functions



(b) Inconsistency found during partial model refinement

Figure 3.3: Sample chain of partial models

Inverse relations. Some references R and R′ always occur in pairs: $\forall o_1, o_2 \colon \mathrm{R}(o_1, o_2) \leftrightarrow \mathrm{R}'(o_2, o_1)$. See e.g., tran and arcs.

Containment hierarchy. EMF models are arranged in a strict tree hierarchy via the containment references. EMF restricts objects not to (i) have multiple containers, and (ii) form circles via containment references. E.g., an Arc cannot be contained by multiple Trans.

Equivalence relation $\sim$ is *reflexive*: $\forall o \colon o \sim o$, *symmetric*: $\forall o_1, o_2 \colon o_1 \sim o_2 \Rightarrow o_2 \sim o_1$, and *transitive*: $\forall o_1, o_2, o_3 \colon o_1 \sim o_2 \wedge o_2 \sim o_3 \Rightarrow o_1 \sim o_3$. In a regular instance model, objects are different from one other, but partial models may have explicit $\sim$ relations.

A partial model P is *concrete*, if (i) there are only 0 and 1 values in $\mathcal{I}_P$, and (ii) $o_1 \sim o_2$ iff o_1 and o_2 are the same element of O_P. A concrete partial model can be interpreted as an instance model M (i.e. a labeled graph). If all structural constraints are also respected ($M \vDash MM$) then M can be *materialized* into a regular EMF model.

> **Example 3.1** A sequence of partial models corresponding to Tran swRep of Fig. 3.1 is listed in Fig. 3.3b. For example, the left-most partial model states that element swRep is a Tran (1), and it is unknown (½) if it is also Place, a TimedTran or ImmediateTran.

3.2.2 Graph predicates

Recall from Definition 2.2 that a *graph predicate* $\varphi(v_1, \ldots, v_n)$ is a first-order logic (FOL) predicate over an infinite set of variables $(o_1, o_2, \ldots)$, the relation symbols of Σ (C_i, R_j, $\sim$), standard logic connectives ($\neg, \wedge, \vee$), and quantifiers ($\exists, \forall$). The *semantics of a graph predicate* $[\![\varphi(v_1, \ldots, v_n)]\!]^P_Z$ can be evaluated on a partial model P with variable binding $Z \colon \{v_1, \ldots, v_n\} \to O_P$ to yield a logic value 0, 1, ½ or ϟ (Definition 2.10). For concrete (2-valued) models this semantics is equivalent to standard FOL. A variable binding Z of $\varphi(v_1, \ldots, v_n)$ is called a *match*, if $1 \geqslant [\![\varphi(v_1, \ldots, v_n)]\!]^P_Z$, i.e., there is a real match or an inconsistency.

Following [Var+18], the structural constraints of a metamodel MM are captured by a *malformedness predicate* φ_{MM} *where a match of the predicate highlights elements that violate the constraint.* If P is an instance model M, and there is no match of predicate φ_{MM} ($1 \not\geqslant [\![\varphi_{MM}]\!]^P_Z$ for all variable bindings Z, i.e. it can be 0 or ½), then M is a valid instance model: $M \vDash MM$, thus it

Type Hierarchy:

SuperUp: $\dfrac{C_2(o)}{C_1(o)\uparrow}$, SuperDn: $\dfrac{\neg C_1(o)}{C_2(o)\downarrow}$ if C_1 is a supertype of C_2,

Join: $\dfrac{C_1(o) \wedge \cdots \wedge C_n(o) \wedge \neg C'_1(o) \wedge \cdots \wedge \neg C'_m(o)}{C^*(o)\uparrow}$
if among types that are not subtypes of any C'_j, C^* is the unique most generic non-abstract common subtype of all C_i ($n \geq 1$, $m \geq 0$, and C^* may be equal to one of $C_1, \ldots, C_n$),

Incomp: $\dfrac{C_1(o) \wedge \cdots \wedge C_n(o) \wedge \neg C'_1(o) \wedge \cdots \wedge \neg C'_m(o)}{C^*(o)\downarrow}$
if among types that are not subtypes of any C'_i, $C_1, \ldots, C_n$ and C^* have no common non-abstract subtype (not even an improper subtype, i.e. one of C_i or C^*),

Relations:

RelUp: $\dfrac{R(o_1, o_2)}{C_1(o_1)\uparrow\ C_2(o_2)\uparrow}$, RelDn: $\dfrac{\neg C_1(o_1) \vee \neg C_2(o_2)}{R(o_1, o_2)\downarrow}$ if C_1 and C_2 are the source and target of R,

Mult: $\dfrac{R(o, o_1) \wedge \neg(o_1 \sim o_2)}{R(o, o_2)\downarrow}$ if R has upper multiplicity 1,

ContMult: $\dfrac{R_1(o_1, o) \wedge \neg(o_1 \sim o_2)}{R_2(o_2, o)\downarrow}$ if R_1, R_2 are containment,

ContLoop: $\dfrac{R_1(o_1, o_2) \wedge \cdots \wedge R_{n-1}(o_{n-1}, o_n)}{R_n(o_n, o_1)\downarrow}$ if all R_i ($1 \leq i \leq n$) are containment

Equivalence:

$\sim$Symm: $\dfrac{o_1 \sim o_2}{o_2 \sim o_1\uparrow}$, $\sim$Tran: $\dfrac{o_1 \sim o_2 \wedge o_2 \sim o_3}{o_1 \sim o_3\uparrow}$, $\sim$Refl: $\dfrac{1}{o_1 \sim o_1\uparrow}$

Figure 3.4: Propagation rules for EMF structural constraints.

can be materialized. Since $\Sigma_F = \emptyset$, this corresponds to a theory (Definition 2.12) containing only error predicates and no other predicate definitions.

> **Example 3.2** A sample graph predicate derived from a structural constraint of the petrinet metamodel (see Fig. 3.2) captures that a Tran needs to be either a TimedTran or a ImmediateTran: $\forall o\colon \mathsf{Tran}(o) \Rightarrow \mathsf{TimedTran}(o) \vee \mathsf{ImmediateTran}(o)$.

3.2.3 Merge functions for partial models

In order to unify the semantic treatment of partial model concretization, view model merge and rule application, we define a *merge function* $m\colon O_P \to O_Q$ between objects of partial models P and Q. Function m is defined to ensure a *refinement relation* $\succcurlyeq\colon P \times Q$ between partial models P and Q [Var+18], which respects information ordering as stated by the following conditions for all $o_1, o_2 \in O_P$:

- $\mathcal{I}_P(C_i)(o_1) \succcurlyeq \mathcal{I}_Q(C_i)(m(o_1))$ for all $C_i \in \Sigma$,
- $\mathcal{I}_P(R_j)(o_1, o_2) \succcurlyeq \mathcal{I}_Q(R_j)(m(o_1), m(o_2))$ for all $R_j \in \Sigma$,
- $\mathcal{I}_P(\sim)(o_1, o_2) \succcurlyeq \mathcal{I}_Q(\sim)(m(o_1), m(o_2))$.

Partial model refinement is information preserving in the sense that all true (resp. false) predicates remain true (resp. false) in any refinement of a partial model (as proved in [Var+18]).

Example 3.3 Before the formal definitions, merge functions are informally illustrated along two different sequences in Fig. 3.3. The first sequence (Fig. 3.3a) starts from a partial model where two objects are marked as equivalent (∼), thus (a) an *object merge* function can be applied, which merges information from input objects: swRep becomes both a Tran (due to the top object) and an TimedTran (due to the bottom object). (b) Then an Incomp propagation rule will refine the model in accordance with the type hierarchy since a TimedTran object cannot be a Place or an ImmediateTran. Finally, (c) the concretization step has no further effect, and we obtain an instance model on the right.
The second sequence (Fig. 3.3b) first (a) applies an Incomp propagation rule to ensure that a Tran is no longer a Place. Then (b) concretization is executed to set ½ values to 0 for TimedTran and ImmediateTran. Now (c) another Incomp propagation rule finds that an abstract Tran needs to be refined into either a TimedTran or an ImmediateTran thus it changes their 0 value to the inconsistent value ↯ (both 0 and 1 at the same time). (d) If a materialization step is now executed then the inconsistent object is removed.

Below, we define the different merge functions for partial models:

1. *Propagation rules* handle type inferencing over 4-valued logic. A propagation rule (detailed in Figure 3.4) takes the form $prop = \frac{\varphi(v_1,\ldots,v_n)}{\alpha_i\uparrow \cdots \alpha_k\downarrow}$, where φ is a precondition, and $\alpha_i\uparrow$ (known to be true) and $\alpha_k\downarrow$ (known to be false) are atomic *actions* over the free variables of φ. For every match Z of φ (with $1 \succcurlyeq [\![\varphi(v_1,\ldots,v_k)]\!]^P_Z$), we obtain a merge function $prop_Z$ from P to a new partial model Q with $\mathcal{O}_Q = \mathcal{O}_P$, $prop_Z(o) = o$, and $\mathcal{I}_Q$ is obtained from $\mathcal{I}_P$:

$$[\![\alpha]\!]^Q_Z = \begin{cases} [\![\alpha]\!]^P_Z \oplus 1, & \text{if } \alpha\uparrow \text{ is an action of } prop, \\ [\![\alpha]\!]^P_Z \oplus 0, & \text{if } \alpha\downarrow \text{ is an action of } prop, \\ [\![\alpha]\!]^P_Z, & \text{otherwise.} \end{cases}$$

   The function $prop_Z$ is a merge function, because both $A \oplus 1$ and $A \oplus 0$ respect the refinement $\succcurlyeq$ of logical values.

2. *Object merge om*: $\mathcal{O}_P \to \mathcal{O}_Q$ merges two distinct objects $o_1, o_2 \in \mathcal{O}_P$ into a joint object $o_{1,2} \in \mathcal{O}_Q$ if $true \succcurlyeq \mathcal{I}_P(\sim)(o_1, o_2)$ and leaves the object unchanged otherwise. Formally, $\mathcal{O}_Q = \mathcal{O}_P \setminus \{o_1, o_2\} \cup \{o_{1,2}\}$, and $\mathcal{I}_Q$ is obtained by combining the contents of the two elements of $\mathcal{I}_P$ with $\oplus$ i.e.

$$\mathcal{I}_Q(\mathsf{C}_i)(o) = \begin{cases} \mathcal{I}_P(\mathsf{C}_i)(o_1) \oplus \mathcal{I}_P(\mathsf{C}_i)(o_2), & \text{if } o = o_{1,2}, \\ \mathcal{I}_P(\mathsf{C}_i)(o) & \text{otherwise.} \end{cases}$$

   The function om_{o_1,o_k} is a merge function, because $\oplus$ respects the refinement $\succcurlyeq$ of logical values.

3. *Concretization* is a merge function *conc*: $\mathcal{O}_P \to \mathcal{O}_Q$ that refines a partial model P to a concretized (partial) model Q by setting all ½ values to 0. Partial model Q can only contain 0, 1 and ↯ values. If Q has no ↯ values then it is a concrete instance model. Concretization preserves partial model refinement, i.e., $P \succcurlyeq Q$.

A *materialization* function *mat*: $\mathcal{O}_P \to \mathcal{O}_Q$ takes a *concretized* partial model P and removes all inconsistent elements by setting all ↯ values to 0 to obtain an instance model Q. In general, materialization is not a merge function (as $P \not\succcurlyeq Q$), since information preservation is violated when rewriting predicates (↯ ↦ 0). However, if a concretized (partial) model is free from ↯ values, then materialization is a trivial merge function due to being idempotent. Materialization is non-invasive, as it keeps all valid model elements in a concretized model, but removes inconsistent model elements to make the instance model EMF-compliant (e.g., serializable).

```
‹view› ::= ‹rule› (‹rule›)*
‹rule› ::= rule ‹pattern-dec› (=> ‹pattern-dec›)? (‹lookup›)*
‹lookup› ::= lookup ‹pattern-dec› => ‹param-list›
‹pattern-dec› ::= ‹pattern-name› ‹param-list›
‹param-list› ::= (‹variable› (, ‹variable›)*)
‹pattern-def› ::= ‹pattern-dec›; ‹pattern-body› (or ‹pattern-body›)*
‹pattern-body› ::= {‹constraint›; (‹constraint›;)*}
‹constraint› ::= C_i(‹variable›) | C_i.R_i(‹variable›, ‹variable›)
               | ‹variable› == ‹variable› | ‹variable› != ‹variable›
               | (find | neg find | count find) ‹pattern-dec›
               | (check | eval) (‹expression›)
```

Figure 3.5: A compositional view transformation language

Correctness of merging partial models Computations over 4-valued partial models carried out by a sequence of merge functions and finalized by concretization and materialization. Formally, if P is a partial model and $m = m_k \circ \cdots \circ m_m$ a *maximal* sequence of propagations and object merges applied to P, then $Q = (mat \circ conc \circ m)(P)$ is an instance model and $1 \not\geqslant [\![\varphi_{MM}]\!]^Q$, where φ_{MM} is the disjunction of error patterns corresponding to enforced metamodel constraints from Section 3.2.1. Therefore the final result is always a valid instance model. Moreover, if we have $[\![\varphi_{MM}]\!]^P = 0$, then $P \geqslant Q$, which means that no information is lost.

Proof sketch: As merge functions are closed over composition, m and $conc \circ m$ are merge functions. Propagation rules ensure that $[\![\varphi_{MM}]\!]^{m(P)} \in \{0, ½, ↯\}$. By changing ½ values in $\mathcal{I}_{m(P)}$ by $conc$, we obtain $[\![\varphi_{MM}]\!]^{(conc \circ m)(P)} \in \{0, ↯\}$, and the ½ values are removed from $\mathcal{I}_{(conc \circ m)(P)}$. Lastly, because enforced metamodel constraint violations φ_{MM} can be corrected by removing objects, $[\![\varphi_{MM}]\!]^Q = 0$, hence mat ensures that Q is an instance model.

If we initially have $[\![\varphi_{MM}]\!]^P = 0$, i.e. the partial model is surely valid, m (or any other sequence of propagations and object merges defined within this paper) does not introduce any ↯ values to $\mathcal{I}_{m(P)}$. Thus mat is the identity function and $P \geqslant Q$. Otherwise a portion of objects is removed when obtaining Q to avoid violating metamodel constraints φ_{MM}.

3.3 View model transformations

In this section we propose a view transformation language with relation-based composition along with a reactive, incremental, validating and inconsistency-tolerant execution engine. The view transformation is based on 4-valued partial models.

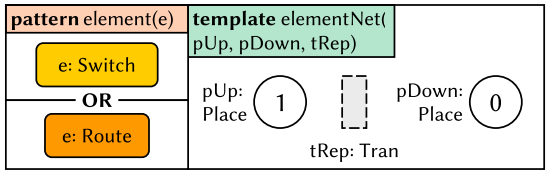
3.3.1 View definition by graph patterns

In this paper we introduce a declarative and fully compositional view transformation language based on graph queries. We reuse the Viatra Query Language [Ujh+15] to form view transformation rules by using pairs of *precondition* patterns, *template* patterns and *lookups* to reference (matches of) other transformation rules.

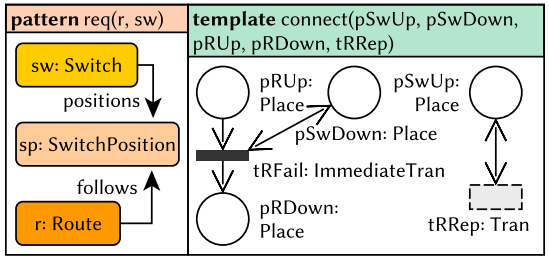
A graph pattern captures structural constraints with a graph predicate. In the concrete syntax of Viatra (see Fig. 3.5), a pattern is declared (‹*pattern-dec*›) by a unique name (‹*pattern-name*›), and a list of formal pattern parameters (‹*param-list*›). The predicate of a pattern is defined by a disjunction of pattern bodies (‹*pattern-body*›) connected by the **or** keyword. A pattern body

Railway model transformation (tr_1):

```
rule element(e) => elementNet(pUp, pDown, tRep);
```

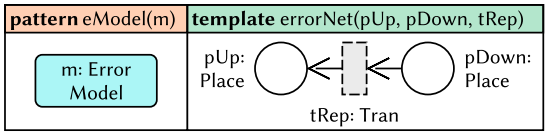



```
rule req(r, sw) =>
    connect(pSwUp, pSwDown, pRUp, pRDown, tRRep) {
    lookup element(r) => (pRUp, pRDown, pRRep);
    lookup element(sw) => (pSwUp, pSwDown, _);
}
```

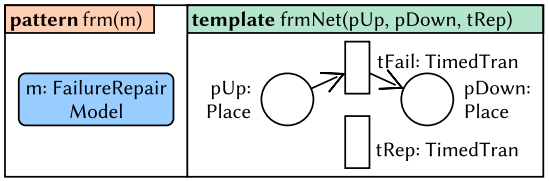


Dependability model transformation (tr_2):

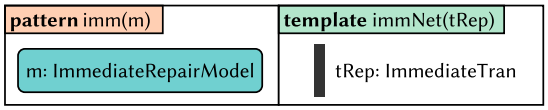
```
rule eModel(m) => errorNet(pUp, pDown, tRep);
```



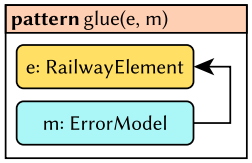

```
rule frm(m) => frmNet(pUp, pDown, tRep) {
    lookup eModel(m) => (pUp, pDown, tRep);
}
```




```
rule imm(m) => immNet(pUp, pDown, tRep) {
    lookup eModel(m) => (pUp, pDown, tRep);
}
```



Glue transformation (g):

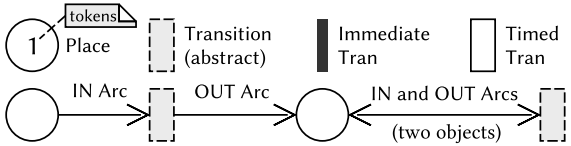



```
rule glue(e, m) {
    lookup element(e) =>
        (pUp, pDown, tRep);
    lookup eModel(m) =>
        (pUp, pDown, tRep);
}
```

```
pattern frm(m) { FailureRepairModel(m); }

@Template pattern frmNet(pUp, pDown, tRep) {
    Place(pUp); Place(pDown);
    TimedTran(tFail); TimedTran(tRep);
    Arc(aUpFail); Arc.kind(aUpFail, ArcKind::IN);
    Arc.place(aUpFail, pUp); Arc.tran(aUpFail, tFail);
    Arc(aFailDown); Arc.kind(aFailDown, ArcKind::OUT);
    Arc.tran(aFailDown, tFail); Arc.place(aRepUp, pDown);
}
```

(b) Precondition pattern frm and template frmNet with VIATRA Query textual syntax.



(c) Graphical syntax for stochastic Petri nets.

(a) Rules for the railway model (tr_1), dependability model (tr_2) and glue (g) transformation definitions, which are composed to obtain the transformation $tr_1 \|_g tr_2$.

Figure 3.6: View transformation rules for Train Benchmark dependability example

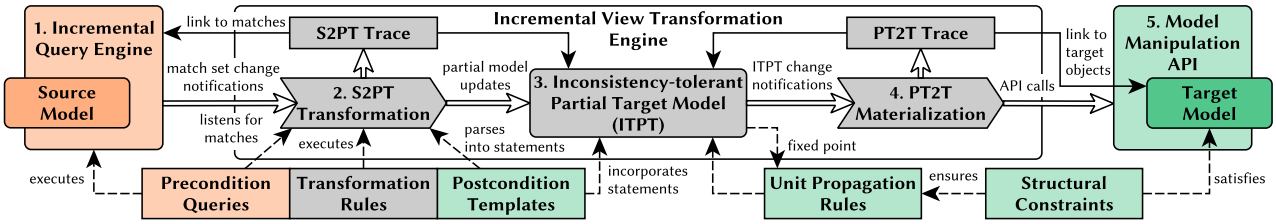


Figure 3.7: Overview of the view transformation

contains a conjunction of constraints that can be type and reference checks ($C_i(\,)$ and $R_i(\,,\,)$), equivalence check (==), positive, negative and aggregated pattern calls to compose complex patterns (resp. **find**, **neg find** and **count find** keywords), or external Java source code (using **check** or **eval** keywords) for attribute checks.

As templates of a view transformation rule, we define a restricted set of graph patterns (denoted by the underlined part of Fig. 3.5), which disallows multiple bodies, inequality constraints, negative and aggregated pattern calls, and **check** or **eval** expressions. In summary, a template pattern is a conjuction of atomic constraints.

A view transformation definition consists of a set of view transformation rules, where each rule consists of a (i) a *precondition pattern for the source language*, (ii) a(n optional) *template pattern for the target language* built from a restricted subset of pattern language elements, and (iii) a list of *lookups* for traceability links and parameter bindings. A *lookup* refers to implicit traceability links between source and target elements created when the source pattern was matched and the corresponding target elements were created by the transformation rule referred in the lookup.

> **Example 3.4** View transformation rules of our running example are defined in Fig. 3.6. A detailed description is provided for the *frm* rule (for dependability transformation) in Fig. 3.6b. Its precondition pattern matches a single FailureRepairModel element m, assuming that the *eModel* rule has already been applied in the context of m as defined by the corresponding lookup. As a result of the rule, the *frmNet* template is applied on the target model, which specifies the creation of two places (pUp and pDown), two TimedTran elements (tFail an tRep), and two corresponding Arcs between them (from pUp to tFail and from tFail to pDown). However, due to the right side lookup directive, the two places pUp and pDown as well as the transition tRep need to be merged with corresponding target Petri net elements already created when rule *eModel* was applied – as defined by the unification introduced by identical variable names.

3.3.2 Execution of view transformations

View models are constructed in four steps as shown in Fig. 3.7.

(1) First, each view transformation rule creates a partial model representing the application of a template predicate in isolation. Next, (2) the partial models are merged together by linking different view fragments along equivalences $\sim$ based on the lookups in rules. After that, (3) the merged partial model is refined by various merge functions to enforce target metamodel constraints. Finally, (4) as the merged view may contain inconsistencies due to the contradicting view specifications, a materialization step operation removes $\lightning$ values from the partial model to end up with a regular target instance model.

I. Reactive (source-incremental) execution. First, the precondition φ^S of a rule R is matched against the source model by calculating the match set $ZS = \{Z \mid 1 \geqslant [\![\varphi^S]\!]^P_Z\}$. We explicitly reuse existing features of the Viatra framework. Changes in the match set of source predicates are handled by using the incremental graph query engine of Viatra [Ujh+15]. All subsequent

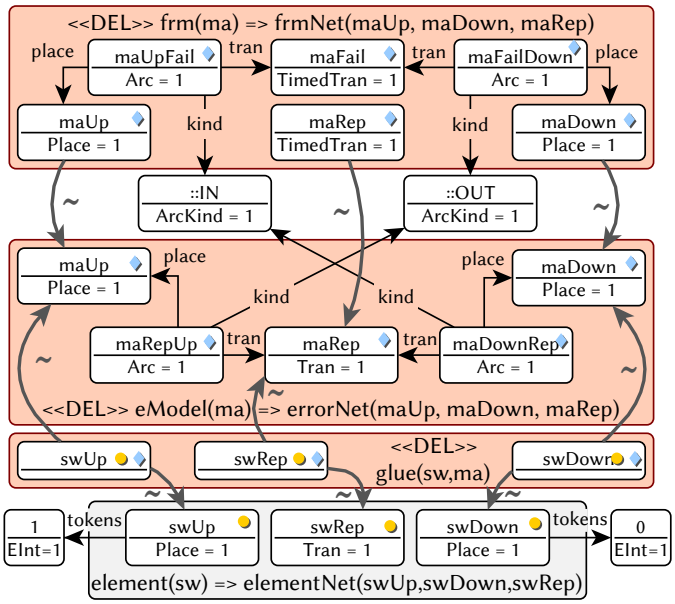


Figure 3.8: Initial partial model derived from predicates with the effects of a source change shown as «DEL» stereotypes

processing steps in our engine are triggered and executed as a reactive transformation [Var+16], therefore our entire engine becomes *reactive*.

II. Template instantiation and model merge. Then each rule R is applied independently. For each match $Z \in ZS$ of rule R a template partial model $T = \langle O_T, \mathcal{I}_T \rangle$ is created for each rule according to the target predicate φ^T. This T is constructed as:

- Each variable v of φ^T is mapped to an object of O_T
- Constraints of φ^T are translated to a 1 value in $\mathcal{I}_T$:
  - If there is a $\mathsf{C}_i(v)$ in the predicate φ^T and the variable v is mapped to an object o, then $\mathcal{I}_T(\mathsf{C}_i)(o) = 1$.
  - If there is a $\mathsf{R}_j(v_1, v_2)$ in the predicate φ^T, and the variables v_1, v_2 are mapped to objects o_1, o_2, then $\mathcal{I}_T(\mathsf{R}_j)(o_1, o_2) = 1$.
  - If there is a $v_1 \sim v_2$ in the predicate φ^T and the variables v_1, v_2 are mapped to objects o_1, o_2, then $\mathcal{I}_T(\sim)(o_1, o_2) = 1$.
- Every other values in $\mathcal{I}_T$ are set to ½.

Next, each independently created template partial model $\{T_1, \ldots, T_n\}$ is copied together into a merged partial model $MP = \langle O_{MP}, \mathcal{I}_{MP} \rangle$ in order to represent all templates and lookups.

- O_{MP} consists of the union of objects of the template partial models: $O_{T_1} \cup \cdots \cup O_{T_n}$.
- $\mathcal{I}_{MP}$ is the same as the $\mathcal{I}_{T_i}$ of template partial model T_i: for each objects $o_1, \ldots, o_n$ in a template model T_i, and for each symbol $\alpha \in \Sigma$: $\mathcal{I}_{MP}(\alpha)(o_1, \ldots, o_n) = \mathcal{I}_{T_i}(\alpha)(o_1, \ldots, o_n)$
- Between the templates, **lookup** rules set additional $\mathcal{I}_{MP}(\sim)$ to 1 to add connections between templates.
- In all other cases $\mathcal{I}_{MP}(\alpha)(o_1, \ldots, o_n)$ is ½.

The partial model PM obtained after this step is *para-complete* [Bel77; KO17], thus it may contain ½ and 1 values, but no 0 and ↯ values.

> **Example 3.5** Fig. 3.8 illustrates the application of the *frm* rule (from Fig. 3.6) for the source models of *Rail* and *Dep* from Fig. 3.1.
> First, the precondition of the rule *frm* checks for the existence of an FailureRepairModel element, and then the template *errorNet* is applied. As a result, the bottom part of the partial

model (marked by a dashed rectangle) is created with model elements corresponding to the template.
Since the rule contains a lookup to another rule *eModel*, partial model elements created by the two rules need to be merged. This is initiated by adding equivalence relations ∼ between the corresponding elements defined by variables such as maUp, maDown and maRep.
Similar equivalences are declared by applying other transformation rules from Fig. 3.6 and Fig. 3.8 presents the entire partial model derived by all rules. The *glue* rule is a special view transformation rule where no target elements are created but only equivalences are declared.

III. Reactive object merge and propagation. By now, all objects of the partial model created by different templates are identified to be merged by marking them with equivalence relations. The merge functions defined for inconsistency tolerant partial models in Section 3.2.3 are executed in an incremental way.

Each propagation rule $prop = \varphi/\alpha_i$ has a graph predicate φ as a precondition which can be captured by a regular graph query evaluated over 4-valued logic. The execution of a propagation rule can be carried out reactively by extending the constraint rewriting technique [SV17] to provide 2-valued *may* and *must* graph predicates for under- and over-approximation. For the incremental execution of an object merge *om*, we rely upon incremental maintenance techniques for strongly connected components used for graph queries with transitive closure [Ber+12b].

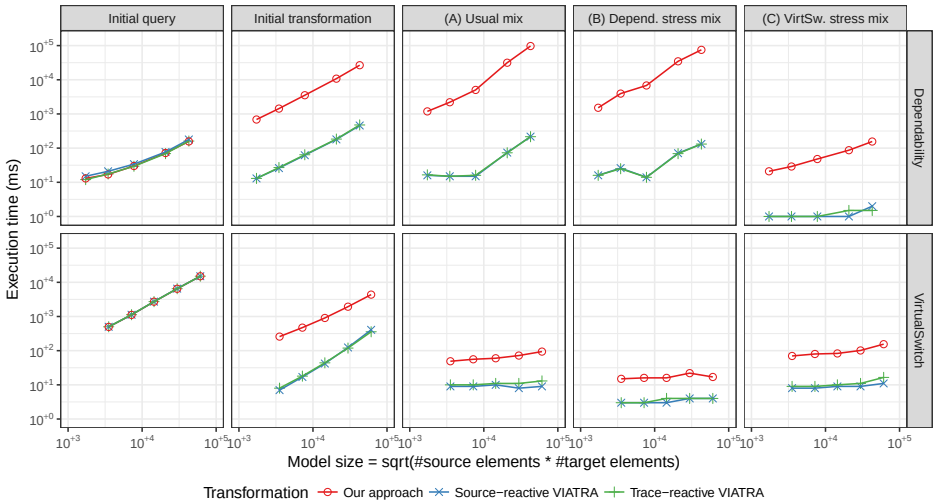
As a result of this step, all ½ values are removed, and all equivalent objects (marked by ∼) are merged, thus the partial model becomes *para-consistent* [Bel77; KO17] as it contains only 0, 1 and ↯ values. However, during the propagation phase, the partial model may contain both uncertain ½ and inconsistent ↯ values.

Example 3.6 The effects of object merge and propagation rules were illustrated in Fig. 3.3a. The two swRep objects of the partial model created by rules *element* (yellow dot) and *frm* (blue diamond).
The Fig. 3.3b case corresponds to a hypothetical source change where the match of rule *frm* no longer exists, thus the effects of the template need to be removed. The exact merge procedure was discussed in Example 3.3. Templates removed from the partial model due to this source change are shown as «DEL» stereotypes in Fig. 3.8.

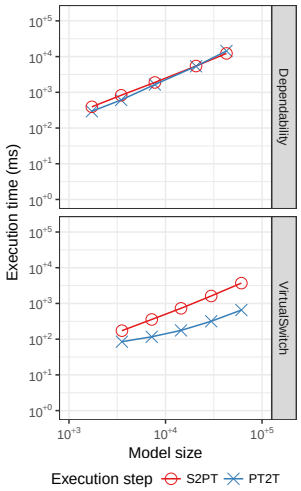
IV. Incremental materialization. At the final step, erroneous elements of the target model are removed by a materialization step. After materialization, the partial model is equivalent to the target instance model, thus (1) all structural constraints of the target metamodel are ensured in accordance with the correctness of merge functions (see Section 3.2.3), hence our technique is *validating*. Moreover, (2) each change in this final partial model can be incrementally propagated to the target instance model, hence our approach is (target-)*incremental*. If a source model change does not affect a view model, then no change is propagated to the target view model. Therefore, (3) our approach is *hippocratic*.

Concerning the (source-target) *consistency* of our approach, we need to separate the case when no ↯ symbols need to be removed during materialization. In such a case, all steps are valid refinement steps, thus it is guaranteed that the final model P refines all applied templates T_i ($T_i \succcurlyeq P$) which ensures consistency. If an ↯ symbol is removed during materialization, then the cause of this inconsistency can be shown by a corresponding match of a propagation rule precondition tracing the found issue back to the applied templates, the source model and the enforced structural constraint of the target metamodel.

Example 3.7 If all the propagation steps are executed for the partial model of Fig. 3.8 then the target Petri net instance model of Fig. 3.1 is obtained.



(a) Complexity of source query initialization, initial transformation, and the sychronization of a (A) balanced mix of modifications of 100 operations, and modification mixes of 100 operations focused on stressing the (B) Dependability, (C) VirtualSwitch transformation.



(b) Complexity of the two execution phases in our approach during initial transformation.

Figure 3.9: Measurement results

3.4 Evaluation

3.4.1 Research questions

Our view transformation approach is fully implemented as an open source project[2]. We carried out an experimental evaluation to address three research questions:

RQ1. What is the complexity of different execution phases in our view transformation engine?

RQ2. What is the performance overhead for the initial run of our view transformation engine compared to reactive imperative transformations with explicit traceability?

RQ3. What is the performance overhead for change-driven behavior of our view transformation engine compared to reactive imperative transformations with explicit traceability?

3.4.1.1 Case studies

We selected two substantially different view transformation challenges for our investigation. (1) *Dependability* is an extended version of the case study used in this paper which aims to compose two separate transformations in a way that the target Petri net model is significantly larger than any of the two source models. (2) *VirtualSwitch* is a filtering transformation taken from [Deb+14] where the size of the source model is significantly larger than the size of the target model. We believe that these transformations are representative for key practical applications of view transformations: the *VirtualSwitch* scenario is typical for in traditional view models with information loss [Bru+17] while the transformation challenges in the *Dependability* case are common for the formal analysis of extra-functional properties of systems [MPB02; Gil+10].

[2] `https://github.com/ftsrg/viewmodel`

3.4.2 Compared approaches

First, we instrumented our *ViewModel* transformation approach to enable the clear separation of different transformation phases to address **RQ1**. Then we compare our approach with two different view transformation styles available in Viatra[3]. These solutions use an *explicit traceability model* (vs. implicit traceability in our approach) and *imperative actions* in transformation rules using Java/Xtend (vs. declarative query-based templates). However, differences in query performance can be mitigated to a large extent. (i) The *source-reactive* solution [Deb+14] uses exactly the same source queries as our view transformation approach, but rule priorities had to be set carefully. (ii) The *trace-reactive* solution [Heg+16] uses queries with both source and traceability elements as part of its precondition. Since both the level of compositionality and the properties of the view transformation engine are different in these approaches compared to our view transformation approach (see Section 3.5), our evaluation may reveal the performance trade-offs of the increased expressiveness of our approach.

3.4.3 Experiment setup

To investigate the initial transformation runs (**RQ2**), our measurement setup contains 5 source models of increasing size. For the *Dependability* case, the source models ranged 1K to 25K while the target models ranged from 3K to 72K. For the *VirtualSwitch* case, the source models were ranging from 25K to 425K elements, while the target models were ranging from 500 to 9K elements. In each case, we measured the initial time for populating the caches of queries and the execution time of the first transformation, while the load time of source models was excluded. To address **RQ1**, we measure how much time the different phases of our view transformation approach takes during this initial run.

To investigate the change-driven behavior (**RQ3**), we first created 10 different elementary changes (modifications of one element) and 5 change mixes containing 100 elementary changes each (with fix ratio between different types of change within each mix). Due to space restrictions, we only present results for 3 change mixes within the paper, while all other measurements (and plots) are available in [d21]. Change mix (A) presents a balanced mix of changes, while types of changes in mixes (B) and (C) were selected from those elementary changes that caused longer synchronization times in the *Dependability* and *VirtualSwitch* cases, respectively.

Each experiment was executed 30 times after 10 warmup runs on a cloud-based virtual environment (with 4 CPU, 16 GB memory and 8 GB disk size) on Amazon AWS.

3.4.4 Results

Our evaluation results comparing the performance of core reactive Viatra transformations and our view model approach are presented in Fig. 3.9a where the two Viatra transformations (source vs. trace-reactive) have very similar behavior. The two key internal phases of our approach separating the source-to-partial model (S2PT) transformation and partial-model-to-target (PT2T) materialization stages (with propagation and concretization) are presented in Fig. 3.9b.

Since the *VirtualSwitch* case is dominated by the size of the source model while the *Dependability* case is dominated by size of the target model, the logarithmic horizontal (x) axis presents a combined model size as the geometric mean ($\sqrt{|src| * |trg|}$) of source and target model sizes (i.e. number of objects) which is compatible with the logarithmic scale of the plots. The logarithmic vertical (y) axis presents the execution times (in ms).

[3]Our repository contains an implementation of the transformations in batch ATL and a partial implementation in eMoflon, but the different performance optimizations in those tools would disallow to separate query performance from transformation performance.

The intermediate partial model for the largest source models had (1) 222K partial model variables and 401K partial model atomic statements to represent 72K target objects (*Dependability*) and (2) 38K partial model variables and 58K partial model atomic statements which represents 8K target objects (*VirtualSwitch*).

3.4.5 Discussion

Based on these experimental results, we make the following observations related to the research questions:

RQ1: Both major view transformation phases seem to grow polynomially in model size, but more data points (model sizes) would be necessitated for a firm statement.

Dependability: The construction of the partial target model and its materialization are both challenging. The S2PT phase (0.4 s on smallest, but 12 s on largest) and the PT2T (0.3 s on smallest, but 14 s on largest) were within 0.5 orders of magnitude, while PT2T wass slower on large models as it has to perform type inferencing and complex object merges.

VirtualSwitch: The key challenge is to filter the source model, thus the intermediate partial model is smaller and necessitates fewer complex merges than above. Thus PT2T was 1 order of magnitude faster (S2PT 3.7 s on largest vs PT2T 0.65 s on largest).

RQ2: The initial query took exactly the same time (0.15 s for largest *Dependability*, 150 s for largest *VirtualSwitch*) for each implementation of the transformations, because the same queries and the same query engine (Viatra) was used, thus our measurements highlight the differences in the transformation phase. There was a 2 orders of magnitude difference in *Dependability* (26.7 s vs 0.48 s on largest), and 1 in VirtualSwitch (4.4 s vs 0.4 s on largest) between execution times in favor of reactive Viatra transformations.

RQ3: In the *Dependability* case, we observed 2.5 orders of magnitude difference in mixes (A) and (B) which cause major changes in the target model (94 s vs 0.2 s on largest). In mix (C), which cause significantly fewer target changes as only attributes of places are modified, Viatra was instantaneous, but our approach also took only 10–150 ms depending on model size to process the change.

In the *VirtualSwitch* case, Viatra was instantaneous even in the modification mix specifically designed to cause target model changes. In (A) and (C), our approach took around 100-150 ms, which is significantly less than the initial transformation.

3.4.6 Conclusion

Our approach is more sensitive to target model size than source model size. The incremental behavior of our approach is also dominated by the size of the implied target change. For small target deltas, the overhead of our approach was less than 150 ms. The S2PT phase takes more time for complex model filtering and weaving challenges, while PT2T is slower when it has to materialize a large partial model. Unlike reactive Viatra [Deb+14; Var+16], our approach achieves compositional and consistent view transformations (i.e. no manual adaptations to compose the original transformations). The performance penalty of this increased expressiveness is about 1–2 orders of magnitude increase in execution time compared to an industrial model transformation engine.

3.4.7 Threats to validity

To mitigate *internal validity*, 10 warm-up runs were included prior to the measurements to decrease the fluctuation of runtime caused by JVM. While our measurements were executed in the cloud (AWS), the same virtual machine was used for comparing the different approaches in a fair way.

To address *external validity*, we selected two transformations with substantially different characteristics (massive filtering in *VirtualSwitch* vs. complex merging in *Dependability*). Train Benchmark models serve as a common source model used in both cases, which may reduce the generalizability of our result to other domains. However, the Train Benchmark [Szá+17] has been actively used within the MDE community as a performance benchmark for different query and transformation tools, thus external validity is not compromised.

3.5 Related work

A desired view transformation approach offers a fully compositional *language* and a reactive, incremental, consistent and validating *engine* but no transformation tools currently exist which support all these properties. Our overview of (the significant amount) of related work primarily focuses on existing transformation tools by categorizing the level of support (1) for parallel compositionality in transformation languages, and (2) for desirable transformation engine properties in Table 3.1. For space considerations, we highlight only the typical restrictions found in the context of multiple tools.

Imperative transformation approaches reactively build the target model (like imperative ATL, Viatra or ETL) but they do not provide consistency guarantees, i.e. certain target models may not be consistent with a source model. Unfortunately, such inconsistencies can propagate to future stages of the transformation.

Bidirectional model synchronization tools (like different TGG implementations or JTL) either guarantee consistency or they abort the execution of the transformation. These tools offer a certain level of serializability, but they are not fully compositional.

Dedicated view transformation approaches (like TGG Views, Viatra Views, Reactive ATL) use a restricted transformation language (wrt. their regular transformation counterpart) to provide desirable engine behavior. However, parallel composition of different transformations is very limited.

Most existing fully compositional approaches (like QVTr, ramification, Epsilon with combined ETL and EML languages) are neither reactive nor incremental and only EML is validating. Only GRoundTram and ATLGT support target incrementality and delta-based source incrementality, but over a custom (non-EMF compliant) model representation. The closest approaches to ours are [HLR06; Cic+10] as they build a knowledge base based on first order logic and target models are derived by logical inference, but these approaches are not fully compositional.

Our work provides a view transformation approach with (1) a *fully compositional* transformation language built on top of an existing declarative query language, and (2) a transformation engine which is *reactive*, *incremental*, *validating* and *inconsistency-tolerant* at the same time. The inconsistency-tolerant engine is a relaxed version of a consistent engine where $Trg^{(j)} \neq tr(Src^{(j)})$ may happen after some conflicting source model changes $Out^{(j)} = exec(tr, \Delta_{Src}^{(j)})$, but all other desirable properties are preserved. Most of the target model satisfying MM_T is preserved, while inconsistencies are explicitly highlighted by the framework. Lastly, by delaying notifications to engine, *reactive* behavior can be optionally replaced with *delta-based* processing.

3.6 Conclusions

We proposed a fully compositional view transformation language executed by a reactive, incremental, validating and inconsistency-tolerant view transformation engine. Our approach reuses the Viatra Graph Query Language [Ujh+15] to define target fragments which are merged during transformation using the concepts of inconsistency tolerant partial models based on 4-valued logic foundations to gracefully handle temporal inconsistencies during transformations.

Table 3.1: Comparison of view model transformation techniques.

		Engine properties				
	Parallel composition	React.	Incr.	Cons.	Valid.	Comment
Our approach	relation-based	R	●	IT	●	
Reactive ATL [JT10; MTD17]	independent	R	●	C	●	Restrictions in source and trace language
TGG Virtualized View [JKS06]	independent	R	●	C	○	Only single node or reference in rule target, limited NAC
TGG Materialized View [Anj+14]	independent	R	●	C	●	Only single node or reference in rule target, limited NAC
Viatra Views [Deb+14]	independent	R	●	C	●	Only single node or reference in rule target
QueST [GDM14]	independent	D	●	C	●	Only single node or reference in rule target
Incremental QVTr [Son+11]	restricted serializable	R	●	?	?	Cons., valid. difficult to determine due to QVTr semantic issues [Gre06; Ste10]
EMF Views [Bru+15]	independent	NR	○	C	○	Infers target metamodel
Active Operations [Bea+10]	restricted serializable	R	●	C	○	Transformation also defines target metamodel
Hearnden et al. [HLR06]	restricted serializable	D	●	IT	○	Produces deduction tree tree as target model
ATL (no imperative code) [Jou+08]	restricted serializable	NR	○	C	●	Restrictions on outgoing references in non-primary rules
ATL (+imperative code) [Jou+08]	serializable	NR	○	NC	●	No consistency checking for imperative actions
eMoflon TGG [Lau+12; Leb+17]	restricted serializable	D	●	C/A	●	Restrictions for negative application conditions (NAC)
Viatra [Var+16; Heg+16]	serializable	R	●	NC	●	No consistency checking for imperative actions
QVTr [QVT] M2M [Wil17]	ID-based	NR	○	?	?	Cons., valid. difficult to determine due to QVTr semantic issues [Gre06; Ste10]
Epsilon ETL [KPP] + EML [KPP06]	relation-based	NR	○	NC	●	Merge operators for composition in separate language
JTL [Cic+10]	serializable	D	○	C/A	●	No answer if the target cannot satisfy constraints
RAMification [Küh+10; Mey16]	ID-based	NR	○	C	○	Metamodel constraints are *relaxed*
GRoundTram, ATLGT [Hid+11; HT16]	relation-based	D	●	C	○	Graph bisimulation based data model, non-EMF
BiGUL [KZH16]	relation-based	NR	○	C/A	●	PutBack-based functional programming may be adapted to EMF [Anj+17]

Legend: R reactive, D delta-based, NR non-reactive (batch); C consistent, C/A consistent or aborts, NC non-consistent, IT inconsistency tolerant; ● yes, ○ no

The execution engine reuses existing support for incremental graph queries as available in the Viatra framework [Var+16] to provide reactive behavior, while graph predicates used in merge functions also enable incremental propagation of changes while ensuring structural constraints of the target language.

Our experimental evaluation also highlighted that such an increased expressiveness on the view transformation language level does not come for free as the core (imperative and reactive) Viatra engine executes 1-2 orders of magnitude faster for the case studies – but the individual transformations had to be modified manually to achieve the necessitated merge functionality.

The detailed evaluation of the different execution phases also points to key directions for future work for a hybrid view transformation engine. A sophisticated static analyzer may automatically reveal transformation rules where compositionality falls into a more simple class, thus many optimizations available in existing view transformation tools would become amenable to improve performance. Nevertheless, our view transformation approach already provides strong support for the most challenging composition problems for a very expressive view transformation language.

Chapter 4

Multiplicity reasoning for consistent graph model generation

Model-based systems engineering frequently uses complex modeling tools, like Capella, Artop, Matlab Simulink or Yakindu Statecharts. When these modeling tools are used in safety-critical systems, safety standards (like DO-330 [RTC11] for avionics systems) may prescribe that (1) only the output of a *qualified tool* can be trusted, and (2) such a tool should meet the same requirements as the critical system component it designs. However, such quality assurance for the software running in modeling tools is very complex, which makes tool qualification an extremely costly process. As such, automated techniques for synthesizing effective test suites used in the software quality assurance of complex modeling tools would be highly beneficial.

The automated synthesis of high-quality test cases is a recurrent challenge in many areas of software and systems engineering in order to simultaneously improve quality and productivity. Since test cases created manually by engineers can easily miss important corner-cases of specifications, certain application areas (e.g. safety-critical software) substantially rely on such automated test case generators.

This chapter focuses on *automated model generators* which represent tests in the form of graph models. This is a subclass of generators with high practical relevance but also high complexity. For example, graphs may models complex test stubs in object-oriented programs [Mil+07; KM04] (e.g. nodes are objects, edges are pointers). The quality assurance of smart cyber-physical systems (CPS) can rely upon prototypical test contexts given in the form of graphs [Mic+12; Ben+18; Iqb+15]. Model generators are also beneficial for testing modeling tools [Sem+17].

Further practical application scenarios are investigated in [Var+18], which identifies a long-term research agenda aiming to provide desirable high-level properties for automated model generators. Using the terminology of [Var+18], an advanced synthetic model generator should be domain-customizable, consistent, diverse, realistic and scalable.

For domain customizability, we use precise underlying specification techniques to capture the domain concepts and their relations captured in the form of a metamodel, while *consistent* models can be further restricted by design rules or *well-formedness constraints* (defined as OCL constraints [OCL] or graph patterns [VB07; Ujh+15]).

There is a wide range of model generators such as Alloy [Jac02; TJ07], Formula [JLB11; JS06], USE [KHG11], UML2CSP [CCR14], SDG [SSB20; SSB17] and Viatra Solver [Sem+20c; Sem+19] to automatically derive *consistent* models for a given domain specification. Several generators are based on precise foundations offered by backend logic solvers (like SAT solvers [LP10; ES03] or SMT solvers [MB08]). These tools excel at finding inconsistencies (if they exist) by interpreting domain specifications as a logic problem, but they can only derive small consistent models. Moreover, they fail to derive a *diverse* set of models [JGS13; Sem+20c], which restricts their use

in practical testing scenarios.

Alternatively, logic reasoning or search-based techniques can be lifted directly on the level of graphs [SSB20; SSB17; SV17] for model generation purposes. These approaches scale better with respect to the size and diversity of the derived models, but they may fail to reveal inconsistencies in specifications.

Finally, the *realistic* nature of synthetic models can also be important in test generation scenarios. For example, realistic test models used for autonomous cars represent real test environments [Ben+18; Iqb+15] while unrealistic test cases (e.g. obscure traffic situations) are considered as false positives. Failures caused by realistic scenarios are more severe, as they have more chance to happen on real workload. Several examples in *testing software-intensive CPSs* [Ben+18; ACG17; Edu+18; Iqb+15; SSB17; SSB20] highlight this realistic aspect. Furthermore, the usability of automatically generated tests may be hindered by test cases that are not realistic (i.e., strange and difficult to comprehend for developers) [HM19].

Problem statement To increase the realistic nature of models, one needs more refined control over the structure of the auto-generated models. For example, *partial snapshots* [FSC12; SFC12] define model fragments that need to be extended by the model generator, thus it defines the expected *initial structure* of each models. Furthermore, *type scopes* [Jac02] allow to precisely define the required number of newly generated elements (per type/class), thus focusing the generation process on more relevant instance models of the target domain.

While logic solver-based model generators support various scope constraints, they have severe scalability issues and they fail to generate complex graphs (without isolated nodes or star structures) with more than 50-70 nodes for complex domains [SNV18]. The search-based approach [SSB20; SSB17] can generate a large number of simple graph models with fine-grained type distributions, but it is unable to derive large *and* connected consistent models. Finally, the graph solver [SNV18] can derive large and connected models, but it only allows to cap the total size of the model, and thus it is unable to fine-tune the models along type scopes.

Contributions In order to improve the scalability and usefulness of automated model generation, we propose a novel technique that combines the advantages of partial model refinement techniques [SV17] with numeric reasoning on model scopes. In particular,

- We define *a mapping of structural and well-formedness constraints into numeric constraints* that can be evaluated on scoped partial models.
- We use existing numerical solvers (i.e. IP and LP solvers) to efficiently guide the generator process.
- We extend an open source model generator [SNV18] with type scope support and integrate various IP and LP solvers to provide a software prototype tool.
- We evaluate the effectiveness of the approach on numerous case studies including a running example of a complex design space exploration challenge [Her+17] introduced by researchers at NASA Jet Propulsion Lab.

The current chapter builds upon but substantially extends past research results in [SNV18; Var+18; Sem+20c]. More specifically, the introduction and handling scope constraints are novel conceptual results of the current chapter. In order to maintain the favorable theoretical properties (e.g. completeness, diversity) of the generic model generation framework formally proved in [Var+18; Sem+20c], the refinement calculus is extended here to incorporate scopes. The prototype implementation builds on and extends [SNV18; Sem+19] by integrating various numerical solvers into the decision procedure. Finally, the experimental evaluation shows how novel results improve scalability and the realistic nature of models wrt. existing work.

Added value With multiplicity reasoning, graph generators can be configured by numeric constraints to focus model generation on the relevant fragment of models. Although a single metric cannot ensure the realistic nature of models, but *ensuring the realistic distribution of model elements* were found to be useful in [SSB20] as it filters out a wide range of surely unrealistic models. As such, automatically synthesized corner-cases will have higher practical relevance (e.g. test scenarios in autonomous driving will investigate relevant traffic situations).

With the help of numerical reasoning, graph generators will be able to measure and efficiently control the quantity of nodes. This significantly improves the performance of existing graph solver algorithms. Moreover, it enables a practical iterative workflow for test generation where initially, one can start with general scopes which are gradually refined to grow larger consistent models. Finally, by adhering to the refinement calculus, the generator continues to provide favorable properties such as consistency, completeness or diversity (but the in-depth investigation of such properties is out of scope for the work).

The contents of this chapter are based on the journal paper [j1].

4.1 Models and partial models

The computational design synthesis of *interferometry mission architectures* has been introduced in [Her+17] as a complex challenge for early mission planning for space missions of NASA where a designated architecture consists of collaborating satellites (of different size and capabilities) and radio communication between them. Each mission architecture involves multiple spacecrafts, which imposes an especially challenging design task. The authors of [Her+17] suggested a technique to automatically enumerate promising design candidates with respect to the requirements, technical and resource constraints, and mission objectives. We adopt this case study, described in more detail in Section 2.1, as a running example throughout this chapter.

In this section, we recall foundations of domain-specific modeling languages (DSLs) and graph-based instance models formalized as partial models using relational logic enhanced with integer linear constraints from Section 2.3. We also illustrate these concepts in the context of the case study.

As a technical foundation for domain modeling, we use EMF [Ste+09] metamodels and Viatra well-formedness constraints [VB07; Ujh+15], which are also used in industrial tools (including e.g., Capella, Artop, Yakindu, Papyrus, etc.) as well as in [Her+17]. Conceptually, the graph generation approach could be applied on other modeling formalisms too, e.g. UML Class Diagrams for defining the types and Object Constraint Language (OCL, [OCL]) for defining constraints as in [SAB09; SSB17].

4.1.1 Scoped partial models

In this paper, we represent architecture models using the *3-valued scoped partial models* introduced in Section 2.3 as an extension of partial models [SNV18]. We combine two techniques to capture uncertainty in a partial model. First, *3-valued logic* is used to explicitly represent uncertain *structural properties* of models with a third ½ (unspecified or unknown) truth value (besides 1 and 0, which stand for *true* and *false*) in accordance with [RSW04; Var+18; SV17]. Secondly, *quantitative information* is attached to the partial model to precisely represent the *known (or required) size* of the models. Later, we use partial models as states of model generation to represent intermediate solutions with uncertain parts denoted with truth-value ½ and its size.

As a difference from the general notations from Section 2.3, in this chapter, we will only consider linear inequalities over the number of objects $\hat{\varepsilon}(p) = \hat{p}$ represented by individuals

$p \in O_P$ of a partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \rangle$. Formally, in the signature $\langle \Sigma, \Gamma, \alpha \rangle$, we will set $\Gamma_C = \Gamma_R = \Gamma_F = \Gamma_X = \emptyset$ and let $\Gamma = \{\varepsilon\}$.

We will rely on the linear inequality variables $\hat{x} \in \mathcal{X}_P$ defined for each individual $x \in O_P$ of the partial model P to encode lower $L \leq C_i$ and upper $C_i \leq U$ type scope bounds. In contrast with including the class symbol $C_i \in \Sigma_C$ that is the subject of the scope constraints (i.e., $C_i \in \Gamma_C$), this approach reduces the number of variables and the number of inequalities in the system of linear inequalities $\mathcal{S}_P$. As a result, linear and integer programming solvers will be able to reason about $\mathcal{S}_P$ more efficiently.

From a formal perspective, the resulting partial modeling technique implements *predicate abstraction* [FFJ12; SV17] on graph model along with *counter abstraction* [BCK01; KK08] on the nodes of the graph model.

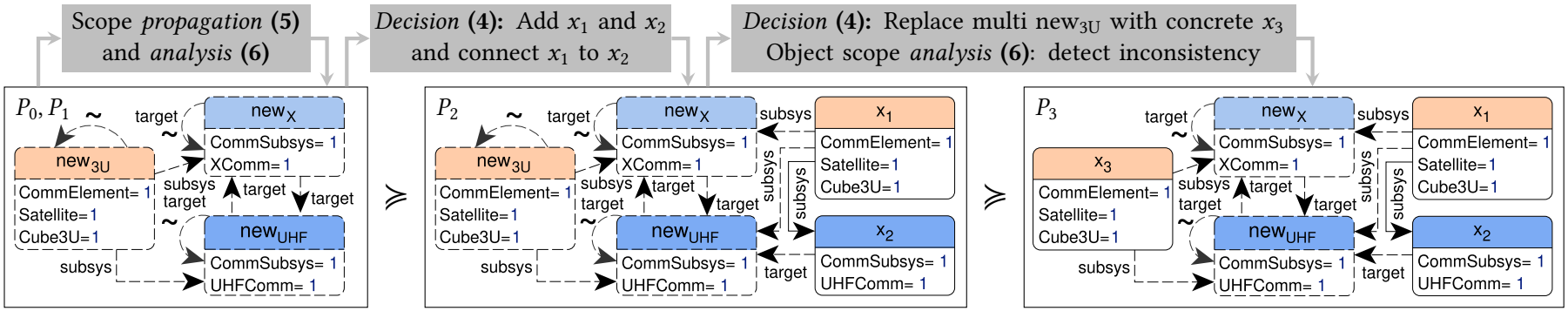
Example 4.1 Fig. 4.1 shows three partial models. Truth values of *class predicates* are denoted by labels on nodes (missing labels correspond to 0 values). *Reference predicates* with 1 and ½ values are denotes as solid and dashed arrows, respectively. Nodes with Dashed borders correspond to ½ values of the *existence* ε predicate. Uncertain *equivalences* are shown with dashed $\sim$ loops, but to reduce clutter, certain self-equivalences are not depicted. Thus, multi-objects have dashed borders and dashed $\sim$ loops and concrete objects are shown with solid borders.
In P_0 (on the left side of Fig. 4.1), the multi-object new_{3U} (with uncertain existence and self-equivalence) is certainly of type Cube3U, but not of type CommSubsys.
Partial models P_0 and P_1 define two systems of linear inequalities ($\mathcal{S}_{P_0}$ and $\mathcal{S}_{P_1}$ respectively) over the same three variables: $\widehat{new_{3U}}$, $\widehat{new_X}$ and $\widehat{new_{UHF}}$. In $\mathcal{S}_{P_0}$, the linear equation $\widehat{new_{3U}} + \widehat{new_X} + \widehat{new_{UHF}} = 10$ ensures that P_0 represent instance models with exactly 10 objects. A potential variable assignment $k\colon \widehat{new_{3U}} \mapsto 4, \widehat{new_X} \mapsto 3, \widehat{new_{UHF}} \mapsto 3$ is a possible solution of both $\mathcal{S}_{P_0}$ and $\mathcal{S}_{P_1}$, and represent models with 4 3U, 3 X and 3 UHF objects. As $\mathcal{S}_{P_1}$ contains more constraints than $\mathcal{S}_{P_0}$, $\mathcal{S}_{P_1} \vDash \mathcal{S}_{P_0}$ is holds trivially. However, $\mathcal{S}_{P_0} \vDash \mathcal{S}_{P_1}$ is not true as $k'\colon \widehat{new_{3U}} \mapsto 10, \widehat{new_X} \mapsto 0, \widehat{new_{UHF}} \mapsto 0$ is a solution for $\mathcal{S}_{P_0}$ but not for $\mathcal{S}_{P_1}$.

4.1.2 Refinement and concretization of PMs

We carry out model generation along a sequence of refinement steps (Definition 2.24) that derive new partial models by increasing their size but gradually reducing the level of uncertainty in each model while continuously checking (an approximated version of) well-formedness and scope constraints.

During refinement, the linear inequality systems are also refined with respect to the entailment relation. Informally, during the refinement of $\mathcal{S}_P$ into $\mathcal{S}_Q$, it (i) may split some variables into the sum of multiple variables (e.g., all occurrences of a variable $\widehat{x}$ in $\mathcal{S}_P$ are replaced with $\widehat{x_1} + \widehat{x_2} + \widehat{x_3}$ in $\mathcal{S}_Q$), and (ii) it may induce stricter constraint over the variables (e.g., $\widehat{x} \leq 3$ is refined to $1 \leq \widehat{x} \leq 2$).

Example 4.2 Figure 4.1 depicts three refinements $P_0 \succcurlyeq P_1$, $P_1 \succcurlyeq P_2$, and $P_2 \succcurlyeq P_3$. P_0 and P_1 have the same object set ($O_{P_0} = O_{P_1}$) and graph structure. Therefore, the refinement relation $ref_1 \subseteq O_{P_0} \times O_{P_1}$ is the identity relation $ref_1 = \{\langle p, p \rangle \mid p \in O_{P_0}\}$. Compared to $\mathcal{S}_{P_0}$, $\mathcal{S}_{P_1}$ contains an additional linear equation. Every solution of $\mathcal{S}_{P_0}$ is also a solution of $\mathcal{S}_{P_1}$, which ensures $P_0 \succcurlyeq P_1$.
The refinement relation $ref_2 \subseteq O_{P_1} \times O_{P_0}$ maps new_{3U}, new_X, new_{UHF} to the objects in P_1 with the same identifiers, while we also have $\langle new_{3U}, x_1 \rangle$ and $\langle new_{UHF}, x_2 \rangle \in ref_2$. The objects x_1 and x_2 were *split* from new_{3U} and new_{UHF}, respectively. To obtain $\mathcal{S}_{P_2}$, we replaced each occurrence of $\widehat{new_{3U}}$ and $\widehat{new_{UFH}}$ with $\widehat{new_{3U}} + \widehat{x_1}$ and $\widehat{new_{UFH}} + \widehat{x_2}$. Furthermore, the constant 1 replaces occurrences of $\widehat{x_1}$ and $\widehat{x_2}$, because $\widehat{x_1} = \widehat{x_2} = 1$ (x_1 and x_2 are concrete objects). As



$\mathcal{S}_{P_0}$: $\widehat{new_{3U}}+\widehat{new_X}+\widehat{new_{UHF}} = 10$ (#n)

$\mathcal{S}_{P_1}$: $\widehat{new_{3U}}+\widehat{new_X}+\widehat{new_{UHF}} = 10$

$5 \leq \widehat{new_X}+\widehat{new_{UHF}}$ (sp_1)

$\widehat{new_{3U}} \leq \widehat{new_X}+\widehat{new_{UHF}}$ (sp_2)

$\widehat{new_X}+\widehat{new_{UHF}} \leq 2\widehat{new_{3U}}$ (sp_3)

$4 \leq \widehat{new_{3U}} \leq 4, 0 \leq \widehat{new_X} \leq 6, 0 \leq \widehat{new_{UHF}} \leq 6$ (sa)

$\mathcal{S}_{P_2}$: $\widehat{new_{3U}}+\widehat{new_X}+\widehat{new_{UHF}} = 8$

$4 \leq \widehat{new_X}+\widehat{new_{UHF}}$

$\widehat{new_{3U}} \leq \widehat{new_X}+\widehat{new_{UHF}}$

$\widehat{new_X}+\widehat{new_{UHF}} \leq 2\widehat{new_{3U}} + 1$

$3 \leq \widehat{new_{3U}} \leq 3, 0 \leq \widehat{new_X} \leq 5$

$0 \leq \widehat{new_{UHF}} \leq 5, \widehat{x_1} = 1, \widehat{x_2} = 1$

$\mathcal{S}_{P_3}$: $\widehat{new_X}+\widehat{new_{UHF}} = 7$

$4 \leq \widehat{new_X}+\widehat{new_{UHF}}$

$\widehat{new_X}+\widehat{new_{UHF}} \leq 3$

$0 \leq \widehat{new_X} \leq 5, 0 \leq \widehat{new_{UHF}} \leq 5$

$\widehat{x_1} = 1, \widehat{x_2} = 1, \widehat{x_3} = 1$

$7 \leq 3$ is UNSAT → backtrack

Figure 4.1: Scoped partial models and their refinements. Linear equation systems were simplified by carrying out substitutions for conciseness.

there are no new linear equations, $\mathcal{S}_{P_2}$ is otherwise equivalent to $\mathcal{S}_{P_2}$.
In $P_2 \succcurlyeq P_3$, x_3 replaces the multi-object new_{3U} with x_3, i.e., $\langle new_{3U}, x_3 \rangle \in ref_3$, while all other objects of P_3 are mapped to the object with the same name in P_2.

As shown in Proposition 2.26, refinement of regular scoped partial models is transitive, i.e., if $P_1 \succcurlyeq P_2$ and $P_2 \succcurlyeq P_3$ with the refinement relations $ref_1 \subseteq O_{P_1} \times O_{P_2}$ and $ref_2 \subseteq O_{P_2} \times O_{P_3}$, then $P_1 \succcurlyeq P_3$ with the refinement relation $ref_1 \circ ref_2$. Hence, after a chain of refinements $P_0 \succcurlyeq P_1 \succcurlyeq \cdots \succcurlyeq M$, we may obtain a concrete model M. Such a refinement chain will be constructed during model generation.

4.1.3 Predicate evaluation over partial models

As a means of representing metamodel (MM) and well-formedness (WF) constraints, we will use the logic for scoped partial models introduced in Definition 2.29. Note that predicates can be *approximately evaluated* directly on partial models by predicate rewriting [SV17] without materializing all potential concrete models.

In the following, we will consider a theory $\mathcal{T} = \langle d, \mathcal{E} \rangle$ for a signature $\langle \Sigma, \{\varepsilon\}, \alpha \rangle$ encoding MM and WF constraints, where the error predicates $\varphi \in \mathcal{E}$ contain no free variables, i.e., every variable is bound by an existential quantifier $\exists$. Scope constraints associated with a partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ over $\langle \Sigma, \{\varepsilon\}, \alpha \rangle$ will be encoded by the linear inequalities $\mathcal{S}_P$.

Our model generation approach can avoid inconsistent partial solutions during model generation by *approximation of predicates*, so the consistency can be checked before a concrete instance model is obtained.

4.1.3.1 Approximation of logic predicates

In [SNV18; Var+18], we defined over- and under-approximations of predicates over partial models to drive the model generation process along meaningful refinements. If an error predicate φ is surely satisfied in a partial model P ($[\![\varphi]\!]^P_Z = 1$, *under-approximation* of errors), then no concrete instance model M obtained from P by a refinement $P \succcurlyeq M$ can be structurally consistent [Var+18]. Thus, partial model P can be safely dropped from the set of candidate intermediate solutions without discarding any valid instance models, and model generation needs to continue along a different refinement chain.

Theorem 4.1 (Forward refinement of predicates) Let φ be a logic expression without free variables and let P and Q be regular scoped partial models, where $P \succcurlyeq Q$.

- If $[\![\varphi]\!]^P = 1$, then $[\![\varphi]\!]^Q = 1$.
- If $[\![\varphi]\!]^P = 0$, then $[\![\varphi]\!]^Q = 0$.

One can establish a dual *over-approximation* property for the validity of Q, which ensures that no valid model will be marked as invalid (and vice versa):

Theorem 4.2 (Backward refinement of predicates) Let φ be a logic expression without free variables and let P and Q be regular scoped partial models, where $P \succcurlyeq Q$.

- If $[\![\varphi]\!]^Q = 1$, then $[\![\varphi]\!]^P \succcurlyeq 1$.
- If $[\![\varphi]\!]^Q = 0$, then $[\![\varphi]\!]^P \succcurlyeq 0$.

4.1.3.2 Approximation of scope constraints

In scoped partial models, analogous properties hold for the constraints imposed on multi-objects by object scopes $\mathcal{S}_P$. For that purpose, we introduce the notation $\#^{½}_{v}[\![\varphi]\!]^P_Z$ to capture the number

of concrete objects and multi-objects that *may* satisfy φ. Moreover, $\#_v^1[\![\varphi]\!]_Z^P$ represents the number of those that *must* satisfy φ. In a concrete model, these two formulas coincide, and they are equal to the number of concrete objects that satisfy φ.

Definition 4.3 (Number of matching objects) Given a logic formula $\varphi(u_1, \dots, u_k, v)$ and variable binding $Z\colon \{u_1, \dots, u_k\} \to O_P$ (which only excludes v),

$$\#_v^{½}[\![\varphi]\!]_Z^P := \sum\{\widehat{x_i} \mid x_i \in O_P, [\![\varphi]\!]_{Z,v\mapsto x_i}^P \geq ½\}$$

denotes the sum of scope variables $\widehat{x_i}$ associated with objects x_i that may satisfy φ. Analogously,

$$\#_v^1[\![\varphi]\!]_Z^P := \sum\{\widehat{x} \mid x \in O_P, [\![\varphi]\!]_{Z,v\mapsto x}^P = 1\}$$

is the sum of scope variables associated with objects that surely satisfy φ.

For example, if $\{x_1, \dots, x_m\} = \{x_i \in O_P \mid [\![\varphi]\!]_{Z,v\mapsto x_i}^P \geq ½\}$ are the objects that possibly satisfy φ, then the linear inequality $L \leq \widehat{x_1} + \dots + \widehat{x_m} \leq U$ can be written as $L \leq \#_v^{½}[\![\varphi]\!]_Z^P \leq U$.

Note that this notation differs from the count aggregation $\#[\gamma(\bar{v}_1, \dots, \bar{v}_{\alpha(\gamma)})]_Z^P$ notation introduced in Definition 2.27. In particular, the number of matching objects can be calculated for any formula φ, while in count aggregation, γ must be a numerically tracked symbol from Γ. Thus, the introduction of $\#_v^{½}[\![\varphi]\!]_Z^P$ and $\#_v^1[\![\varphi]\!]_Z^P$ notations allow us to define constraints about the number of objects represented in the partial model P even after setting $\Gamma = \{\varepsilon\}$ to reduce the size of $\mathcal{S}_P$.

However, if we *were* to add some symbol γ other than ε to Γ, we could indeed add constraints

$$\#_{v_1}^1[\![\gamma(v_1, v_2, \dots, v_{\alpha(\gamma)})]\!]_Z^P \leq \#[\gamma(*, v_2, \dots, v_{\alpha(\gamma)})]_Z^P \leq \#_{v_1}^{½}[\![\gamma(v_1, v_2, \dots, v_{\alpha(\gamma)})]\!]_Z^P$$

for any variable binding $Z\colon \{v_2, \dots, v_{\alpha(\gamma)}\} \to O_P$ to $\mathcal{S}_P$ without shrinking the set of possible concrete refinements of P, since in concrete models, only surely existing objects can contribute to the match set of γ.

Example 4.3 In P_2 in Fig. 4.1, $\#_s^{½}[\![\exists c : \mathrm{subsys}(s, c)]\!]^{P_2} = \widehat{new_{3U}} + \widehat{x_1}$ is the linear expression for the number of objects that may have an outgoing subsys reference. $\#_s^1[\![\exists c : \mathrm{subsys}(s, c)]\!]^{P_2} = \widehat{x_1}$ is the number of objects that surely have an outgoing subsys reference. (The empty variable binding $Z = \emptyset$ was omitted from the notation for conciseness.)

Now we can evaluate type scope bounds on partial models checking linear inequalities on the objects scopes in a partial model P.

Definition 4.4 (Scope (in)consistency of a partial model) Given a regular scoped partial model P and a set of type scope bounds $\{L_i \leq \mathrm{C}_i \leq U_i \mid i = 1, \dots, m\}$, P is *scope inconsistent* if there exists a type scope bound $L_i \leq \mathrm{C}_i \leq U_i$ such that $\mathcal{S}_P \vDash \#_v^{½}[\![\mathrm{C}_i(v)]\!]^P < L_i$ or $\mathcal{S}_P \vDash \#_v^1[\![\mathrm{C}_i(v)]\!]^P > U_i$. P is *scope consistent* if $\mathcal{S}_P \vDash \#_v^1[\![\mathrm{C}_i(v)]\!]^P \geq L_i$ and $\mathcal{S}_P \vDash \#_v^{½}[\![\mathrm{C}_i(v)]\!]^P \leq U_i$ for all $i = 1, \dots, m$.

Now we can over- and under-approximate scope constraints on partial models and maintain scope consistency during model generation as follows:

Theorem 4.5 (Forward refinement of scopes) Let φ be a logic expression with free variables V, P and Q regular scoped partial models with $P \geqslant_{ref} Q$, $Z\colon V \to O_P$ and $Y\colon V \to O_Q$ variable binding with $\langle Z(v), Y(v)\rangle \in ref$ for all $v \in V$, and $L, U \in \mathbb{Z}$. Then the following

implications hold:

$$\mathcal{S}_P \vDash \#_v^{\frac{1}{2}}[\![\varphi]\!]_Z^P < L \implies \mathcal{S}_Q \vDash \#_v^{\frac{1}{2}}[\![\varphi]\!]_Y^Q < L, \quad \text{(i)}$$
$$\mathcal{S}_P \vDash \#_v^{1}[\![\varphi]\!]_Z^P > U \implies \mathcal{S}_Q \vDash \#_v^{1}[\![\varphi]\!]_Y^Q > U, \quad \text{(ii)}$$

i.e., (i) when objects that *may* satisfy φ violate a lower bound L in P, they also violate it in any refined partial model Q, and (ii) objects that *must* satisfy φ similarly carry forward the violation of the upper bound U.

Therefore, if a partial model P is scope inconsistent, it can be safely dropped from the set of potential intermediate solutions, as all of its refinements remain scope inconsistent.

Dually, if Q is scope consistent $P \succcurlyeq Q$, then P cannot be scope inconsistent. This statement, formalized below, is the over-approximation of validity for scope constraints.

Theorem 4.6 (Backward refinement of scopes) Let φ be a logic expression with free variables V, P and Q regular scoped partial models with $P \succcurlyeq_{ref} Q$, $Z\colon V \to \mathcal{O}_P$ and $Y\colon V \to \mathcal{O}_Q$ variable binding with $\langle Z(v), Y(v)\rangle \in ref$ for all $v \in V$, and $L, U \in \mathbb{Z}$. Then the following implications hold:

$$\mathcal{S}_Q \vDash \#_v^{1}[\![\varphi]\!]_Y^Q \geq L \implies \mathcal{S}_P \nvDash \#_v^{\frac{1}{2}}[\![\varphi]\!]_Z^P < L,$$
$$\mathcal{S}_Q \vDash \#_v^{\frac{1}{2}}[\![\varphi]\!]_Y^Q \leq U \implies \mathcal{S}_P \nvDash \#_v^{1}[\![\varphi]\!]_Z^P > U.$$

The forward and backward refinement properties enable the generation of structurally and scope consistent models along partial model refinements, where WF and scope constraints are approximately checked. Theorems 4.1–4.6, as discussed in Section 4.2.6, ensure the correctness and completeness of the process.

4.2 Model generation with scope reasoning

In this section we exploit numerical information present in object scopes of PMs to efficiently generate large instance models that satisfy type scope bounds, as well as structural and WF constraints. We combine techniques from *advanced graph query processing, SAT solving* and *integer programming* to tackle the scalability problems of existing graph generation approaches.

As the core conceptual contribution of the current paper, we combine the evaluation of relational constraints and numerical reasoning with object scopes by propagating information between 3-valued logic interpretation and objects scopes of the partial model. The intuition behind this idea is that while constraints expressed as object scopes are not as expressive as those captured in relational logic, dedicated numerical solvers allow earlier detection of constraint violations by considering the global effects of all constraints on the number of objects in the generated instance models at the same time. Therefore, the evaluation of the original WF constraints on the partial models and the scope analysis are complementary to each other.

As a summary, object scopes will allow early detection of partial models that cannot be completed to an instance model due to the inappropriate number (e.g., too few or too many) of objects, while WF constraint evaluation will enforce more complex structural validation rules.

4.2.1 Model generation process

We propose a model generation process (shown in Fig. 4.2) based on partial models with object scopes that can exploit the numeric information present in scoped partial models. The generation starts from an initial partial model, which is gradually refined until it obtains a concrete model satisfying the generation objectives defined by the number of required objects and the WF

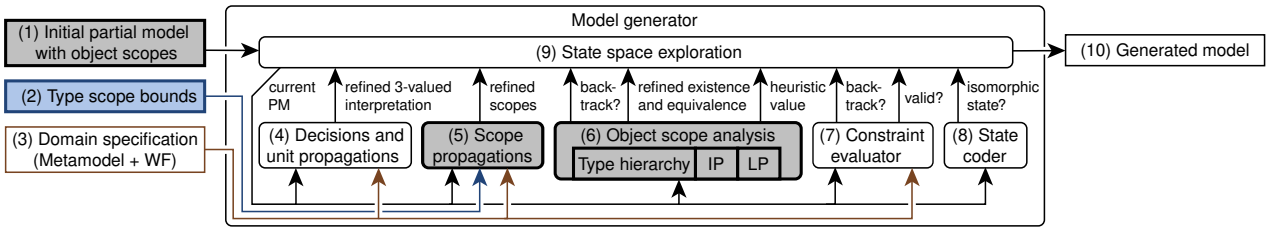


Figure 4.2: Block diagram of the model generator. Blocks interacting with object scopes are shaded for emphasis.

constraints. Thus, the generator explores the state space formed by partial models that are reachable by refinement, which ensures that isomorphic states are explored only once. Our generator has the following components:

- The *initial partial model* **(1)** is the starting point of the generation, which express type scope constraints as object scopes. It is either set to the most general (maximally underspecified) partial model or to a partial snapshot model provided by an engineer which is to be extended by the generator. The other inputs of the generator are the *type scope bounds* **(2)** and the *structural and WF constraints* **(3)** to be satisfied by the generated models.
- *Refinement operators* include decision rules, unit propagation rules and scope propagator rules to obtain new PMs from already discovered ones.
  - *Decisions* add new information to the PM and *unit propagations* enforce the necessary consequences of decisions by evaluating structural and WF constraints using the 3-valued interpretation. For decisions and unit propagations, we reuse the set of operators **(4)** defined by the *GraphSolver* (GS) [SNV18] (which were proved to be sound and complete).
  - *Scope propagators* **(5)** restrict object scopes according to type scope bounds, structural, and WF constraints. This gives an opportunity for numerical reasoning with the new object scope information.
- *Object scope analysis* **(6)** performs numerical reasoning using state-of-the-art integer programming (IP) and linear programming (LP) techniques on object scopes. The results of numerical reasoning are fed back to the partial model and the best-first search strategy.
- 3-valued logic semantics **(7)** are exploited to *over-* and *under-approximate* WF constraint violations. PMs that cannot be repaired by refinement (i.e., which surely violate a constraint) are discarded by backtracking.
  - The constraint evaluation component uses an *efficient, incremental graph query engine* [Ujh+15; Var+16] to ensure the scalability of this step.[1]
  - Unsatisfiable objects scopes, which are caused by type scope or WF constraint violations that cannot be repaired by refinement, are discarded by *backtracking* according to Theorems 4.1–4.6. Detecting these violations as early as possible is crucial for reducing the traversed state space.
- Isomorphic PMs reached by different refinements are detected by *state coding* **(8)** based on graph shapes [Ren06], which ensures that isomorphic states are explored only once.

[1]The incremental graph query engine requires in-place updates to the partial model, which is (technologically) limited to be single-threaded. Nevertheless, it is possible to parallelize incremental query evaluation [Szá+14; Ben+15], as well as to maintain several partial models for parallel state space exploration [Abd+14]. Integrating these improvements to the solver is in the scope of future work.

- *Heuristic* best-first search **(9)** combined with *backjumping* and *random restarts* preferentially investigates PMs that can be quickly refined into valid concrete models. Object scope analysis allows selecting such PMs more accurately than existing approaches that only rely on 3-valued interpretation.
- When a structurally (Definition 2.32) and scope consistent (Definition 4.4) concrete model **(10)** is found, it is recorded as output. The generation is either terminated, or (if additional models are desired) the search is resumed after backtracking (as if the found solution was invalid). For the collected outputs, the solution management features of *GraphSolver*, which can ensure the diversity of the models [Sem+20c], can be leveraged.

Next, we discuss the key novel components of our generator in more details.

4.2.2 Initial scoped PM

Model generation starts from an initial partial model P_{init}, which is a common abstraction of all possible concrete instance models of the metamodel. In P_{init},

- there is an object new_i for each non-abstract class C_i, i.e., $O_{P_{init}} = \{new_i \mid \mathrm{C}_i \in \Sigma\}$;
- for each C_i, new_i is multi, i.e., $\mathcal{I}_{P_{init}}(\varepsilon)(new_i) = ½$ and $\mathcal{I}_{P_{init}}(\sim)(new_i, new_i) = ½$; and
- $\mathcal{I}_{P_{init}}(\mathrm{C}_i)(new_i) = 1$ (new_i is an instance of C_i).

Other class C_j and reference R_k predicates are set to 1 or 0 wherever required by type hierarchy and conformance constraints. Otherwise they are set to ½.

The object scopes $\mathcal{S}_{P_{init}}$ in the initial PM introduce a variable $\widehat{new_i}$ for each class C_i, which allows expressing type scope bounds directly.

If model generation extends an initial partial snapshot, it can also be incorporated into P_{init}. For each given object x_i, $\mathcal{S}_{P_{init}}$ contains the equality $\widehat{x_i} = 1$ to mark x_i as a concrete object with exactly one instance. Interpretation of type and reference predicates between given objects are set in accordance with the initial partial snapshot, while reference predicates leading between *new* objects are ½.

> **Example 4.4** The partial model P_0 in Figure 4.1 is a fragment of the initial partial model P_{init} for generating instances of the *satellite* metamodel in Figure 2.1. The multi-objects new_{3U}, new_{X}, and new_{UHF} correspond to the classes Cube3U, XComm, and UHFComm. In $\mathcal{S}_{P_0} \supseteq \{\widehat{new_{3U}} + \widehat{new_{3U}} + \widehat{new_{3U}} = 10\}$, the linear equation (marked as #n in Figure 4.1) encodes that models with exactly 10 objects shall be generated.

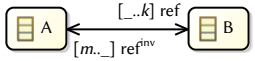
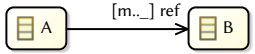
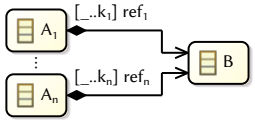
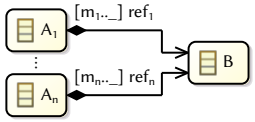
4.2.3 Scope propagation

Scope propagation refines the partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ into a new partial model $P \geqslant Q = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \cup S \rangle$, where S is a set of linear inequalities deduced from P, type scope bounds, as well as structural and WF constraints. Because the inequalities are necessary consequences of the constraints, every consistent concrete model $P \geqslant M$, satisfies them. Therefore each consistent instance model M is also a refinement of Q.

Table 4.1 summarizes the rules used to deduce linear inequalities implied by type scope bounds and structural metamodel constraints from the partial model. In the table, the relation $\leq$ refers to the usual *implication order* $0 \leq ½ \leq 1$ of truth values (and not the *refinement order* $\geqslant$).

> **Example 4.5** Fig. 4.1, the linear equations (sp_1-sp_3) in $\mathcal{S}_{P_1}$ were obtained from P_0 and the type scope bound $5 \leq$ CommSubsys by scope propagation. The *lower type scope bound* CommSubsys implies (sp_1). By applying the *containment hierarchy, lower* and *upper bound* rules to the containment (CON) reference subsys [1..2] according to the multiplicity (MUL) bounds defined in Fig. 2.1, yielding the linear equations (sp_2) and (sp_3)

Table 4.1: Scope propagation rules for type scope bounds and structural constraints

Constraint / class diagram	Description	Linear inequality
$L_i \leq \mathsf{C}_i$	**Lower type scope bound.** There must be at least L_i instances of C_i in the concrete model. Hence, objects that *may* be C_i ($\mathcal{I}_P(\mathsf{C}_i)(x) \geq ½$) must represent at least L_i concrete objects.	$L_i \leq \#_v^{½}[\![\mathsf{C}_i(v)]\!]^P$
$\mathsf{C}_i \leq U_i$	**Upper type scope bound.** There may be at most U_i instances of C_i in the concrete model. Hence, objects that *must* be C_i ($\mathcal{I}_P(\mathsf{C}_i)(x) = 1$) may represent at most U_i concrete objects.	$\#_v^{1}[\![\mathsf{C}_i(v)]\!]^P \leq U_i$
	**Upper bound with inverse lower bound.** Each A may be connected to at most k B instances by the reference ref, and each B must be connected to at least m A instances by the inverse $\mathsf{ref}^{\mathrm{inv}}$. Hence, there can be at most k B instances for each possible m A instances.	$m \cdot \#_u^{1}[\![\mathsf{B}(u)]\!]^P \leq k \cdot \#_v^{½}[\![\mathsf{A}(v)]\!]^P$
	**Lower bound.** Each A instance must be connected to at least m B instances by the reference ref. Hence, for each existing A instance, potential targets of ref must represent at least m concrete objects.	$m \leq \#_v^{½}[\![\mathsf{ref}(u,v)]\!]^P_{u \mapsto x}$ for all $x \in O_P, \mathcal{I}_P(\varepsilon)(x) = \mathcal{I}_P(\mathsf{A})(x) = 1$
	**Containment hierarchy, upper bound.** Let $\mathsf{A}_1, \ldots, \mathsf{A}_n$ be the possible containers of B. For every $i = 1, \ldots, n$, instances of the class A_i can contain at most k_i instances of B (infinite upper bounds $k_i = *$ are replaced by a suitably large finite constant K). Hence, for each possible instance of each A_i, there may be no more than k_i instances of B.	$\#_u^{1}[\![\mathsf{B}(u)]\!]^P \leq \sum_{i=1}^{n} k_i \cdot \#_v^{½}[\![\mathsf{A}_i(v)]\!]^P$
	**Containment hierarchy, lower bound.** Let $\mathsf{A}_1, \ldots, \mathsf{A}_n$ be the possible containers of B. For every $i = 1, \ldots, n$, each instance of the class A_i must contain at least m_i instances of B. Hence, for each instance of each A_i, there must be at least m_i possible instances of B.	$\sum_{i=1}^{n} m_i \cdot \#_u^{1}[\![\mathsf{A}_i(u)]\!]^P \leq \#_v^{½}[\![\mathsf{B}(v)]\!]^P$

Other WF constraints which have numerical consequences can also be translated to object scopes by adding object scope constraints corresponding to lower and upper bounds of the number of objects allowed by the constraint.

Example 4.6 Consider the error predicate $\varphi_8(e) := \big(\exists s\colon \mathsf{subsys}(e,s) \wedge \mathsf{KaComm}(s)\big) \wedge \neg\mathsf{SmallSat}(e) \wedge \neg\mathsf{GroundStation}(e)$. Because subsys is a containment (CON) reference, φ_8 enforces that each KaComm instance be contained in a SmallSat or a GroundStation. Due to the upper multiplicity (MUL) bound of 2, for each SmallSat or GroundStation, there may be no more than 2 KaComm instances. We obtain the following scope propagation rule as the linearization of φ_8:

$$\#_u^{1}[\![\mathsf{KaComm}(u)]\!]^P \leq 2 \cdot \#_v^{½}[\![\mathsf{SmallSat}(v) \vee \mathsf{GroundStation}(v)]\!]^P.$$

In our current implementation, the user needs to manually provide linear inequality versions of well-formedness constraints to exploit them during object scope propagation. A higher level of automatization seems feasible (similarly as in [YBP07]) and is in the scope of future work.

Alternatively, predicate symbols corresponding to the (error) predicates can be added to the set of numerically tracked symbols Γ of the signature $\langle \Sigma, \Gamma, \alpha\rangle$ and to the theory $\mathcal{T} = \langle d, \mathcal{E}\rangle$. This allows leveraging the reasoning techniques from Section 2.3 by either automatically generated or user provided scope propagation rules. We take this approach in Chapter 6 for the WCET analysis of query based monitor programs in embedded systems.

4.2.4 Object scope analysis

Object scope analysis is responsible for numerical reasoning with object scope constraints, which guides model generation and refines the interpretation $\mathcal{I}_P$. The refined relations may allow applying further unit and scope propagation operators, which in turn are opportunities for further scope analysis. The analysis requires efficient maintenance and solution of linear constraints.

Linear constraint maintenance As the size of the partial model P grows, the number of variables and constraints in $\mathcal{S}_P$ may also grow. Two techniques reduce the size of $\mathcal{S}_P$ to improve analysis. Firstly, concrete objects always stand for a single object ($\mathcal{S}_P \vDash \widehat{x} = 1$ if x is concrete). Instead of explicitly storing coefficients of a variable $\widehat{x}$ for each concrete object and linear constraint, occurrences of $\widehat{x}$ are replaced with the constant 1. Thus, the number of variables equals to the number of multi-objects, which usually does not grow during model generation.

Secondly, redundant linear inequalities are eliminated to prevent the number of scope constraints from growing indefinitely, exploiting the following two properties: (i) In our decision and scope propagation rules [SNV18], no new multi-objects are added to the partial model. (ii) In our object scope analysis rules, the coefficients of multi-object variables only depend on the meta-model. This results in many pairs of constraints of the form $L_1 \leq \alpha_1\widehat{x_1} + \cdots + \alpha_n\widehat{x_n} \leq U_1$ and $L_2 \leq \alpha_1\widehat{x_1} + \cdots + \alpha_n\widehat{x_n} \leq U_2$, which can be replaced by $\max\{L_1, L_2\} \leq \alpha_1\widehat{x_1} + \cdots + \alpha_n\widehat{x_n} \leq \min\{U_1, U_2\}$.

Numerical reasoning Numerical reasoning carried out by object scope analysis (i) discovers refinements of the existence ε and equivalence $\sim$ relations implied by the object scopes, (ii) initiates backtracking on unsatisfiable object scopes, and (iii) calculates a heuristic for guiding the search based on the number of objects required to finish the model.

Scopes are analyzed to find lower and upper bounds of object scope variables $\widehat{x}$ associated with each object $x \in O_x$. If the lower bound is positive ($\mathcal{S}_P \vDash \widehat{x} \geq 1$), x represents at least one object and cannot be removed from P. We set $\mathcal{I}_P(\varepsilon)(x) = 1$ to record this fact. If the upper bound is 1 ($\mathcal{S}_P \vDash \widehat{x} \leq 1$), x represents at most one object, implying $\mathcal{I}_P(\sim)(x,x) = 1$. Lastly, an upper bound of 0 ($\mathcal{S}_P \vDash \widehat{x} \leq 0$) means x can be removed from P.

Table 4.2: Scope analysis methods

	Type scope	Structural	Other WF
Type hierarchy	●	○	○
IP solver	●	●	●
LP solver	●	●	●

Legend: ○ = not supported, ● = supported

If a contradiction is detected when obtaining variable bounds, there is no instance model represented by the scoped PM P. The generator discards P and backtracks.

Otherwise, the sum of lower bounds is used as a heuristic in best-first search to approximate the number of decisions still required to obtain a valid instance model. This heuristic prefers the creation of smaller models when possible. However, due to the randomized state-space exploration, it does not guarantee models of minimum size.

Example 4.7 In Fig. 4.1, the linear equations (sa), which represent feasible lower and upper bounds of the object scopes, were obtained by scope analysis of $\mathcal{S}_{P_1} \setminus \{(\text{sa})\}$.
Scope analysis of P_3 detects an inconsistency (highlighted in red in Fig. 4.1) caused by the unsatisfiable object scopes $\mathcal{S}_{P_3}$. No refinement of P_3 is a valid instance model. Therefore P_3 can be safely discarded by backtracking.

4.2.5 Scope analysis methods

We propose three methods for the reasoning, which are shown in Table 4.2. The *Type hierarchy based* analysis can only handle linear equations derived from type scope bounds. It is a quick preliminary step that can detect some contradictions early without invoking an external solver. Analysis with *Integer Programming* (IP) and *Linear Programming* (LP) *solvers* is considerably more precise, and handles any linear equations. However, the invocation of the external solver may be costly, especially in the case of IP, which is NP-complete. In Section 4.3, we compare the effectiveness of these approaches.

Type hierarchy based scope analysis analyses linear equations coming from type scope bounds, which are always of the form $L_i \leq \widehat{x_1} + \cdots + \widehat{x_k} \leq U_i$ (Table 4.1). Exploiting that all variables in $\mathcal{S}_P$ are nonnegative, the inequalities $L_i \leq \widehat{x_1} + \cdots + \widehat{x_k} \leq U_i$ and $L_j \leq \widehat{x_1} + \cdots + \widehat{x_k} + \cdots + \widehat{x_m} \leq U_j$, which are formed when C_i is a subtype of C_j, can be replaced with $L_j \leq \widehat{x_1} + \cdots + \widehat{x_k} \leq \min\{U_i, U_j\}$ and $\max\{L_i, L_j\} \leq \widehat{x_1} + \cdots + \widehat{x_k} + \cdots + \widehat{x_m} \leq U_j$. This process is performed for each pair of compatible inequalities until no more bounds can be tightened. Contradiction is detected when the lower bound of some inequality becomes larger than the upper bound.

Integer programming solvers are used for scope analysis by translating the object scope constraints into an IP problem and repeatedly solving for lower and upper bounds of variables. Formally, for all object $x \in O_P$, the problems

$$x_{min} = \min \widehat{x}, \quad \text{s.t. } \mathcal{S}_P, \quad \forall y \in O_P : \widehat{y} \in \mathbb{N}, \qquad x_{max} = \max \widehat{x}, \quad \text{s.t. } \mathcal{S}_P, \quad \forall y \in O_P : \widehat{y} \in \mathbb{N}$$

are solved and the inequality $x_{min} \leq \widehat{x} \leq x_{max}$ is added to $\mathcal{S}_P$. Results of solver calls are cached to reduce invocations.

Linear programming. By replacing the set of natural numbers $\mathbb{N}$ with the nonnegative reals $\mathbb{R}^{\geq 0}$, the LP *relaxation* of the problem is obtained. In contrast with IP, LP can be solved in

polynomial time. However, the obtained bounds for scope variables may not be as accurate, and opportunities for backtracking or refinement of the ε and ∼ predicates may be detected later. In order to detect these opportunities as early as possible, we rely on the fact that the number of object represented by a multi-object is always an integer. When the relaxation produces an inexact solution with non-integer x_{min} or x_{max}, the solution is rounded to assert the constraint $\lceil x_{min} \rceil \leq \widehat{x} \leq \lfloor x_{max} \rfloor$.

4.2.6 Correctness and completeness

As the main benefit of 3-valued PMs, a *multi-object* may represent multiple separate, unequal *concrete* objects in an instance model. As such, even sets of very large instance models can be abstracted by a small PM, which enables the model generator to use a concise representation of their state as a scoped partial model.

Based on Section 4.1.3, logic constraints can be approximately evaluated over intermediate solutions. Forward- and backward approximation theorems Theorems 4.1 and 4.5 ensure that if a partial model violates a WF or scope constraint, all refinements of that intermediate solution will also surely violate it, thus it can be safely discarded. WF and scope constraints are also directly evaluated on all finished (concrete) models, thus ensuring the *correctness* of the approach, i.e. all generated models are instances of the metamodel, and satisfy all WF and scope constraints.

Additionally, according to Theorems 4.2 and 4.6, if there is a valid concretization of an intermediate model, the partial model will not be discarded due to WF and scope constraint violations. In a bounded scope, all valid partial models will be considered [Var+18]. Therefore, the approach is *complete* within a bounded scope (i.e., when models up to a finite size are sought) and it will explore all valid solutions.

While multiplicity reasoning can greatly increase the performance the model generator, the descriptive power of ordinary PMs is *limited to linear constraints*. This limits the multiplicity reasoning on simple scopes, but ensures that the numerical problems can be efficiently solved in each step of model generation.

4.3 Evaluation

We carried out an experimental evaluation of generating consistent instance models with multiplicity reasoning provided by object scopes to address the following research questions:

RQ1 How effective are the different scope analysis techniques for model generation in terms of execution time?
RQ2 How does our approach scale in execution time on satisfiable problems...
  RQ2.1 ...in the presence of type scope bounds?
  RQ2.2 ...with unbounded type scopes?
RQ3 How does our approach scale in execution time on unsatisfiable problems?
RQ4 To what extent can type scope bounds help in generating models with realistic type distributions?

4.3.1 Domains

Due to the absence of systematically constructed performance benchmarks for the evaluation model generation for DSLs, we evaluated our approach in the context of 3 different domains (and the corresponding DSLs) that include complex structural and WF constraints. The first domain served as the running example in this chapter (Figure 2.1):

- **Sat** is the design space exploration challenge introduced by researchers at NASA Jet Propulsion Lab [Her+17]. As a specific characteristic of this case study, structural constraints specify the number of CommSubsystems and Payloads that can be fitted to a number of Spacecraft, while WF constraints encode additional design rules concerning the satellite communication network.

Two additional case studies exemplify test generation scenarios for industrial modeling tools:

- **Sct:** *Yakindu* is an industrial modeling environment for statecharts [Yak]. This scenario represents generating tests for a concrete modeling tool (Yakindu Statecharts). The WF constraints of the language help avoiding common semantic errors (e.g., the lack of an Entry object signifying state). As a specific characteristic of this case study, most constraints can participate in object scope propagation (after linearization) to determine the possible numbers of objects (e.g., the Entry and its outgoing Transition instance that denotes the initial state).
- **Met:** *Ecore* is the meta-modeling language of EMF [Ste+09]. This scenario represents test generation for a modeling framework (e.g., code generation and persistency). As a specific characteristic, while this case study uses a large number of classes in a complex inheritance hierarchy along with WF constraints, only few of them can be translated into linear inequalities for scope propagation.

In addition to their practical relevance, the these two languages have been used as case studies by multiple model generation papers [SNV18; SVV16; CCR07; Büt+12; Wu16; Alk+20].

4.3.2 Scope analysis methods

Setup *This experiment aims at determining which scope analysis method should test engineers use for scalable model generation.* We generated models containing up to 100 objects in the **Sat** domain.The original *GraphSolver* (**GS/O**) served as a baseline (with type scopes translated to WF constraints). We evaluated the *Type Hierarchy* (**GS/S/TH**) scope analysis method, which relies on no external solver, in addition to the *Integer Programming* (**GS/S/IP**) and *Linear Programming* (**GS/S/LP**) methods. We selected external solvers widely used in industry and research from the *COIN-OR* suite: *COIN-OR Branch and Cut* v2.9.9 for **GS/S/IP** and *COIN-OR Linear Programming Solver* v1.16.10 for **GS/S/LP**[2].

As there were no manually created models available for **Sat**, type scope bounds derived from engineering expectation. When specifying the type scope bounds, we ensured that they were satisfiable, i.e., a valid model exists with the specified number of objects. Unsatisfiable scope bounds are quickly detected by the IP/LP sovers, but cause other approaches to explore a very large number of partial models.

A timeout of 5 minutes was set for each model generation with increasing model sizes. Runs for a given model size were repeated 30 times to account for variance caused by the random exploration and backjumping employed in the generator, as well as the runtime environment.

We also accounted for warm-up effects and memory handling of the Java 11 virtual machine (JVM). Mitigating warm-up effects for benchmarks of small programs (execution time < 2 s) may need a large number of runs [Bar+12]. However, since our macrobenchmarks for the scalability evaluation **GS** and **A** had much longer execution times (up to 300 s), 10 extra runs before the

[2]We also experimented with the *νZ* [BPF15] v4.8.5 optimizing SMT solver. With *Real* variables (used as an LP solver) it produced results similar to CBC and CLP, albeit it performed object scope analysis slightly slower. With *Integer* variables (used as an IP solver), it produced out-of-memory errors. Results were omitted for space considerations.

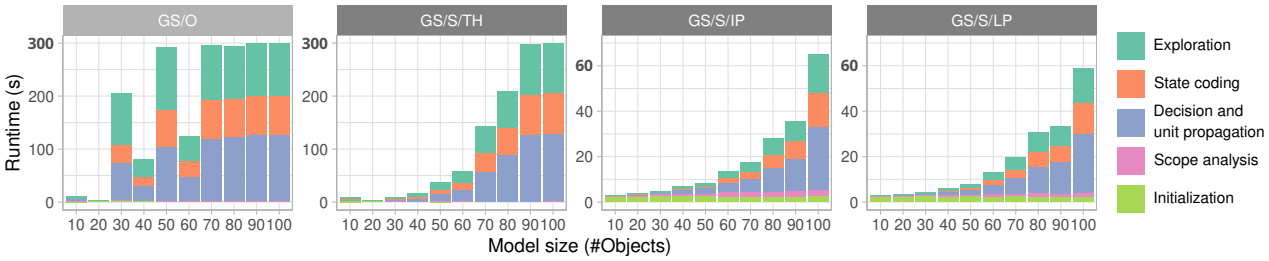


Figure 4.3: Comparison of scope analysis methods on the **Sat** domain

actual measurements and explicit garbage collector calls between runs were sufficient for the stabilization of performance.

All measurements were executed on a high-performance server (2 × AMD EPYC 7551 32-core, 64-thread 2 GHz CPU, 512 GiB RAM) with a hard memory limit of 32 GiB, 16 GiB of which were assigned to the JVM heap to account for additional memory usage by IP and LP solvers. While the model generator is single threaded, parallel garbage collection of the JVM could take advantage of the 8 CPU cores (16 hardware threads) assigned to a measurement.

Results The median running times of the approaches for different model sizes are shown in Fig. 4.3. **GS/O** frequently ran out of the 300 s limit when generating models larger than 30 objects. Timeouts were less frequent in the case of 20 and 40 objects, which caused the median execution time of all runs (including timed out ones) to be discontinuous. This phenomenon can be partially explained by the interaction of type scope bounds and structural multiplicity constraints in **Sat**, which are somewhat easier to satisfy for these model sizes. **GS/S/TH** only reached the time limit for 80 and 90 objects. The median execution times for **GS/S/IP** and **GS/S/LP** were much smaller, not exceeding 65 s to generate models with 100 objects. This makes them the only approaches that were able to produce models of this size.

For all approaches, most of the execution time was spent in the decision and scope propagation, state coding, and exploration steps. The overhead of scope analysis remained below 3.1 s even for the largest generated models, which is negligible compared to other phases of model generation.

For models with 90 elements, **TH** scope analysis reduced number of states (partial models) explored during successful model generation from 41 000 (**GS/O**) to 36 000. **IP** and **LP** further reduced this to around 4000 states, indicating the effectiveness of scope analysis in discarding partial models with no valid concretization. While **IP** and **LP** reduced the state space virtually identically, linear programming (**LP**) was slightly faster: The overall runtime of the external solver was 1.7 s when generating 100-object models compared to the 3.1 s of **IP**.

RA1 Object scope analysis can significantly reduce both the execution time and the state space of model generation. Linear programming can provide the largest reductions with only a minor overhead of external solver calls.

4.3.3 Scalability of model generation

Setup *This experiment aims at determining whether our model generation runs in practical time for test case generation with type scopes.* We generated models with increasing size in the **Sat**, **Sct**, and **Met** domains. For answering **RQ2.1**, we used scope bounds (**+S**) based on engineering expectations for **Sat**, and bounds based on real type distributions for **Sct** and **Met** (see the elaboration of **RQ4**). For answering **RQ2.2**, type scope bounds were omitted (**−S**) by definition.

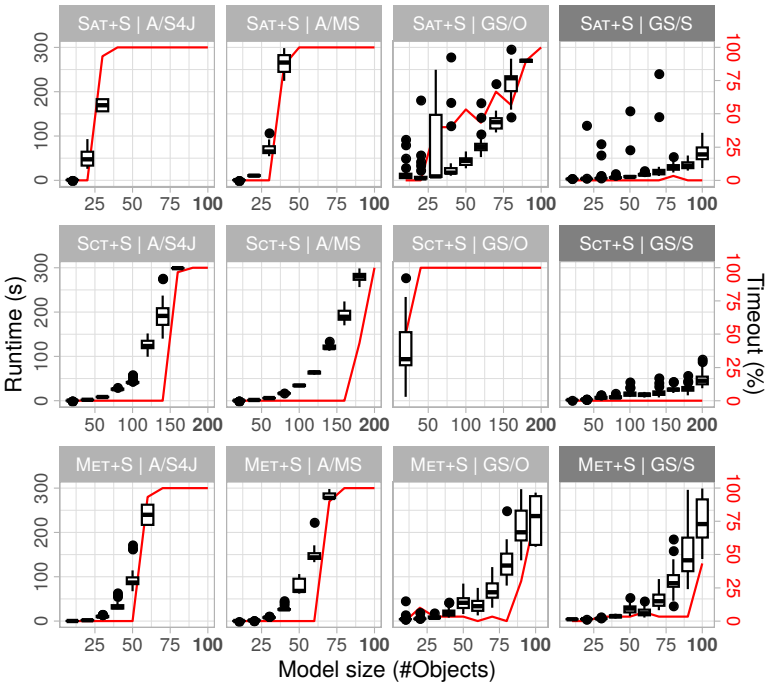


Figure 4.4:
Model generation with type scope bounds

The hardware environment and measurement protocol was identical to that of **RQ1.** We compared the scalability of the following model generators:

- **A**: Alloy Analyzer [Jac02] v4.2 is a popular model finder based of SAT solving (we used the default Sat4J background solver). We translated the model generation problem into an Alloy model by known mappings [TJ07]. We benchmarked both the Sat4J (**A/S4J**) and MiniSat (**A/MS**) background solvers.
- **GS/O:** To generate models with type scope bounds using the original *GraphSolver*, the bounds were translated into WF constraints.
- **GS/S:** Following the findings of **RQ1,** our graph generator used **LP** for object scope analysis.

Results Figs. 4.4 and 4.5 show the execution times of the generators. The random exploration and backjumping heuristics caused large variance in the execution time, including frequent (but nondeterministic) timeouts of **GS/O** for larger models. To enable the in-depth analysis of these effects, the figures show boxplots of *successful* execution times for a given model size. Thus, the medians for **GS/O** are lower than those in Fig. 4.3, which were computed across all (successful or unsuccessful) runs. A red line chart shows the percentage of *unsuccessful* (timed out) executions out of the 30 runs for a given model size.

A encountered out of memory errors as the SAT problems grew too large with the increase of the desired number of objects in the models. In contrast, **GS/O** and **GS/S** were only limited by the execution timeout as limit (and hence the number of partial models the could explore) thanks to the concise representation of the space state by PMs.

It is clear that type scope bounds make the model generation tasks more challenging. For **Sat**, **GS/O** was unable to generate any model of 100 objects. With scope bounds, timeouts started to appear from 30 objects, while without bounds, models with up to 70 elements were generated without timing out. **GS/S** could generate models with 100 objects within 108 sec. However, **GS/S** with scope bounds exhibited some random slowdowns, where generation took an exceedingly long time or reached the time limit. These slowdowns, which were not experienced

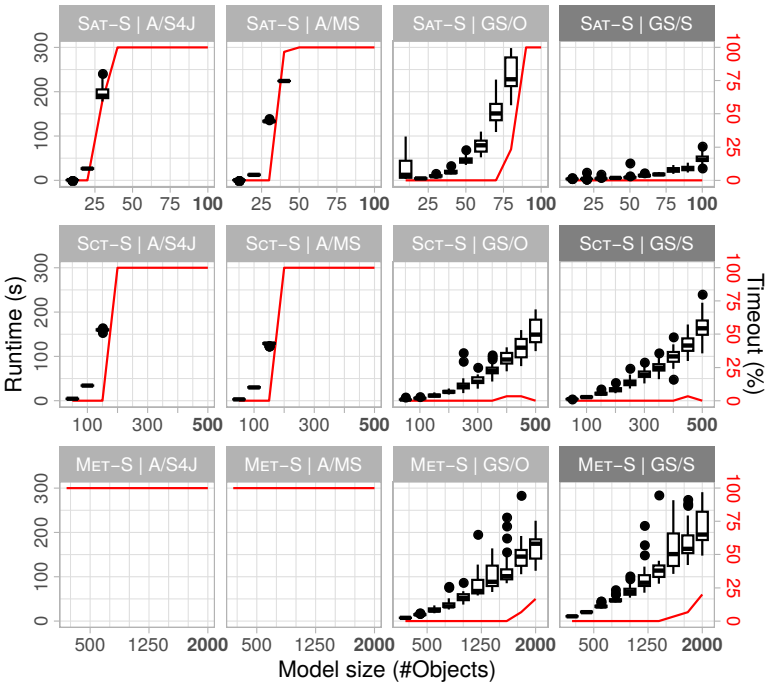


Figure 4.5:
Model generation without type scope bounds

during model generation *without* scope bounds, could possibly be mitigated by refining the backjumping and restarting strategies.

The interaction of type scope bounds with structural and WF constraints in **Sct** made generation of models with realistic type distributions difficult. **GS/O** failed to generate any model of 40 objects or larger, while **GS/S** could generate models with 200 objects within 95 sec.

The removal of type scopes bounds greatly simplifies the task. Both **GS/O** and **GS/S** could produce models with up to 500 objects. In this domain, type scope analysis in **GS/S** yielded a median overhead of 16 s (on a total runtime of 149 s) without reducing the state space (and thus the execution time) of the generator compared to **GS/O** for models with 500 objects.

As **Met** does not contain any structural multiplicity constraints or WF constraints that affect the number of possible objects in the model, **GS/S** could only analyze the type scope bounds themselves. This reduced the median runtime of successful model generation by 18 s and the fraction of timed out runs by 36%. Like **Sct**, the removal of type scope bounds in **Met** made the problem easier. **GS/O** and **GS/S** could generate models with up to 2000 elements with similar performance (with a median scope analysis overhead of 21 s for models of 2000 objects). **A** failed to produce a model even for the smallest size (200 objects) in this scenario.

As a stress test, we also determined the maximum size of a model that **GS/S** can generate within the time limit of 5 minutes. With type scope bounds, these were 155 objects for **Sat**, 436 for **Sct**, and 121 for **Met**. Without satisfying the type scope bounds, much larger models are possible: 157 objects for **Sat**, 649 for **Sct**, and 2631 for **Met**.

RA2.1 For model generation problems with type scope bounds, object scope analysis improves the scalability model generation. The effect is most visible with up to 7-fold reduction in execution times when the type scope bounds interact with structural multiplicity constraints and WF constraints.

RA2.2 In model generation problems without type scope bounds, object scope analysis improves the scalability of model generation in domains with complex structural multiplicity constraints. When no such constraints are present, where is no performance improvement,

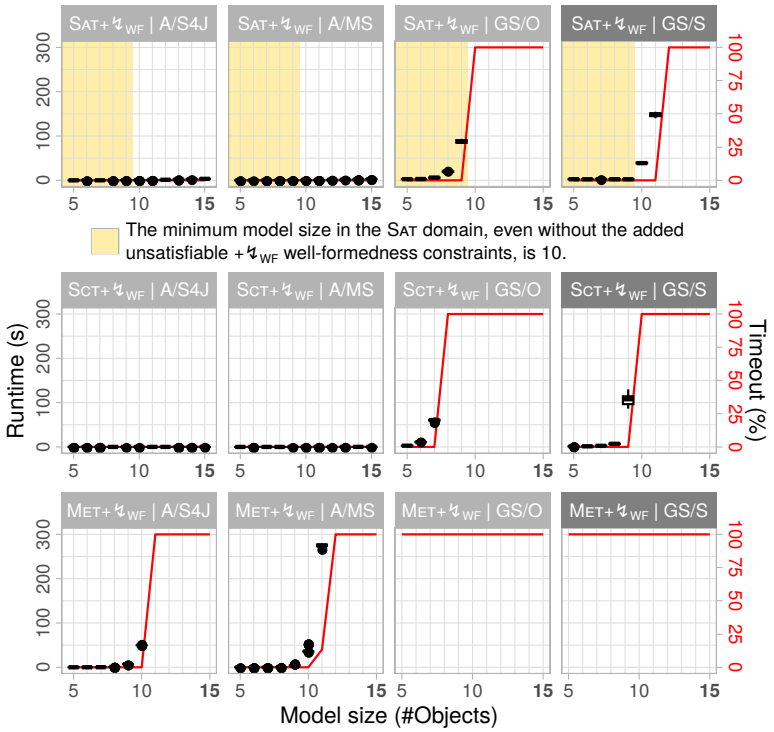


Figure 4.6: Model generation with unsatisfiable well-formedness constraints with problem sizes up to 15 objects

but the overhead incurred by the analysis remains small.

4.3.4 Behavior on unsatisfiable problems

Setup *The purpose of this experiment is to assess the performance degradation occurring in our approach in case of unsatisfiable problems.* Due to the lack of existing benchmark sets of unsatisfiable model generation problems, we introduces two modifications to the domains from **RQ2**. Firstly, we extended each domain with a negated WF constraint, obtaining model generation problems with unsatisfiable WF constraints ($+\lightning_{\mathbf{WF}}$). For example, in **Sat+**$\lightning_{\mathbf{WF}}$, we added the error pattern $\varphi_4'(s) := \neg\varphi_4(s)$, which specifies that no Spacecraft may have a communication path to the GroundStation. Combined with the original φ_4 that forces such communication paths for all Spacecraft, the set of WF constraints $\{\varphi_1, \ldots, \varphi_4, \ldots \varphi_8, \varphi_4'\}$ have no consistent model. Error patterns for **Sct+**$\lightning_{\mathbf{WF}}$ and **Met+**$\lightning_{\mathbf{WF}}$ were defined analogously.

Secondly, we also studied the effect of *unsatisfiable type scope constraints* ($+\lightning_{\mathbf{S}}$), i.e., type scope constraints that correspond to no well-formed models. We changed the required number of objects such that multiplicity (*MUL*) and containment (*CON*) constraints cannot be satisfied due to type scope bounds, e.g., in **Sat+**$\lightning_{\mathbf{S}}$, we required at least 30% of the objects be Satellites but only 25% be CommSubsys instances, despite a Satellite having to contain at least one CommSubsys. We omitted **Met** from this benchmark, as it does not have any *MUL* constraints on *CON* relations.

Although we selected the type fractions to make type scope bounds unsatisfiable, rounding the fractions to whole numbers (quantization errors) of objects may cause the problems to be nevertheless satisfiable for very small instances. Thus, we had to account for these small satisfiable instances in our analysis.

Results Fig. 4.6 shows the execution times of the generators on $+\lightning_{\mathbf{WF}}$ problems up to 15 objects. Even though **GS/S** are primarily aimed at model generation, and thus had to explore a large portion of possible partial models before concluding unsatisfiability, they remained

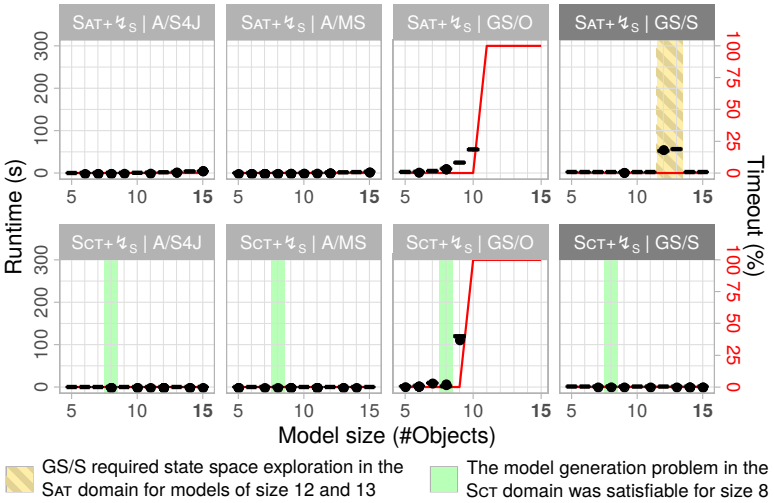


Figure 4.7:
Model generation with unsatisfiable type scope bounds with problem sizes up to 15 objects

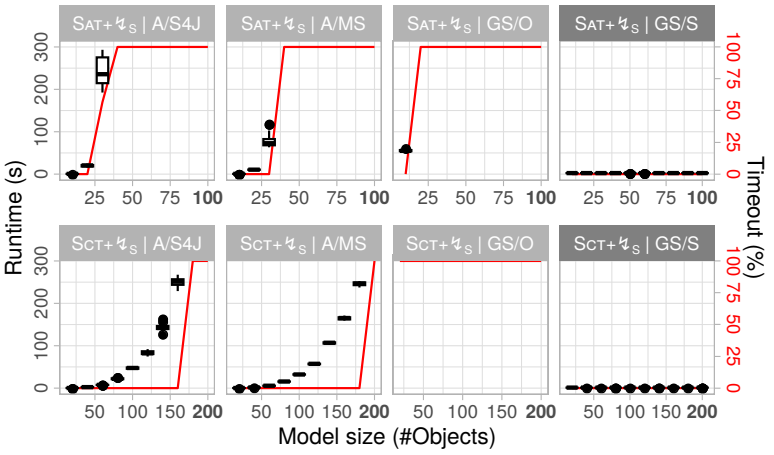


Figure 4.8:
Model generation with unsatisfiable type scope bounds with problem sizes up to 100 and 200 objects

competitive in **Sat+**$\lightning_{\mathbf{WF}}$ and **Sct+**$\lightning_{\mathbf{WF}}$ in problems with up to 9 and 11 objects, respectively. Because **Sat** is unsatisfiable for less than 10 objects (even without +$\lightning_{\mathbf{WF}}$), **GS/S** could terminate without exploring the state space for the first 5 cases. In **Sct+**$\lightning_{\mathbf{WF}}$, although **GS/S** could not outright avoid state space exploration, it explored 16 times less states than **GS/O** thanks to scope analysis. **A**, which is much better suited for problems with unsatisfiable constraints, managed to prove unsatisfiability within 4 s for all model sizes in **Sat+**$\lightning_{\mathbf{WF}}$ and **Sct+**$\lightning_{\mathbf{WF}}$.

Met+$\lightning_{\mathbf{WF}}$ was more difficult for all approaches: while **A** could prove unsatisfiability with up to 11 objects (running out of memory at 12 objects), **GS/O** and **GS/S** did not terminate within the time limit even for 5 objects, exploring 19 000 states before timeout.

Figs. 4.7 and 4.8 show the execution times of the approaches on +$\lightning_{\mathbf{S}}$ problems. For small +$\lightning_{\mathbf{S}}$ problems with up to $n = 15$ objects, both **A** and **GS/S** terminated successfully within 5 s except in the cases of $n = 12$ and 13 in **Sat+**$\lightning_{\mathbf{S}}$ for **GS/S**. Due to the rounding of the type scope fractions into whole number bounds, these cases did not result in immediate unsatisfiable systems of linear equations. Therefore, **GS/S** had to explore 4982 and 5413 states, respectively, before concluding unsatisfiability. There also was a rounding effect that made **Sct+**$\lightning_{\mathbf{S}}$ satisfiable for $n = 8$. The model was found by **GS/S** after exploring 16 states. **GS/O** ran out of time after 10 objects, because it had to exhaustively enumerate partial models.

For larger problems with up to 100 objects for **Sat+**$\lightning_{\mathbf{WF}}$ and up to 200 objects for **Sct+**$\lightning_{\mathbf{WF}}$ in Fig. 4.8, the execution time of **GS/S** remained constant below 5 s. In contrast, the execution times of **A** increased cubically with n.

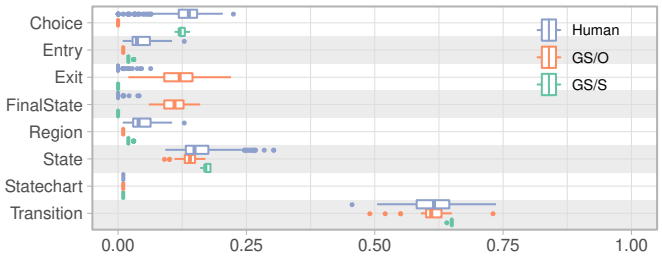


Figure 4.9: Fractions of objects of given types in **Sct**

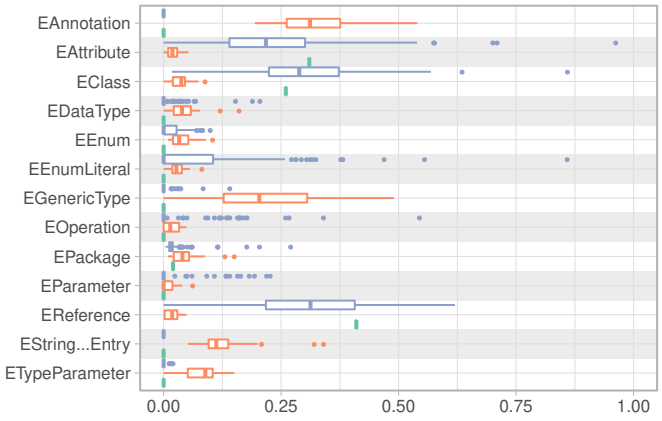


Figure 4.10: Fractions of objects of given types in **Met**

RA3 For model generation problems with unsatisfiable well-formedness constraints, object scope analysis can improve the scalability of search space exploration for model generators. However, SAT solvers are better suited to tackle such problems. For model generation problems with unsatisfiable type scope bounds, object scope analysis can eliminate the need for exploring the state space, and the time taken for proving unsatisfiability is independent from the (potential) size of the state space.

4.3.5 Type distributions of models

Setup *This experiment aims at comparing test generation approaches (without and with type scope bounds) where the test engineer desires to avoid unrealistic test models.* To address **RQ4**, we calculated the distribution of the fractions of objects of given types (i.e. the number of objects of a type in the model divided by the model size) of human (manually created) models, and compared them to the type distributions of automatically generated models. The use of type distributions as means of realistic nature of models was motivated by [SSB20; SSB17]. As human models were only available for **Sct** and **Met**, we excluded **Sat** from this comparison.

- **Human:** We gathered 304 **Sct** models with sizes between 90 and 110 objects that were submitted as part of a homework assignment [SysMod], where students solved similar (but not identical) modeling challenges. For **Met**, we collected 153 manually created class diagrams (those generated from XML schema were excluded) with sizes between 50 and 200 from open source projects hosted on GitHub[3].
- **GS/O:** For both domains, we generated 30 models of 100 objects (without any type scope bounds) with the original *GraphSolver* [SNV18] tool.
- **GS/S:** We generated models with realistic type distributions with our graph solver enhanced with type scope support. To determine the lower and upper bounds for each type, we

[3]We queried the GitHub (`https://github.com`) API for the 1000 most recent Ecore models as of July 31st, 2019 and filtered for model size and the lack of XML schema.

computed the lower $Q_i^{(1)}$ and upper $Q_i^{(3)}$ quartiles of the object fractions in the **Human** models for each non-abstract class C_i. Then for the generation of models with $n = 100$ objects, we added the type scope bounds $\lfloor Q_i^{(1)} n \rfloor \leq C_i \leq \lceil Q_i^{(3)} n \rceil$.

Results The distribution of type fractions is shown in Figs. 4.9 and 4.10. In **Sct**, **GS/O** generated a large number of Exit and FinalState objects compared to the **Human** while it almost entirely omitted Choice, Entry, and Region. The average discrepancy between the type distribution of **Human** models in **GS/O** models is 25 objects per model (25%) that would need a different type to match the **Human** distribution.

In **Met**, **GS/O** overused the EAnnotation, EGenericType, EStringToStringMapEntry, and ETypeParameter classes at the expense of EAttribute, EClass, and EReference objects. The average discrepancy was 83 objects per model (with 100 objects). Models generated by **GS/S** had identical type distributions, which for EAttribute and EClass coincided with the upper type scope bound.

Therefore, **GS/O** failed to generate models matching the type distributions of **Human** models. In contrast, **GS/S** can be parameterized to satisfy type distribution requirements, e.g., *probabilistic types* and *histograms* [SSB17]. Furthermore, to capture more complex correlation between distributions of different types, users can inspect generated models and easily (albeit manually) refine type scope bounds to exclude results that are not realistic, using an iterative process based on previously generated undesired models.

All models generated by **GS/O** and **GS/S** were connected (i.e. no islands or forest of nodes) and they were structurally different from each other, which is guaranteed by the underlying state space exploration strategy [SNV18].

RA4 Models generated without type scopes bounds greatly differ in type distribution compared to human (manually created) models. The use of type scope bounds allows generating nontrivial, connected graph models with designated type distributions.

While type distributions were found to be a useful metric to characterize the realistic nature of models [SSB20; Var+18], further investigations are necessitated along various metrics to claim that the auto-generated models are truly realistic.

We also confirmed that the internal diversity [Sem+20c] of the synthesized models is not impacted negatively by the proposed approach. The relevance of this metric in mutation testing is shown in [Sem+20c].

4.3.6 Limitations and threats to validity

Limitations Our approach shares some of its strengths and limitations with *GraphSolver* [SNV18]. Namely, it operates over connected sparse graphs with edges and relations, i.e., without edge identities or parallel edges (which is suitable to represent standard EMF models).

The expressive power of the graph predicates capturing WF constraints is equivalent to first-order logic with transitive closure over binary predicates. While type scope bounds and object scopes do not bring additional expressiveness, so they can be transformed back into WF constraints, they considerably improve scalability in various domains. Object scopes consisting of linear inequalities can exactly encode type scopes bounds (including the bound on the overall model size), and they can also encode weakened versions of structural *MUL* and *CON* constraints (including the XOR between different containment relations of objects), guiding state space exploration in challenges that often arise from class diagrams. However, for model generation tasks without such constraints, it may not be possible to (even manually) encode useful linear inequalities, and the introduced object scope analysis may pose a slight overhead over the baseline generator.

The sound and complete set of decision rules allow formal reasoning within the bounded scope defined by type scope bounds. However, unlike many SAT and SMT solvers, there is currently no support for an unsatisfiable core (a minimal contradictory set of formulas) that would highlight the contradiction between WF constraints or type scopes.

The work presented in this paper only considers classes and references, but not attributes. While the three-valued logic framework can support basic attributes, placing and maintaining scope bounds for attribute values would require additional abstractions, such as [FFJ12].

In unsatisfiable problems, proving unsatisfiability with a model generator may require exponential time to exhaustively traverse the search space if the search cannot be aborted early with scope analysis. Thus such problems may be more amenable to SAT solving instead.

The generation of models with realistic type distributions assumes the availability of real models to determine type histograms. For ensuring realistic properties other than type distributions, additional heuristics may be needed.

Internal validity Our scalability experiments incorporated a warm-up phase prior to actual measurements and garbage collector calls between actual measurements to reduce variance of execution times due to the JVM (but not due to the inherent behavior of the model generators). To further mitigate disturbances from the environment, each measurement was pinned to a single memory controller and the associated CPU cores on our server. We used default configurations for the external **IP** and **LP** solvers, as well as **A**. Domain-specific fine-tunings may reduce the execution times of these programs, but in most cases they were already negligible.

As noted in Section 2.1.3, **A** only supports limited type scopes. The **+S** problems cannot be formalized in **A** without the use of the # operator, even if lower bound constraints are omitted. However, as **A** performance was similar on both **+S** and **−S** problems, our encoding of the bounds likely did not introduce scalability bottlenecks.

For determining realistic type distributions of the industrial modeling languages, we considered manually constructed and automatically generated models of similar size to minimize discrepancies caused by different scales. For **Sat**, the distribution were prescribed manually. The behavior of the model generators did not change drastically upon changing the prescribed distribution, as long as the arising type scope bounds remained satisfiable.

External validity Our measurements cover 3 domains (1 from a design space exploration challenge published by NASA researchers, 2 from industrial modeling languages) both with and without *realistic type scope bounds.* All domains had *complex structural* and *WF constraints* that interacted in various ways with the type scope bounds. Consequently, our experimental scalability results of our graph generator are likely generalizable to other domains of similar size.

As the performance of object scope analysis based on IP and LP depends on the selected external solver, we integrated the *v*Z optimizing SMT solver in addition to the well-known solvers from the COIN-OR project. In case of LP problems, performance was comparable to CLP, while for IP problems, CBC proved to be significantly better. Therefore, the reported scalability of IP and LP object scope propagation likely matches what is achievable with state-of-the-art external solvers.

4.4 Related work

Logic solver approaches Several approaches map a model generation problem into a logic problem, which is solved by underlying SAT/SMT-solvers. Complete frameworks with standalone specification languages include Formula [JLB11] (using the Z3 SMT-solver [MB08]), Alloy [Jac02] (using SAT-solvers like Sat4j [LP10]) and Clafer [Bak+13] (using reasoners like Alloy).

There are several approaches aiming to validate standardized engineering models enriched with OCL constraints [GBR05] by relying upon different back-end logic-based approaches such as constraint logic programming [CCR07; CCR08; BC12], SAT-based model finders (like Alloy) [SAB09; Ana+10; Büt+12; KHG11; Soe+10; Sem+17; SVV16; Men+17], CSP solvers [Gon+12], first-order logic [BKS02], constructive query containment [Que+12] or higher-order logic [GRR09]. Partial snapshots and WF constraints can be uniformly represented as constraints [Sem+17]. Growing models are supported in [JS07; SVV16] for a limited set of constraints.

Scalability of all these approaches are limited to small models / counter-examples. Furthermore, these approaches are either a priori bounded (where the search space needs to be restricted explicitly) or they have decidability issues. As our approach is independent of the actual mapping of constraints to logic formulae, it could potentially be integrated with most of the above techniques by complementing or replacing the back-end solvers.

Uncertain models Partial models are similar to uncertain models, which offer a rich specification language [FSC12; SC15] amenable to analysis. They are a more user-friendly language compared to 3-valued interpretations, but without handling additional WF-constraints. Potential concrete models compliant with an uncertain model can be synthesized by the Alloy Analyzer [SFC12], or refined by graph transformation rules [Sal+15]. Each concrete model is derived in a single step, thus their approach is not iterative like ours. Scalability analysis is omitted from these papers, but refinement of uncertain models is always decidable.

Approaches like [Fam+13] analyze possible matches and executions of model transformation rules on partial models by using a SAT solver (MathSAT4) or by automated graph approximation (referred to as "lifting"), or by graph query engines with [SV17]. As a key difference, our approach carries out model refinement while simultaneously evaluating graph query evaluation.

Iterative approaches Iterative approaches generate models by multiple solver calls. In [SVV16] models are generated in by calling Alloy in multiple steps, where each step extends the instance model by a few elements. This approach scaled up to 50 object in 45 s for generating valid Yakindu Statecharts. An iterative approach is proposed *specifically for allocation problems* in [KJS10] based on Formula. An iterative, counter-example guided synthesis is proposed for higher-order logic formulae in [Mil+15], but the size of models is fixed and smaller than 50 objects.

Symbolic model generation techniques Certain techniques use abstract (or symbolic) graphs for analysis purposes. A tableau-based reasoning method is proposed for graph properties [SLO17; Pen08; ADW16], which automatically refines solutions based on well-formedness constraints, and handles the state space in the form of a resolution tree. As a key difference, our approach refines possible solutions in the form of partial models, while [SLO17; Pen08] resolves the graph constraints to a concrete solution. Therefore our approach is able to exploit efficient graph query engines to evaluate partial solutions, while those techniques are demonstrated on small (< 10 objects) graphs or with no scalability evaluation.

Different approaches use abstract interpretation [RD06], or predicate abstraction [RSW04; FFJ12; Gop+04] for partial modeling. In those approaches, concretization is used to materialize (typically small) counter-examples for designated safety properties in a graph transformation system. However, their focus is to support model checking of abstract graph transformation systems, which can evaluate complex trajectories, but do not scale in the size of the models.

Additionally, counter abstractions by *Petri graphs* were used in the verification of graph transformation systems [Var+06] and as heuristic functions for rule-based design-space exploration [HHV15]. The Augur framework [BCK01; KK08; KK06] uses similar counter abstraction on graph properties for in graph transformation systems, which can be analyzed as a transition system. As a key difference, a graph transformation rule can both increase or decrease amounts

in abstract graphs, while in our approach the constraints are respecting the refinement relation, thus we can utilize IP and LP solvers instead of model checkers.

The *Incidence Matrix with Multiplicity* (IMM) data structure was proposed by Levendovszky et al. [LLC05] for the automatic instantiation of UML class diagrams according to multiplicity constraints [Lev06].

Smart bound selection for the number of objects was used in the satisfiability checking of OCL formulas in [CGC19].

Numerical abstractions Verification of programs containing numerical (integer or real) variables by abstract interpretation relies on numerical abstract domains [Min04; SPV18], including polyhedra defined by systems of linear equations [CH78; BHZ08], as a key component to over- and under-approximate the sets of possible program states. Numerical abstract domains are combined with graph abstractions in two main ways to verify heap and pointer based programs.

Firstly, numerical abstract domains may summarize object attributes (field) in value analysis of heap programs [Mag+07; MRS10; FFJ12]. Summarized dimensions [Gop+04] were introduced to succinctly represent attributes of a potentially unbounded set of objects via multi-objects. This approach can be seen as complementary to ours, as it enables attribute handling in three-valued partial models, and allows checking for refinements by abstract subsumption [APV09].

Secondly, numerical abstract domains can aid reasoning about the number of objects in a graph (usually a program heap) by structural counter abstraction [Ban+13]. This approach is closely related to ours, but its use is limited to program verification. In contrast, we explicitly incorporate uncertain types and references by three-valued partial modeling to enable model generation.

Model-based quantifier instantiation approach [Rey+13] in SMT solvers for finite model finding also relies on counter abstractions. It can be seen as a dual to scoped model generation: it aims to merge terms to satisfy finiteness constrains instead of splitting multi-objects to add new objects.

4.5 Conclusions

In this chapter, we proposed a new 3-valued scoped partial modeling and reasoning technique which allows to explicitly represent multiplicity constraints on the size of partial models with a linear inequality system. Those constraints cover the requested size of the completed model for each class (type) and the additional constraints imposed by the metamodel (e.g., reference multiplicities) and well-formedness constraints. The resultant multiplicity constraints can be efficiently solved by an underlying IP or LP solver to get a more precise view on the number of objects in a potential concretization of the partial model, or to detect infeasible numerical requirements on it. Based on the advanced numerical reasoning on partial models, we extend the graph solver algorithm of [SNV18; Sem+19] with scoped partial models and numerical reasoning using IP or LP solvers, which greatly improves the performance of the solver (and outperforms related solvers like [TJ07]). Additionally, the proposed technique enables the efficient use of type scopes, which allows the generation of more realistic or useful models.

Chapter 5

Creating phased-mission models by view transformations

Model-driven engineering in manufacturing system design [Lie+14] is typically used to capture system architectures, behavior and possible reconfigurations. However, the extra-functional requirements, such as dependability (reliability, availability), as well as performance concerns (including timing and resource utilization) remain challenging to address [Vog+15], especially in the context of changing and evolving architectures [Fel03; RSV17].

The viewpoints of dependability and performability are concerned with the probabilistic behaviors of systems, and often need formal stochastic models for rigorous analysis. Automated derivation of stochastic models from architecture models by model transformations [BMM99; Koz10; BMP12] can integrate such analyses into multi-paradigm workflows.

In this work, we propose an approach to capture dynamic reconfigurations, fault handling and parameter changes on the level of the architecture model. To do this, we define a mission automaton formalism by extending Graph Transformation Abstract State Machines (GT+ASM) [VB07] with stochastic and timing properties. GT+ASM leverages graph pattern matching, which is the technique applied on the architecture model for the description of reconfigurations and fault handling. The changes described by the mission automaton result in multiple, non-overlapping phases of operation, that can be captured by stochastic modeling of phased-mission systems (PMS) [MB99]. We also define the (graph transformation based) construction of PMS analysis model from the architecture model and its evolution described by the mission automaton.

The PMS construction workflow is illustrated in Fig. 5.1. Domain modeling experts can construct or reuse the architecture modeling language, the graph patterns used for specifying changes, as well as the architecture models themselves. The modeling language is extended with run-time attributes to describe stochastic behaviors, such as failures and performance related events. Stochastic modeling experts specify the analysis model transformation in terms of Generalized Stochastic Petri Nets (GPSN) [MCB84], which are especially amenable for automated, modular construction [MB15] and allow leveraging existing transformations, e.g. [BMM99]. After the domain modeling expert specifies the mission automaton, its state space is automatically explored and GSPN models of the phases of operation are derived for PMS analysis.

We implemented a PMS analysis model construction tool based on change-driven target incremental execution [CH06; Ber+15] of model transformations. Therefore, PMSs even with a large number of phases can be created efficiently. We demonstrate our approach with a running example and case study of a flexible manufacturing system.

The contents of this chapter are based on the conference paper [c12] and the report [r19].

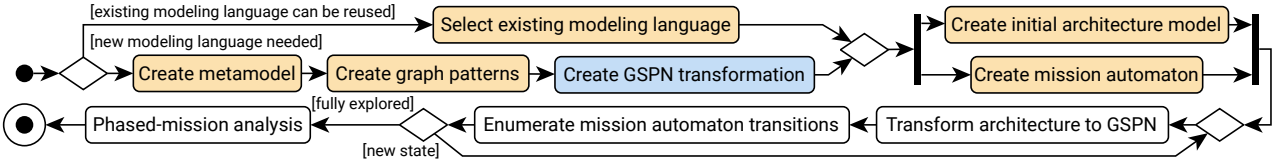


Figure 5.1: Dependability analysis using mission automata, with modeling expert (orange), stochastic analysis expert (blue) and automated (white) activities

5.1 Preliminaries

5.1.1 Architecture models and metamodeling

In model-driven engineering graphs are used as formal descriptions of models, including UML, SysML and AADL artifacts [BG01], such as production system architectures.

Metamodels explicitly describe the abstract syntax of modeling languages, including the classes, references and attributes that comprise the language. An architecture model is an *instance model* of the architecture modeling language.

We will use 4-valued partial models (Section 2.2) with attributes and compositional view transformations (Chapter 3) to reason about architecture and dependability models. However, we will restrict our attention to *concrete* models in the systems engineering process. As such models only contain 1 and 0 logic values and concrete numbers as attribute values, the presentation can be considerably simplified compared to Section 2.2.

In the implementation part of our work, we used the EMF [Ste+09] along with the view transformation engine described in Chapter 3.

> **Example 5.1** The class diagram in Fig. 5.2 shows a metamodel for flexible manufacturing system (FMS) architectures based on the FMS performance model by Ciardo and Trivedi [CT93], which will serve as a running example throughout this chapter.
> The FMS contains Machines, which have Capabilities to perform Tasks. A Task takes inputs from a Stockpile to produce an output. Machines have reliability attributes mttf and repairTime, while Capabilities contain the executionTime of the Task. Tasks may be marked with HIGH importance, and the initialSize of a Stockpile can be specified.
> Run-time adaptation and reconfiguration may happen in response to changes in the failed states of machines and the size of Stockpiles. Machines may be added or removed, as well as Tasks may be assigned to Machines.
> An example instance model, which is represented as an object graph, is shown in Fig. 5.3.

As in Definition 2.2, we will use graph predicates to encode queries about models. As part of our simplified presetation, we will assume that the evaluation of predicates only returns 1 and 0 values, i.e., they cause no invalid numeric operations (such as division by zero) that would give rise to an $\lightning$ result. Therefore, we will consider any architecture models or reconfigurations that result in invalid numeric operation a modeling error. Since the techniques presented in the remainder of this chapter rely on generating the set of all specified reconfigurations before performability analysis, such modeling errors can be detected before invoking costly external analysis tools or solvers.

Here, we introduce some notions for the traceability of view transformations over concrete models. *Parameter variables,* which are the free variables of the predicate, match distinguished objects inside a pattern. Let $O(M)$ denote the set of objects in an instance model M. The tuple $\langle o_1, \ldots, o_k \rangle \in O(M)^k$ is a *match argument tuple* of the k-argument graph pattern $\phi(x_i, \ldots, x_k)$, i.e., $M \vDash \phi(o_1, \ldots, o_k)$, if ϕ matches M after binding each x_i inside ϕ to the corresponding o_i.

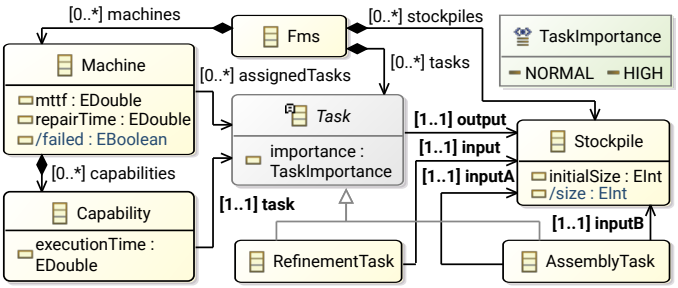


Figure 5.2: Class diagram for the FMS architecture metamodel

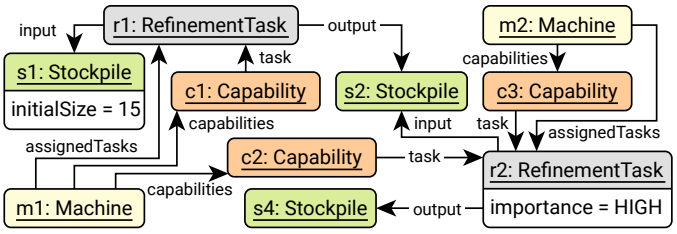


Figure 5.3: Example FMS architecture instance model

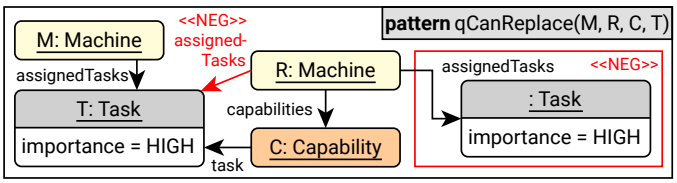


Figure 5.4: Example graph pattern

Example 5.2 Fig. 5.4 shows a graphical representation of the graph pattern qCanReplace. A match argument tuple $\langle M, R, C, T\rangle$ expresses that the Machine R can replace M in performing the high-importance Task T, because it is currently not assigned to T or any other high-importance Task, but it has Capability C to perform T.
In Fig. 5.3, we have $\vDash$ qCanReplace($m2$, $m1$, $c3$, $a1$).

5.1.2 Generalized Stochastic Petri Nets

Generalized Stochastic Petri Nets (GSPN) are a commonly used formalisms for the dependability evaluation of asynchronous systems. Formally, they are directed bipartite graphs with a set of places P and transitions T [MCB84].

A *marking* $m\colon P \to \mathbb{N}$ assigns *token* counts to the places. Starting from the initial marking m_0, if enough tokens are available at its input arcs, and no transition with higher priority is fireable, a transition may be *fired* to remove tokens from its input places and put tokens to its output places.

A continuous-time Markov chain (CTMC) represents the stochastic behaviors of the GSPN. *Timed* transitions are fired when an exponentially distributed delay with rate parameter $\lambda(t)$ has elapsed, while *immediate* transitions are fired immediately according to their priority $\Pi(t)$ and probability weight $\lambda(t)$ when they become enabled.

5.1.3 Phased-mission systems

Phased-mission systems are characterized by consecutive phases of operation caused by changes in system configuration or environment [SRA92]. Modeling and analysis of PMSs are made more complex than single phased systems by the history of the system, such as degradation of the components, affecting subsequently occurring phases.

In state-based stochastic PMS models, each phase is described by a *lower level model*. The *upper level model* determines the length of each phase and the possible phase transitions. In

order to propagate the history of the system, phase transitions map states of the lower level model associated with the source phase to the target phase.

5.2 Automated analysis model construction

5.2.1 Dependability analysis models for static architectures

First we consider the construction of analysis models for failure processes of unchanging architectures, which will be called the *static* analysis model transformation. Creation of the *dynamic* analysis models that incorporate reconfigurations of the architecture models calls the static transformation as a subroutine.

In case of architecture models, the typical approach is a modular transformation where patterns from the architecture model are systematically mapped (based on the types of elements) to interconnected model fragments in the analysis model. This process is facilitated by modular and compositional extensions to Petri nets [MB15]. For example, the modular Petri nets formalism [KP09] allows the assembly of large models by instantiating and connecting net fragments.

Model transformations tools, such as [Jou+08; Ber+15; QVT], construct target (right-side) models according to matches of precondition patterns in the source (left-side) models. The left side of a single transformation rule is a *precondition* graph pattern ϕ. The right side is a template for target model objects, in our case, a Petri net fragment [KP09]. For each match argument tuple $\langle x_1, \ldots, x_k \rangle$ of ϕ its right side is instantiated by adding a copy of it to the target model.

Traceability information relates the source and the target instance models. The *horizontal trace* hyperedges connect the objects of the match argument tuples on the left to the target model objects on the right. By adding *local names* to objects within Petri net fragment templates, target model elements can be referenced through horizontal traces: $\phi(x_1, \ldots, x_k).p_j$ unambiguously identifies the place with local name p_j in the fragment instance associated with the rule ϕ and match arguments $\langle x_1, \ldots, x_k \rangle$.

5.2.2 Handling architecture model changes

No we look at architecture model changes that produce a new model M_2 from the original architecture M_1.

5.2.2.1 Vertical trace

The *vertical trace* relation $\sim : O(M_1) \times O(M_2)$ between the objects of M_1 and M_2 describes the objects preserved, created and removed. If $x \sim y$ holds, the modification preserved x as y in M_2. Created objects y have no corresponding x, while removed objects x have no y.

Given horizontal traceability information for M_1, M_2, we can derive vertical traces for their analysis models PN_1, PN_2. Let $\phi(x_1, \ldots, x_k).p \sim_{PN} \phi(y_1, \ldots, y_k).p$ when $x_i \sim_M y_i$ $(i = 1, \ldots, k)$, where $\sim_M$ and $\sim_{PN}$ are the vertical traces of the architecture and analysis models, respectively. That is, a Petri net node is preserved if the match arguments of the responsible rule activation were preserved by the architectural change.

Horizontal and vertical traces are illustrated in Fig. 5.5. From the starting architecture M_0 and its analysis net PN_0 several other models are created by architectural changes. A vertical trace relation accompanies each modification. Each architecture M_i is connected to its Petri net PN_i by the horizontal trace hyperedges. The vertical trace between each adjacent PN_i, PN_j can be derived from the vertical trace between M_i, M_j and the corresponding horizontal traces.

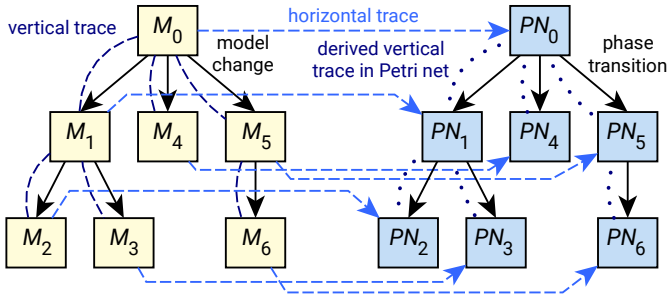


Figure 5.5: Traceability relationships of the architecture and analysis models

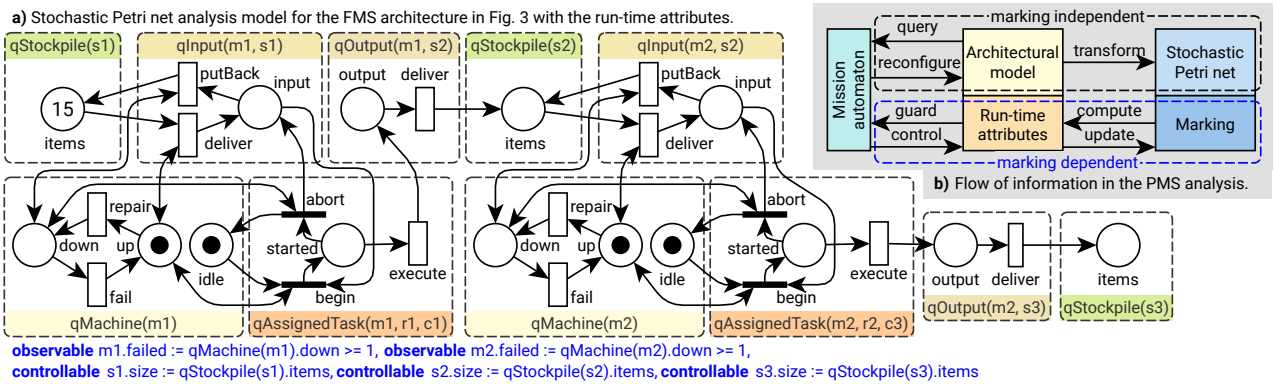


Figure 5.6: (a) Example Petri net analysis model for our running example and (b) flow of information in the PMS analysis.

5.2.2.2 Maintenance of markings

Along with the stochastic Petri net describing the failure processes of the production system, we also maintain the original architecture model so that it can be used to specify reconfigurations. Hence the states of the system are represented by pairs $\langle M_i, m_j \rangle$ of architectures M_i and markings m_j of the corresponding PN_i.

When the architecture changes add or remove objects, the restriction of $\sim_M$ to $O(M) \cap O(M')$, and hence the restriction of $\sim_{PN}$ to $PN \cap PN'$ is invertible. Upon such architecture change $M \mapsto M'$, the state change $\langle M, m \rangle \mapsto \langle M', m' \rangle$ updates the marking as follows: Tokens are *preserved* along the derived vertical traceability, i.e. $m'(p') = m(p)$, where $p \sim_{PN} p'$. *New places* q of PN' with no corresponding place in PN are set to their initial marking $m(q) = m_0(q)$ instead. Tokens on *deleted places* are lost.

While architecture element splits and merges are unusual in object-oriented modeling, they can be also handled. Upon a *split* with $p \sim_{PN} q_1$ and $p \sim_{PN} q_2$ $(q_1 \neq q_2)$, the tokens $m(p) = m'(q_1) = m'(q_2)$ are copied. *Merges* $p_1 \sim_{PN} q$, $p_2 \sim_{PN} q$ are only possible when $m(p_1) = m(p_2)$ so that $m'(q) = m(p_1) = m(p_2)$ is well-defined.

5.2.3 Augmenting architecture models with run-time features

For capturing the effects of run-time events, especially failures and execution of tasks, the *static* architecture model is extended by so-called *run-time attributes*, which are computed from the markings of the Petri net fragments that represent these events.

Precisely, run-time attributes are computed from the marking m_j in each system state $\langle M_i, m_j \rangle$ and are affixed to the objects of the architecture model M_i. Reconfiguration strategies can be formulated without knowing the details of the analysis model and the related transformation. Instead only the architecture model and its run-time extensions are accessed, promoting information hiding and modularization.

Observable run-time attributes are read-only attributes computed from the marking of the Petri net. In contrast, *controllable* run-time attributes can be modified by reconfigurations, causing changes in the Petri net marking. For simplicity, a controllable attribute must correspond to the number of tokens on a single place p, so that updating the attribute updates $m(p)$ directly.

To prevent the change of the Petri net marking from triggering the construction of a new analysis model, graph patterns may not depend on run-time attributes.

> **Example 5.3** The analysis model for the architecture in Fig. 5.3 is shown in Fig. 5.6.a. The static transformation incorporates failure models for machines into the FMS model adapted from [CT93]. The dashed rectangles correspond to the Petri net fragment instances and their horizontal traces.
> The marking of items (for a stockpile) is the initial marking is the initialSize of the Stockpile. The timed transitions deliver and putBack model pallets which move work items between stockpiles and machines, while the immediate transitions begin and abort are responsible for starting tasks and aborting them when a fault occurs.

5.3 Mission automata

In-place graph transformation rules (GT) controlled by an abstract state machine (ASM) were proposed [VB07] as a mathematically precise description of model changes. In this section we define a stochastic and timed variant of GT+ASM for reconfigurations of automated production systems.

Informally, the *mission automaton* is a GT+ASM running along the stochastic Petri net analysis model. Transitions in the automaton may be triggered by changes of run-time attributes in the analysis model or by the elapsing time. Actions attached to transitions may reconfigure the architecture model, as well as update a set of global variables.

Fig. 5.6.b illustrates the data flow between the mission automaton, the architecture models and the stochastic Petri nets. In the mission automaton, interactions between the static elements and run-time attributes of the architecture model are avoided by forbidding access to the run-time attributes in actions that modify the static architecture elements. Instead, as described in the next section, a specific run-time attribute update action is offered with limited control flow.

Therefore, the parts of mission automaton that depend solely on the static architecture model are separated from those that also depend on run-time attributes, and hence the Petri net marking. Thus the state space of the mission automaton can be overapproximated without exploring the state spaces of the derived Petri nets. State space and probability distribution handling is delegated to a PMS analysis tool.

5.3.1 Formal definition

A *mission automaton* is a 5-tuple $\langle L, \ell_0, F, G, T \rangle$, where L is the set of *locations*, $\ell_0 \in L$ is the *initial location*, $F \subseteq L$ is the set of *final locations*, G is the set of *global variables*, and T is the set of *transitions*. A transition $\langle \ell_1, \ell_2, V, \phi, \vec{x}, \tau, \vec{\alpha} \rangle \in T$ from $\ell_1 \in L$ to $\ell_2 \in L$ is equipped with a set of local variables V, a k-parameter *precondition pattern (guard)* ϕ, a list of *parameters* $\vec{x}$, a *trigger* τ and list of *actions* $\vec{\alpha}$. The parameters $\vec{x} = \langle x_1, \ldots, x_k \rangle$ are global or local variables ($x_i \in G \cup V$), such that each local variable $v \in V$ appears at least once in $\vec{x}$. Variables will be bound to objects of the architecture model.

5.3.1.1 Expressions

Algebraic and logical *expressions* can be formed using the attributes of objects $\mathcal{O}(M)$ pointed by the variables. We do not specify syntax and semantics for the expressions other than the

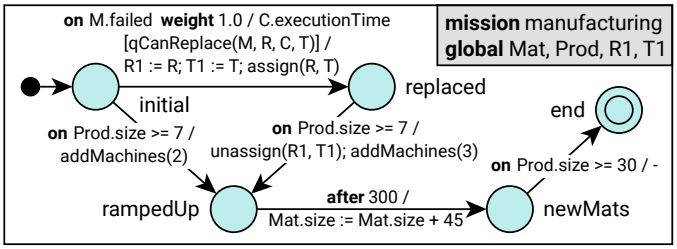


Figure 5.7: Example mission automaton

attribute references $x.a$, where a is an attribute from the architectural metamodel and $x \in G \cup V$.

An expression is *run-time attribute dependent* if it contains an attribute reference $x.a$ where a is a run-time attribute.

5.3.1.2 Triggers

The trigger τ may be a *(a) state-based* trigger "**on** e **weight** w", where e and w are (possibly run-time attribute, and thus marking dependent) Boolean and *weight* expressions, respectively; or a *(b) timed* trigger "**after** d", where the delay expression d is not dependent on the value of any run-time attribute.

5.3.1.3 Actions

Each action α_i of $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_m \rangle$ is either a *(a) variable update* "$g := x$", where $g \in G$ and $x \in G \cup V$; an *(b) attribute update* "$x.a := e$", where $x \in G \cup V$, a is a controllable attribute and e is a (possibly run-time attribute dependent) *value* expression; or a *(c) model manipulation action.* Model manipulation actions modify the architecture model by adding or removing objects or references, as well as changing static attributes. Due to space constraints, we refer to an existing *host language*[1] for semantics model mutation, as well as algebraic graph transformations for a precise mathematical foundation [Ehr+06; VB07].

> **Example 5.4** Fig. 5.7 shows a mission automaton for FMS reconfiguration, which showcases the expressiveness of our formalism. Transitions are denoted as $\tau\ [\phi(\vec{x})]/\vec{\alpha}$, where the guard is omitted if it is the trivial graph pattern (which has 0 parameters and holds always).
> The global variables Mat and Prod should be initialized to the stockpiles of raw materials and final products, respectively. When the FMS completes 7 products, the automaton moves to the rampedUp location from initial, and a model manipulation action adds 2 more machines. However, if some machine M responsible for a HIGH importance task T fails before ramping up, the transition to replaced is taken instead. The pattern qCanReplace (Fig. 5.4) also binds local variables to the replacement machine R and its capability C, so we can give faster machines larger weight among multiple possible replacements. R and T are saved to global variables R1 and T1, and a model manipulation action assigns T to R. When ramping up, this assignment is removed, and 3 more machines are added. 300 time units later, 45 more raw materials are added by an attribute update action. The final location end is reached after the FMS completes 30 products.

5.4 Phased-mission analysis

Analysis of architecture models and mission automata is performed in two steps. Firstly, the mission automaton is *unfolded* by taking into account the potential instantiations of its transitions,

[1] The Xbase language and the Xtext framework (`https://www.eclipse.org/xtend/`) were used as the hosts in our implementation.

as well as the modifications of the architecture model instance. The unfolded mission automaton and architecture model configurations form a tree (Fig. 5.5).

The (indirectly) marking dependent behavior of the mission automaton is overapproximated by ignoring triggers in the automaton. Each mission automaton transition is considered fireable, regardless of the reachable markings of the stochastic Petri net analysis model.

Dependence on the static and run-time parts of the architecture models is *erased* from the triggers and actions by substituting architectural concepts with their Petri net representations in the analysis model.

Secondly, the tree of architecture configurations is turned into a PMS for analysis. The *upper level* model is the unfolded mission automaton, where transitions between phases of operation are governed by the trigger expressions. The *lower level* models are the stochastic Petri nets that describe the behavior of the system during a phase. As the mission completes successfully upon reaching a final location, the corresponding phases are marked in the upper level model.

The hierarchical methodology proposed by Mura and Bondavalli [MB99; MB01] can be adapted for the PMS analysis.

5.4.1 Mission unfolding

We now present an explicit state-space exploration method for determining the reachable configurations of the mission automaton. Based on the reachable configuration, we derived the upper and lower level models for the PMS.

In the resulting PMS, the firings of Petri net and mission automaton transitions are interleaved (in a cyclic execution) as follows:

1. Immediate Petri net transitions are fired first.

2. Reconfigurations described by the mission automaton happen in response to the behavior of the stochastic Petri net, as well as after deterministic delays associated with timed triggers elapse.

3. Timed Petri net transitions are only fireable if no mission automaton transitions can be fired.

5.4.1.1 Mission automaton configurations

We define a *configuration* of the automated production system model as a tuple $\langle M, val, \ell \rangle$, where M is an architecture model, $val\colon G \to \mathcal{O}(M) \cup \{\bot\}$ is a *global variable valuation*, and $\ell \in L$ is the active location of the mission automaton.

Given an initial architecture model M_0 and initial valuation val_0 the unfolding determines the set configurations reachable from $\langle M_0, val_0, \ell_0 \rangle$, where ℓ_0 is the initial location of the mission automaton.

5.4.1.2 Transition instances

Let $t = \langle \ell_1, \ell_2, V, \phi, \vec{x}, \tau, \vec{\alpha} \rangle \in T$ be a mission automaton transition. A global variable valuation may be *extended* along t to a local valuation $lval\colon G \cup V \to \mathcal{O}(M) \cup \{\bot\}$, where $lval(g) = val(g)$ for all $g \in G$, while local variables $v \in V$ take some value $lval(v) \in \mathcal{O}(M)$ from the architecture model. A valuation is *acceptable* for t if the precondition $M \vDash \phi(lval(x_1), \ldots, lval(x_k))$ holds.

Acceptable transition instances in $\langle M, val, \ell \rangle$ are pairs $\langle t, lval \rangle$ where $lval$ is an acceptable extension of val along t and the active location ℓ is the source location ℓ_1. When exploring the reachable configurations, each acceptable transition instance is considered fireable.

For each transition instance $\langle t_i, lval_i \rangle$, its variable update actions are executed to yield the updated valuation val'_i. Moreover, the model manipulation actions are executed on M with the local valuation $lval_i$ in scope to obtain the reconfigured architecture M'_i. Thus a new reachable configuration $\langle M'_i, lval'_i, \ell'_i \rangle$ is found, where ℓ'_i is the target location of t_i.

The exploration strategy is an overapproximation in the sense that the run-time attribute dependence of the mission automaton transitions is ignored. Each transition instance with a state-based trigger is considered fireable, even if no marking of the Petri net analysis model is reachable with positive probability (or is reachable at all) that satisfies the trigger. Therefore, some extraneous configurations may be explored that have 0 probability of being reached. In Section 5.4.2, where the solution of the phased-mission model is discussed, these configurations will propagate zero probability vectors.

Architecture models M_i are not checked for equivalence, thus all reached configurations are considered different. This results in a tree of explored configurations with $\langle M_0, lval_0, \ell_0 \rangle$ at its root, but precludes the use of loops.

> **Example 5.5** Fig. 5.8 shows the reachable configurations of the mission automaton in Fig. 5.7 with the initial architecture in Fig. 5.3 and the initial global valuation {Mat ↦ *s1*, Prod ↦ *s4*, R1 ↦ ⊥, T1 ↦ ⊥}.
> The architecture model M_1 was obtained from the initial architecture M_0 by adding two new *Machines*. The static structures of M_2 and M_3 coincide with M_1; however, in M_2 the run-time attribute *s1.size* was increased by 45.
> In M_4 the additional task *a1* was assigned to *m1* via its capability *c3*, and this fact is memorized in the global variables R1 and T1. M_5 is obtained from M_4 by unassigning *a1* from *m1* and adding two new machines, while in M_6, the run-time attribute *s1.*size was increased. Lastly, M_7 conincides with M_6.
> Increasing the number of Machines would lead to an increased number of qCanReplace graph pattern matches, and thus more initial ↦ replaced transition instances would be acceptable. Hence more children would be added to the root configuration of the tree.

5.4.1.3 Erasure

We define $[\![e]\!]_{lval}$ be the *erasure* of the expression e. The goal of erasure is to resolve the architectural concepts (e.g. run-time attributes) by substituting these with their Petri net based representations, so that places of the analysis model *PN* associated with architecture model M of the active configuration are referenced.

The erased expressions that reference lower-level Petri net elements appear in the higher-level model of the PMS, so that the PMS can be processed by Petri net analysis tools without referencing the architecture models.

For every reference $x.a$ to an attribute a of the variable $x \in G \cup V$ first the object $o = lval(x)$ is looked up in the architecture model. If a is a run-time attribute, the vertical trace is traversed and $x.a$ is replaced in e with a function that calculates a of o from the Petri net marking. Otherwise $x.a$ is replaced with the value of the attribute a of o. Therefore the erasures of marking dependent expressions are functions of the Petri net marking such that $[\![e]\!]_{lval}(m)$ is the value of the expression in the marking m, while the erasures of marking independent expressions are plain values $[\![e]\!]_{lval}$.

The erasure $[\![\tau]\!]_{lval}$ of a trigger replaces its constituent expressions with their erasures.

The erasure of the attribute update action $\alpha = x.a := v$ is $[\![\alpha]\!]_{lval} = p := [\![v]\!]_{lval}$, where p is the place associated with the controllable attribute a of $lval(x)$.

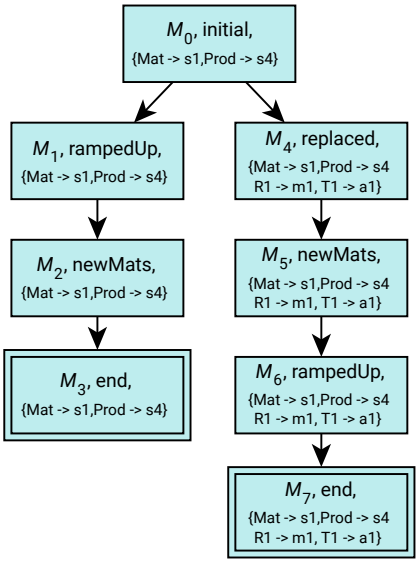


Figure 5.8: Reachable configurations of the mission automaton in Fig. 5.7 with the initial architecture in Fig. 5.3. Variable bindings to ⊥ were omitted.

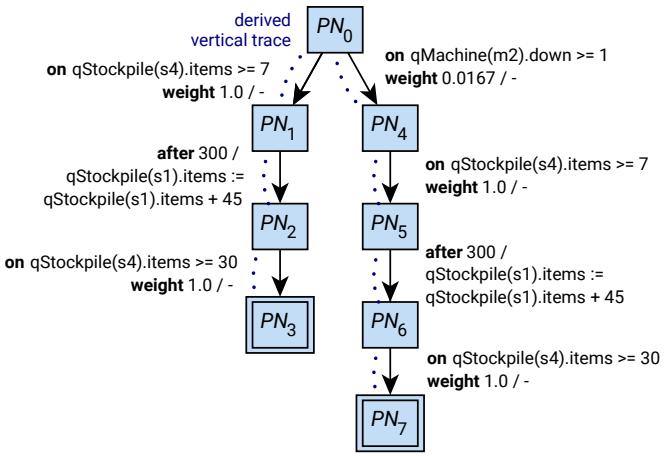


Figure 5.9: Example PMS upper-level model based on Fig. 5.8.

5.4.1.4 Phased-mission construction

As mentioned in Section 5.2.2, each reachable configuration $\langle M_i, val_i, \ell_i \rangle$ is turned into a lower level PN_i model of the phased-mission system, which represents production system behavior during a single phase.

The upper level model is a tree of the lower level models. If the configuration $\langle M_j, val_j, \ell_j \rangle$ was obtained from $\langle M_i, val_i, \ell_i \rangle$ by firing the transition instance $\langle t, lval \rangle$, the tree edge $PN_i \longrightarrow PN_j$ is annotated with

- the erasure $\tilde{\tau}_{ij} = [\![\tau]\!]_{lval}$ of the trigger τ of t,
- the erasures $\tilde{\vec{\alpha}}_{ij} = \langle \tilde{\alpha}_{ij}^{(1)}, \ldots, \tilde{\alpha}_{ij}^{(k)} \rangle$ of the attribute update actions associated with t,
- the derived vertical trace $\sim_{ij} \subseteq PN_i \times PN_j$.

Since global variable update and model manipulation actions are executed during the exploration of reachable configurations, they are not needed in the numerical solution of the PMS. We only preserve attribute update actions, since they can modify the Petri net marking.

For each lower-level model (*phase*) PN_i, there may be at most one tree edge $PN_i \longrightarrow PN_j$ with a timed trigger $\tilde{\tau}_{ij} = \textbf{after}\ \tilde{d}_{ij}$. Note that weight-based random selection of delayed phase transitions can be written as a combination of an "**after** d" and several "**on** true **weight** w" triggered transitions in the mission automaton. Therefore this restriction is merely syntactic. We call this edge the *delay edge* of PN_i. Because timed triggers cannot be marking dependent, $\tilde{d}_{ij}$ is a constant, which we call the *sojourn time* c_i of PN_i.

The phase PN_i is *final* if the corresponding location ℓ_i is final in the mission automaton.

> **Example 5.6** Fig. 5.9 show the upper-level model obtained from the tree of configurations in Fig. 5.8.
> Each transition is annotated with the erasure of its trigger and its attribute update actions, which only refer to Petri net level concepts. Elements of the Petri net are identified according

to the horizontal trace for readability. However, horizontal trace links may be replaced by unique IDs to avoid referencing the architectural model.
Transitions also carry derived vertical trace information. Therefore the effects of degradation processes and other information in the markings can be propagated.

5.4.2 Solution of the hierarchical model

Starting from the root, the nodes of the upper level tree are traversed to determine the probability of successful mission completion. The lower level model PN_i, which describes system behavior during the traversed phase, is solved by steady-state or transient stochastic analysis [RST89]. The resulting probability distribution vectors are propagated through the tree to complete phased-mission analysis [SRA92; MB99].

A (sub)probability distribution over the markings of the lower level model PN_i is represented as a *(sub)probability vector* $\vec{\pi}$. Each $\pi[a]$ is the probability of some reachable marking m_a.

The generator matrix Q of the CTMC PN_i governs the time evolution $\frac{\partial}{\partial t}\vec{\pi} = \vec{\pi}Q$ of $\vec{\pi}$.

Probability vectors have the property $\vec{\pi} \cdot \vec{1}^T = 1$, i.e. the sum of marking probabilities is 1. Branchings of the upper level tree introduce subprobability distributions, which satisfy $0 \leq \vec{\pi} \cdot \vec{1}^T < 1$ instead.

5.4.2.1 Lower-level model solution

Consider a node PN_i of the upper level model. Without loss of generality, let the child (successor) nodes of PN_i with state-based triggers be $PN_j, \ldots, PN_{j+m-1}$, where the erasures of the triggers are $\tilde{\tau}_{i,j+\ell} = \textbf{on } \tilde{e}_{i,j+\ell} \textbf{ weight } \tilde{w}_{i,j+\ell}$ $(\ell = 0, \ldots m-1)$.

Analysis of PN_i starts from the *initial (sub)probability vector* $\vec{\pi}_i$. In the initial phase ($i = 0$), $\vec{\pi}_0$ assigns probability 1 to the initial marking of PN_0 and 0 for any other marking. Otherwise, PN_i is computed by a previous analysis step.

The set of reachable markings of PN_i should be explored from every marking $supp\ \vec{\pi} = \{m_a \mid \pi[a] > 0\}$ in the support of the initial (sub)probability distribution.

When the generator matrix Q is constructed, transition rates from markings satisfying $\tilde{e}_{i,\text{any}} = \bigvee_{\ell=1}^{m-1} \tilde{e}_{i,j+\ell}$ are set to 0. Hence the marking will not change after the first time a state-based trigger is activated. In contrast, $\neg\tilde{e}_{i,\text{any}}$ markings are yet to activate any state-based trigger.

If $PN_i \xrightarrow{\textbf{after } \tilde{d}_{i,j+m}} PN_{j+m}$ is the delay edge of PN_i, the transient solution $\vec{\pi}'_i$ at time $c_i = \tilde{d}_{i,j+m}$ is calculated. Otherwise, let $\vec{\pi}'_i$ be the steady-state subprobability vector from the initial subprobability distribution $\vec{\pi}_i$.

5.4.2.2 Outgoing subprobability computation

For each edge $PN_i \xrightarrow{\textbf{on } \tilde{e}_{i,j+\ell} \textbf{ weight } \tilde{w}_{i,j+\ell}} PN_{j+\ell}$ the vector $\vec{\pi}''_{i,j+\ell}$ is calculated by distributing subprobability mass from $\vec{\pi}'_i$. If $\tilde{e}_{i,j+\ell}$ holds in m_a, then $\vec{\pi}''_{i,j+\ell}[a] = \vec{\pi}'_i[a]$. If there are multiple such ℓ, the values are distributed in proportion to $\tilde{w}_{i,j+\ell}(m_a)$.

The remaining mass is $\vec{\pi}''_{i,*} = \vec{\pi}'_i - \sum_{\ell=0}^{m+1} \vec{\pi}''_{i,j+\ell}$. If there is a delay edge from PN_i, let $\vec{\pi}''_{i,j+m} = \vec{\pi}''_{i,*}$. Otherwise $s_i = \vec{\pi}''_{i,*} \cdot \vec{1}^T$ is the probability of getting *stuck* in the phase PN_j. If PN_j is a final phase, s_j is added to the probability of successful mission completion.

Now the initial subprobability vectors of the subsequent phases $PN_{j+\ell}$ are computed. For each update action $\tilde{a}^{(u)}_{i,j+\ell}$, a matrix $A^{(u)}_{i,j+\ell}$ shifts the subprobability masses according to the reassignment of token counts. Thus $A_{i,j+\ell} = \prod_u A^{(u)}_{i,j+\ell}$ expresses the whole action sequence $\tilde{\vec{\alpha}}_{i,j+\ell}$.

Table 5.1: Measurement results

	initial phase		total phases			
N	$\lvert PN_0\rvert$	time/ms	phases	$\sum_i \lvert PN_i\rvert$	time/ms	ph.t./ms
1	41	186	8	415	303	16.6
2	78	185	20	1818	426	12.7
4	152	213	68	11 278	1022	12.1
6	226	247	148	35 530	2150	12.9
8	300	281	260	81 678	3955	14.2
10	374	319	404	156 826	6267	14.8
12	448	353	580	268 078	9346	15.5
14	522	390	788	422 538	13 729	16.9
16	596	426	1028	627 310	19 481	18.6

The horizontal trace $\sim_{i,j+\ell}$ also causes a shift of subprobabilities. Removal of places causes the summation of masses, while the addition of a new place p assigns the masses to markings of $PN_{j+\ell}$ where p bears its initial token count. This shift is expressed by the matrix $T_{i,j+\ell}$.

The matrix $\Lambda_{i,j+\ell} = A_{i,j+\ell} T_{i,j+\ell}$ converts $\vec{\pi}''_{i,*}$ to the initial subprobability vectors $\vec{\pi}_{j+\ell} = \vec{\pi}''_{i,*} \Lambda_{i,j+\ell}$ of $PN_{j+\ell}$. Analysis can be performed recursively on the child phases.

5.5 Evaluation

We implemented a domain-specific language for mission automata, as well as the mission automaton unfolding and erasure algorithm from Section 5.4, using the VIATRA incremental transformation engine [VB07; Ber+15; Ujh+15] for EMF models and the Xtext language engineering framework.

We evaluate the scalability of our analysis model construction approach in the context of incremental analysis model transformations. Reachable configurations are explored during the unfolding in a depth-first manner. Exploiting the transaction support of EMF both forward moves and backtracking modify the architecture model in-place. Hence the static transformation, which is also powered by VIATRA, can be executed in a change-driven style.

Setup: The mission automaton from Fig. 5.7 was unfolded with scaled versions of Fig. 5.3, such that the initial architecture contains N copies of the machines *m1* and *m2*, respectively. The query qCanReplace matches N^2 pairs of machines, thereby increasing the number of phases.

The experiments were ran on a computer with an Intel Core i7-5700HQ 2.7 GHz CPU with 4 GiB of heap space reserved for the Java 1.8.0u144 virtual machine. Each experiment was repeated 30 times after 10 warm-up repetitions.

Results: Table 5.1 presents the number of elements in the Petri nets obtained by unfolding, as well as the median running times. We report the running time of the transformation of the initial phase, including the execution of the static transformation in batch mode. We also report the total running time, and the average incremental execution time for the non-initial phases (ph.t.).

Discussion: The sizes of the analysis models for a single phase grew linearly as machines were added to the architecture, while the number of phases in the upper level model grew quadratically, along with the execution time of the full PMS construction. The unfolding could take advantage of incremental execution of the static transformation. Thus no more than 20 ms

per non-initial analysis model was taken. Total execution time remained below 20 seconds.

5.6 Related work

Methods for the construction of stochastic analysis models are widespread in the evaluation of architecture models, especially for component-based design [Koz10; BMP12]. The Palladio component model [BKR08] was suggested as a specialized modeling language for performance prediction. Recently, the Renew metamodeling framework [MCH16] was proposed for adding Petri net semantics to modeling languages. In [Get+18] co-evolution of architectures and fault trees were investigated.

A dependability-driven design methodology for embedded systems was suggested in [Sur+10], while [Hoa+17; RSV17] investigated adaptation for dependability by agent-based planning.

The two main groups of dependability analysis techniques for phased-mission systems are state-space based and combinatorial approaches [WT07]. State-space based analysis may happen by solving stochastic models for individual phases and propagating the results [SRA92; MB99], as well as with deterministic and stochastic Petri nets models [MB01]. Combinatorial approaches can take advantage of efficient data structures [Xin07; WT07], but may require more restricted models, such as fault trees or block diagrams.

Combinations of CTMCs were investigated with timed and hybrid automata [DHS09; Che+11a; Bal+15]. While these approaches are significantly more expressive for linear-time properties than ours, they do not support the reconfiguration of the model.

5.7 Conclusions

We presented (1) a mission automaton formalism for specifying reconfigurations and fault handling in production system architectures and (2) a mapping from these to stochastic PMS analysis models. The dependence between the phase transitions in the mission and the events represented in the analysis model is abstracted by incorporating run-time attributes into the architecture model. According to our empirical evaluation, good scalability can be provided by our incremental analysis model construction.

Possible extensions include relaxing the separation between the static and the run-time attribute dependent parts of the mission automata, for example, by adopting stochastic timed automata model checking techniques [Che+11a]. Graph-based state encoding [HHV15] during mission unfolding may reduce the PMS from a tree to a directed graph, potentially decreasing the number of phases to be analyzed by merging equivalent phases.

Chapter 6

Worst-Case Execution Time calculation for query-based monitors

Runtime monitoring has become a key technique in the assurance of safety-critical and intelligent cyber-physical systems (CPS) such as autonomous vehicles [Pek+20] (e.g., self-driving cars, drones) where traditional upfront design time verification is problematic due to the dynamically changing environment and the data-intensive nature of the system. However, it is an open challenge how to align real-time requirements with the ability to capture the highly dynamic operation context of the system. A promising line of research aims to leverage high-level, model-based techniques to manage system complexity [JDR17; TH19].

In traditional embedded systems, runtime monitoring programs are integral components of the system that analyze events and execution traces [Bar+18] in order to detect potentially critical situations that violate a requirement. Since this requires formal precision to capture safety requirements, logic-based formalisms (e.g., propositional logic, temporal logic) are frequently used to specify execution traces. Furthermore, monitoring programs can be automatically synthesized from such specifications that are ready to be used in traditional hard real-time systems without compromising task schedulability and real-time properties of the existing program [Pik+10; HR02].

However, existing runtime monitoring approaches used in safety-critical applications have certain limitations, which are increasingly problematic for the new generation of data-intensive, intelligent, and self-adaptive, yet safety-critical CPSs. First, the *moderate expressiveness of the specification language* [Hav15] makes it difficult for engineers to capture and understand complex rules. Moreover, while *safety-critical programs typically use statically allocated data with bounded input sizes* and they conservatively avoid many programming language constructs, dynamically evolving data and advanced language constructs are inherent parts of data-intensive programs.

Recent advances in runtime monitoring aim to overcome these limitations by (1) offering high-level and expressive query-based [DBB18; Búr+18] or rule-based [Hav15] formalisms to capture the properties to be monitored, and (2) using runtime graph models as an in-memory knowledge base which capture dynamic changes in the system or its environment at a high-level of abstraction [Búr+18; Har+19]. Such *data-driven safety monitors* derived from high-level specifications can analyze aggregated changes triggered by complex sequences of atomic events by evaluating queries over a continuously evolving data model. For example, in the railway domain, queries can check if a path exists between two points along a railway track, or identify cargo waiting at stations for more than a specified duration. As such, query-based monitoring programs use a network of linked objects as data structures and *exhibit heavily input-dependent and semantic-aware complex control and data flow.*

To enable the use of data-driven safety monitors in hard real-time systems, the computation

of safe worst-case execution time (WCET) estimates is required. While recent research has investigated data-driven runtime monitors for intelligent and critical CPSs in a distributed environment [Búr+18; Har+19; DBB18], and various testing approaches have been proposed [Abd+18; SNV18], the timeliness aspect of the problem has been neglected. In fact, only very few initial ideas are available [TGS06], which suggest limiting the maximum graph size and employing optimized query plans. A wide range of existing timing analysis techniques and tools (e.g., aiT [FH04], Chronos [Li+07], OTAWA [CS06], and SWEET [Lis14]) can provide safe and tight WCET bounds for traditional critical embedded software. However, there is a high degree of inherent design-time uncertainty present in data-driven monitoring programs. In particular, the unknown contents of the runtime knowledge graph capturing the system and its operating environment constitutes an enormously large input space which can compromise the accuracy of existing techniques. Therefore, novel techniques are needed to complement existing WCET analysis techniques to efficiently incorporate domain-specific restrictions for program inputs on a high-level of abstraction and automatically incorporate this domain knowledge as flow facts.

In order to obtain safe and tight WCET bounds for data-driven runtime monitors, major challenges in timing analysis need to be tackled. (i) First, domain-specific flow constraints pose several complex restrictions on the program flow, but the respective flow facts need to be manually formulated and the program needs to be annotated by experts. Such additional program flow information largely helps to enhance the precision of safe WCET bounds in existing timing analyzers. However, there are no generally applicable methods to automatically obtain such flow facts by exploiting high-level, domain-specific information and constraints on program flow during timing analysis. Specifying the flow facts manually is highly error-prone and the resulting annotations need to be updated after subsequent modifications to the program [Abe+15]. (ii) Furthermore, runtime graph models have varying underlying structure and memory demands which makes WCET analysis problematic since the worst-case graph structure needs to be considered regardless of domain-specific constraints. While memory can be preallocated to allow the timing analyzer to produce WCET bounds [HR09], but a WCET estimate from the size of the preallocated memory for graph data without appropriate flow facts would still be overly conservative. Moreover, the contents of the runtime model are regularly updated at runtime. Therefore, value analysis has no upfront access (at design-time) to the data that the reserved memory space will store. (iii) Finally, traditional WCET analysis challenges also need to be tackled: detailed information is required about the executable binary and execution platform, including precise memory, pipeline, and cache descriptions [Wil+08].

Contributions This paper aims to address WCET estimation in the challenging setting of query-based runtime monitoring programs. In particular, we present the following contributions.

1. We adapt query-based runtime monitoring programs derived from high-level graph query specifications [Búr+18] to real-time platforms (Section 6.1.3).

2. We provide a novel high-level static analysis technique for query-based monitors to estimate execution time on a given runtime model. The approach provides precise flow facts by counting the number of basic block executions during query evaluation w.r.t. the given model even if exact memory allocation information is unavailable. Moreover, it combines such flow facts with constraints obtained from existing low-level analysis tools (Section 6.3.3).

3. We estimate the WCET of query-based programs by estimating execution time over designated *witness models*. Such witness models have the highest estimated execution times for graph query programs executing over any input models up to a predefined model size and domain-specific constraints, and they are derived by a state-of-the-art *graph solver* [SNV18] (Section 6.3.4).

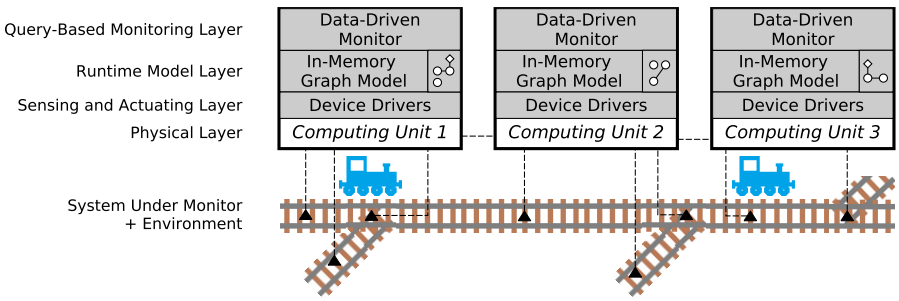


Figure 6.1: Runtime monitoring by graph queries

4. We perform an extensive experimental assessment of query evaluation times over a variety of graph models executed on an industry-grade real-time platform, and we compare our WCET estimates with those provided by two popular timing analyzers (OTAWA and aiT) (Section 6.4).

Novelty Our technique complements existing WCET estimation methods by extracting program flow information from domain-specific constraints of the abstract input space on top of existing constraints derived by traditional timing analysis. To our best knowledge, our approach is the first to provide safe and tight WCET bounds of real-time graph query programs by abstraction refinement using state-of-the-art model generation techniques. This enables the use of query-based runtime monitoring programs in a real-time context by providing safe WCET estimates for a desired set of input models satisfying domain-specific constraints.

The contents of this chapter are based on the journal paper [j5]. *Code generation* for embedded query programs (Section 6.1) and the concept of *witness models* are the contributions of Márton Búr [Búr+20; Búr21][j5].

6.1 Query-based runtime monitors[1]

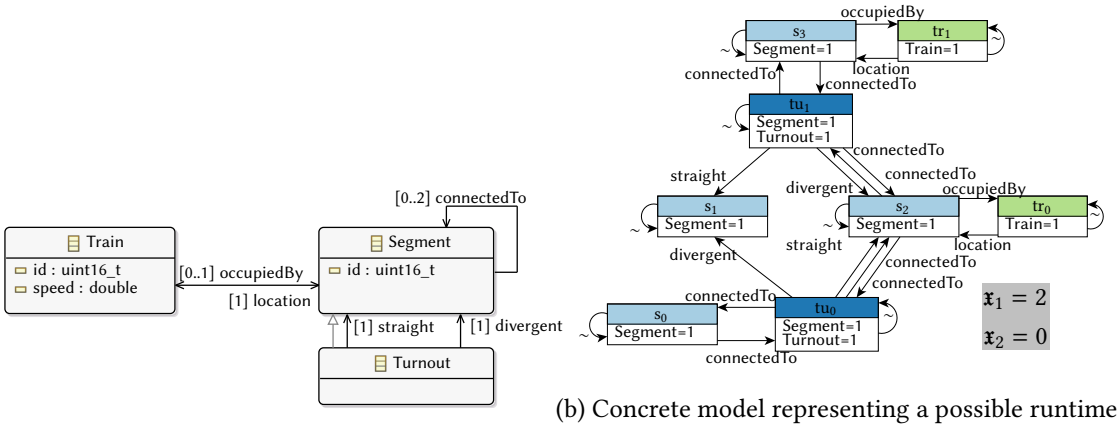
In this section, we provide an informal overview of query-based runtime monitors, while their formal treatment is deferred to Section 6.2.

6.1.1 Running example: the MoDeS3 CPS Demonstrator

Our key concepts are illustrated in the context of the open source *Model-Based Demonstrator for Smart and Safe Cyber-Physical Systems* (MoDeS3) [Vör+18b] platform, which showcases various challenges of modern intelligent yet safety-critical CPS applications. The demonstrator (see Figure 6.1) is a model railway system with an added layer of safety to prevent train collision and derailment using runtime monitors. The railway track is equipped with several sensors and actuators, which are represented by black triangles in the lower part of Figure 6.1. Train shunt detectors can sense when trains pass by on a particular segment of the track, while direction of turnouts can be read and set.

The system is managed by a (distributed) monitoring service running on a network of heterogeneous computing units, such as Arduinos, Raspberry Pis, BeagleBone Blacks, etc. Relevant runtime information gained from sensor reads (e.g., the occupancy of a segment, or the status of

[1]This section was directly adapted from [Búr21][j5] and does not comprise the contribution of the author of this thesis. Nevertheless, we include it for completeness and to provide background for query-based runtime monitors in order to make the thesis self-contained.



(a) Metamodel of the MoDeS3 domain

(b) Concrete model representing a possible runtime snapshot

Figure 6.2: An instance (concrete) model and a partial model in the MoDeS3 domain

a turnout) is uniformly captured in an in-memory *runtime graph model*, which is also deployed on the platform.

Safety monitors are formally captured as *graph queries*. Alerts from the monitoring services may trigger control commands of actuators (e.g., to change turnout direction) to guarantee safe operation. The monitoring and control programs are running in a real-time setting on the computing units.

While the MoDeS3 platform can demonstrate various challenges of CPSs, this paper exclusively focuses on the real-time aspect of query-based runtime monitoring programs deployed to some embedded devices with limited resources (memory, CPU, etc).

6.1.2 Graph models at runtime

6.1.2.1 Graph models

The **models@run.time** paradigm [BBF09] facilitates the capture of runtime knowledge about the system and its environment as a (typed and directed) *graph model* continuously maintained at runtime for the system. Such graphs are dynamically changing in-memory data structures which encode domain-specific instance models typed over a *domain metamodel*, which captures core concepts (classes) in a domain and the relations (references) between those concepts.

> **Example 6.1** The domain concepts of the MoDeS3 runtime model are captured in a metamodel excerpt shown in Figure 6.2a using the Eclipse Modeling Framework (EMF) notation [Ste+09]. One domain concept is Train. Class Segment represents a section of the railway track with the connectedTo reference which describes what other segments it is linked to (up to two). Moreover, each train maintains a location reference to a segment to describe its current position. Instances of class Segment record if they are occupied by a train with the occupiedBy reference. Moreover, Turnout is a special Segment that can change its connections between straight and divergent segments.

A runtime (instance) model captures a snapshot of the underlying system in operation [BBF09; SZ13]. Relevant changes in the system are reflected in the runtime model and operations executed on the runtime model (e.g., setting values of controllable attributes of objects or updating links between objects) are reflected in the system itself (e.g., by executing scripts or calling services). In this work, we use *concrete (graph) models* to formally capture runtime model snapshots.

```
1 typedef struct {
2   uint16_t segment_id;
3   Train *train;
4   Segment *connected_to[2];
5   uint8_t connected_to_count;
6 } Segment;
```

Listing 6.1: Segment class

```
1 typedef struct {
2   uint16_t train_id;
3   double speed;
4   Segment *location;
5 } Train;
```

Listing 6.2: Train class

```
1 struct Modes3ModelRoot {
2   Segment segments[SEGMENTS];
3   uint16_t segment_count;
4   Train trains[TRAINS];
5   uint16_t train_count;
6 } runtime_model;
```

Listing 6.3: Graph model root

> **Example 6.2** Figure 6.2b shows a concrete model in a graphical syntax. The graph has six Segment objects (including two Turnouts) with their respective connectedTo links. Turnouts tu_0 and tu_1 can switch between segments s_1 and s_2 (see their straight and divergent edges). In the depicted state, both turnouts are switched to s_2 and the trains tr_0 and tr_1 are located on s_2 and s_2, respectively.

6.1.2.2 Graph data structures in embedded systems

Runtime monitors captured by graph queries are continuously evaluated over the runtime models. This section informally summarizes our assumptions and requirements about such programs while the theoretical background is introduced in Section 6.2.

For data-driven monitors, the structure of the underlying graph model directly impacts the performance of query evaluation. Since an embedded device may have limited available CPU and memory resources, a lightweight data structure is needed to efficiently capture runtime graph models. While the in-depth discussion of such a graph data structure is out of scope for this paper, we make the following assumptions about the supported operations of the underlying graph:

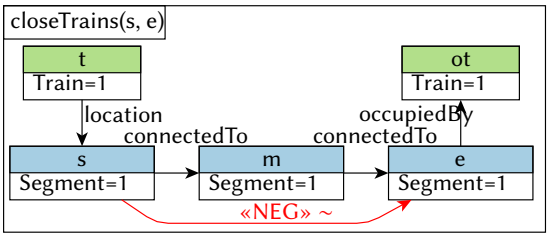
- **Dynamic element creation and deletion.** The runtime model serves as the knowledge base about the underlying system and its environment. For this reason, it needs to accommodate graph models without a theoretical a priori upper bound for model size. Based on [HR09], one way to support this is to allocate the maximum amount of memory that is physically possible to be used for storing the graph. However, only the allocated memory is determined at compile time, the type (and distribution) of objects stored in the graph is runtime information.
- **Indexing of objects by type using unique identifiers.** As query evaluation typically starts by iterating over all elements of a given type or accessing specific objects, it necessitates efficient object access, e.g., by maintaining a real-time index for memory resident data [CK96].
- **Navigability along edges.** Query evaluation often navigates along the edges of selected objects to find further appropriate variable substitutions for unbound query variables. This feature can be supported by, e.g., maintaining direct pointers to reachable objects.

It is also important to note that the same graph model can be represented in memory in many ways, because different placements of the same data can cause different run times. For example, two memory images of the same graph may differ in the order the objects are stored in the array. For this reason, two different in-memory representations of the same graph may not necessarily yield identical run times, which must be considered when computing WCET of graph query programs.

> **Example 6.3** Listing 6.1 and Listing 6.2 show a possible C implementation of data structures for Segment and Train classes in the metamodel of Figure 6.2a. Line 2 in Listing 6.1 and lines 2 and 3 in Listing 6.2 are fields created from respective attributes present in the metamodel,

```
1 pattern closeTrains(s, e) {
2   Train.location(_t, s);
3   Segment.connectedTo(s, m);
4   Segment.connectedTo(m, e);
5   Segment.occupiedBy(e, _ot);
6   s != e
7 }
```

(a) Description of a query in VQL



(b) Graphical query presentation

$$\varphi_{\mathrm{CT}}(s,e) = \exists t : \mathrm{Train}(t) \wedge \mathrm{Location}(t,s) \wedge \exists m : \mathrm{ConnectedTo}(s,m) \wedge \mathrm{ConnectedTo}(m,e) \wedge \neg(s=e) \wedge \exists ot : \mathrm{OccupiedBy}(e,ot)$$

(c) Graph query as logic predicate

Figure 6.3: Monitoring goal formulated as a graph query φ_{CT} for closeTrains

e.g., the speed attribute of class Train is represented by line 3. For each type, an `id` attribute encodes the type of the object for indexing and model manipulation purposes. Uniqueness of this attribute needs to be guaranteed at runtime to distinguish objects. Furthermore, in this example, we implement graph edges as pointers (line 4 in Listing 6.2) or pointer arrays with sizes (lines 4 and 5 in Listing 6.1). Representing links between objects with pointers is highly efficient from a performance viewpoint.
Listing 6.3 shows how a simple graph model container `Modes3ModelRoot` can allocate static memory for graph objects in C. The maximum memory used by the graph is preallocated in lines 2 and 4 by the `segments` and `trains` arrays which have a length of the maximum expected number of trains (denoted by the constant `TRAINS`) and the maximum expected number of segments (`SEGMENTS`). The `id` attribute of a given object used for indexing these arrays, i.e., encodes their positions in the arrays.

6.1.3 Graph query programs in real-time systems

Data-driven runtime monitors defined by graph queries can check structural properties of the runtime model representing a snapshot of the system. In other words, they focus on the most up-to-date data (maintained either by periodic updates with a certain frequency, or by certain event-driven triggers [BBF09; SZ13]) available on the underlying system's state at a given point of time.

Classical *event-based runtime monitors* rely on some temporal logic formalism to detect sequences of events occurring in the system at different points in time, while the underlying data model is restricted to atomic propositions. As such, data-driven and event-based monitors are complementary techniques. While graph queries can be extended to express temporal behavior, our current work is restricted to (structural) safety properties where safety violations are expressible by graph queries.

6.1.3.1 Graph Queries

A *graph query* is a declarative description of a model fragment to be identified by a set of variables and a set of constraints (type, reference, and equality assertions) [Var+15]. A *match* of the query is a binding of the query variables to objects in the model such that the constraints are satisfied. In data-driven runtime monitors, such high-level descriptions allow us to automatically generate and optimize the monitor program by adaptive *query planning*.

Algorithm 1: Code generation from search plans

```
1  Function CompileSearchPlan(sp, idx)is
2      if idx > sp.size() then return code for storing a match;
3      step = sp[idx]
4      matcherCode = ""
5      if step is extend then
6          for uv ∈ step.getFreeVariables() do
7              matcherCode +=
8                  AddAssignmentFor(uv,
                   step.getConstraintFor(uv))
9      else if step is check then
10         matcherCode +=
11             AddIfFor(step.getAllVariables(),
               step.getConstraint())
12     return matcherCode + CompileSearchPlan(sp, idx + 1)
```

Table 6.1: A possible search plan for query closeTrains where free variables are underlined

Constraint	Step#	Op. type
Train($\underline{t}$)	1	extend
Location(t, $\underline{s}$)	2	extend
ConnectedTo(s, $\underline{m}$)	3	extend
ConnectedTo(m, $\underline{e}$)	4	extend
$\neg(s = e)$	5	check
OccupiedBy(e, $\underline{ot}$)	6	extend

Example 6.4 Trains are required to keep long headway distances to ensure that they can safely decelerate without collision [Eme11]. The safety case captured by the *closeTrains* (CT) graph query represents a situation when the headway distance between trains located on connecting segments is reduced below the safety limit. Any match of this query highlights segments where immediate action (e.g., stopping the trains) is required. The declarative query specification is presented in Figure 6.3a in a textual syntax to identify the violating situation as a hazardous case. The query returns pairs of segments s, e where there is a train located on segment s that is one segment away (i.e., there is a middle segment m) from a different segment e, which is also occupied by a train. Any variables not appearing in the parameter list of the query are existentially quantified.
Figure 6.3b shows the same query in a graphical presentation often employed by modeling tools, and Figure 6.3c presents it as a first-order logic (FOL) formula φ_{CT} (discussed later in Section 6.2.2).
In the runtime snapshot Figure 6.2b, the variable bindings $\{s \mapsto \text{s}_2, e \mapsto \text{s}_3\}$ and $\{s \mapsto \text{s}_3, e \mapsto \text{s}_2\}$ are matches of the *closeTrains* query.

6.1.3.2 Local search-based graph query evaluation

Among the many possible query evaluation strategies [Gal06], our runtime monitoring framework uses *local search-based query evaluation* [Var+15] to find matches of monitoring over the entire runtime model. This strategy at its core uses a tailored depth-first search graph traversal. This keeps the memory footprint of the query evaluation algorithm constant. To obtain efficient performance at runtime, query evaluation is guided by a *search plan* [Var+15], which maps each constraint in the query to a single pair of ⟨*Step index*, *Operation type*⟩. In this tuple, *Step index* specifies the order in which query evaluation should attempt to satisfy the respective constraint. *Operation type* can be one of the followings:

- An **extend operation** evaluates a constraint with at least one free variable. Execution of such operations requires iterating over all potential variable substitutions and selecting the ones for which the constraint evaluates to 1.
- A **check operation** evaluates a constraint with only bound variables. Execution of such operations determines if the constraint evaluates to 1 over the actual variable binding.

Example 6.5 Table 6.1 shows a possible search plan for the φ_{CT} query. Each row represents a search operation. The first column shows which constraint is enforced by the given step where free parameters at the start of the execution of the operation are underlined. The

second column shows the ordering of steps, i.e., the step index, and the third column shows the search operation type (check or extend) which is based on the variable bindings prior to the execution of the search operation: if the constraint parameters are all bound, then it is a check, otherwise, it is an extend.

6.1.3.3 Implementations of query programs

Although constructing effective search plans for graph queries is a complex challenge, it is outside of the scope of the current paper and has been formerly extensively studied (see, e.g., [Var+15] for a possible solution). However, we present pseudo-code that generates embedded query code from a search plan in Algorithm 1. The function *CompileSearchPlan* takes a search plan and a search step index as parameters. Line 2 returns a code snippet to register a match if the provided index is beyond the index of the final search step. Otherwise, the search step is extracted (line 3) and the variable *matcherCode* to hold the generated code is initialized to an empty string (line 4). Then, the different operation types of the query search plan are translated to structured imperative code:

- Each **extend operation** binds all free variables of the respective constraint (lines 5–6). For each variable, this translates to either a single assignment or a `for` loop iterating over a set of candidate variable bindings, depending on the multiplicity of the respective navigation edge (reference constraint) (lines 7–8).
- Each **check operation** (line 9) is mapped to an `if` statement to check if the current variable binding satisfies a given condition created from the query constraint (lines 10–11).

Finally, in line 12, the generation continues recursively appending the code generated from the subsequent steps to the result. The query code for the entire search plan *sp* can be generated by calling *CompileSearchPlan*(*sp*, 1). As a result, the source code contains a deep hierarchy of embedded for-loops and if-statements based on the ordering of constraints prescribed by the search plan.

Besides obtaining a WCET, we also need to estimate the number of matches of a query to allocate appropriate space in memory in advance. In the case of runtime monitors of safety properties, we can assume that only a few violating matches will be detected [Var+18], thus the query result set is expected to be small and memory required for storing matches can be reserved at compile time.

Example 6.6 Listing 6.4 shows the C code generated from the query specification of closeTrains. Assuming that a global variable `model` points to the root of the entire graph model including its up-to-date model statistics, calling the function `close_trains_matcher` with a pointer to the result set structure `results` will compute and store all matches over the model in `results`.
In the example, initially all variables are assumed to be free, as indicated in line 2 with `NULL` values, because we aim to find all matches in the entire model. In line 3, the size of the result set is initialized to 0. The `for` loop in line 6 represents step 1 from the search plan (see Table 6.1) and iterates over all trains in the model, binding the variable `vars.t` to all possible objects in line 7. Lines 8–10 together represent search step 2. In line 9, `vars.s` is assigned a segment referred by `vars.t` via a single `location` link. If such a segment exists in line 10, execution continues with the third search operation that is mapped to lines 11–14, which iterates over segments connected to `vars.s` and assigns them to `vars.m`, one at a time. The next step in lines 15–18 does the same but with the connecting segments of `vars.m` and assigns them to `vars.e`. Search step 5 is a check, which is mapped to lines 19–20 to ensure that the segments referred by `vars.s` and `vars.e` are not the same. The final step of the search plan is mapped to lines 21–23. Here the train occupying the segment stored in `vars.e` is assigned to `vars.ot`.

```
1  void close_trains_matcher(CloseTrainsMatchSet *results) {
2    CTVars vars = {t=NULL, ot=NULL, s=NULL, m=NULL, e=NULL}
3    int match_cntr = 0;
4    // Constraint: ∃t : Train(t)
5    int loop_bound0 = model->train_count;
6    for (int i0 = 0; i0 < loop_bound0; i0++) {
7      vars->t = model->trains[i0];
8      // Constraint: Location(t,s)
9      vars->s = vars->t->location;
10     if (vars->s != NULL) {
11       // Constraint: ∃m : ConnectedTo(s,m)
12       int loop_bound1 = vars->s->connected_to_count;
13       for (int i1 = 0; i1 < loop_bound1; i1++) {
14         vars->m = vars->s->connected_to[i1];
15         // Constraint: ConnectedTo(m,e)
16         int loop_bound2 = vars->m->connected_to_count;
17         for (int i2 = 0; i2 < loop_bound2; i2++) {
18           vars->e = vars->m->connected_to[i2];
19           // Constraint: ¬(s = t)
20           if (vars->s != vars->e) {
21             // Constraint: ∃ot : OccupiedBy(e,ot)
22             vars->ot = vars->e->train;
23             if (vars->ot != NULL) {
24               // Register match
25               results->matches[match_cntr].s = vars->s;
26               results->matches[match_cntr++].e = vars->e;
27   } } } } } }
28   results->size = match_cntr; }
```

Listing 6.4: Source code generated for query closeTrains

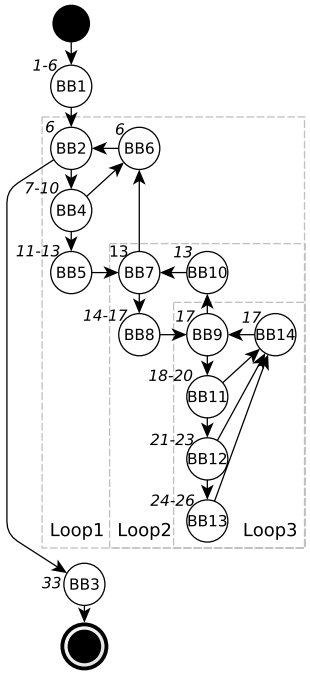


Figure 6.4: CFG of example query program

> If such a train exists, a match is registered by assigning the corresponding variable values to parameter variables in a new match (lines 24–26) and incrementing the matches found counter `match_cntr`. The execution concludes with saving the number of matches (line 28). Static analysis of the query code itself in Listing 6.4 would not impose any restrictions on line 20 despite the fact that the domain-specific constrains prescribe connectedTo links to be symmetrical. Therefore, at least every other execution of line 20 will jump back to line 17 instead of proceeding to line 22, yielding a flow constraint that is not discoverable by analysis of the code only.

Cyclomatic complexity (CC) is frequently used as a metric in safety-critical software to estimate code complexity [Rie17]. As a general recommendation, code with high CC is traditionally avoided in a safety-critical system as it requires extra efforts to test and maintain. However, the derived imperative source code of data-driven monitoring programs is inherently complex even for small queries, which is largely attributed to the declarative nature of query specifications. For example, the CC of Listing 6.4 is 7, which already indicates substantial complexity.

While modern WCET analyzers can analyze complex code fragments, they heavily rely on manual annotation of the code (loop bounds, in particular) and design time information about variable values to be able to come up with an estimate that is both safe and tight. A key contribution of the current paper is to complement the existing WCET analysis by providing means to automatically exploit domain-specific restrictions of input data and tighten the resulting WCET estimate. For data-driven monitors, this is a key step in order to enable their use in a safety-critical context.

6.2 Formal background

This section provides the formal background for the static analysis of data-driven runtime monitors, introduces definitions from traditional IPET-based approaches for WCET estimation,

and revisits the state of the art of domain-specific graph modeling and graph model generation.

6.2.1 Implicit path enumeration technique for estimating WCET

Timing analysis frequently relies on the Implicit Path Enumeration Technique (IPET) [LM97] that uses the control flow graph (CFG) of a program to estimate the WCET. This section revisits definitions from [PS97; KKZ13] to introduce this classic WCET estimation approach.

Definition 6.1 A *program* is a pair $\langle BB, Loops \rangle$, where BB is the set of *basic blocks* and $Loops$ is the set of (well-structured) loops. Each loop $\ell \in Loop$ has a *header* $bb_{\ell,h} \in \ell$, while the rest of its blocks $bb_i \in \ell$ constitute its *body*.

Definition 6.2 A *weighted control flow graph* corresponding to a program $\langle BB, Loops \rangle$ is a tuple $CFG = \langle V, E, s, t, w, tr \rangle$, where

- V is a finite set of *nodes*;
- $E \subseteq V \times V$ is the set of *edges*;
- s and $t \in V$ are the *program start* and *end* nodes, respectively;
- $w \colon E \to \mathbb{N}$ is the *weight function* that assigns execution times to the edges;
- $tr \colon V \to BB$ is the *traceability* function that maps nodes to originating program blocks.

In the simplest case, $V = BB$ and tr is the identity function. However, even on simple embedded processors, basic block execution times may vary due to microarchitectural effects (e.g., pipeline or cache state), which necessitates representing basic blocks with multiple nodes to encode context-sensitive execution times. The extended CFG encodes information from a *low-level timing analysis*. For example, the VIVU approach [Mar+98] addresses this concern by *virtual loop unrolling* and creates additional nodes and edges in the CFG to explicitly model the first executions of loops.

Definition 6.3 Every execution of the program (i.e., program trace) can be represented by a path $\pi = \langle e_1, \ldots, e_m \rangle$ in the CFG from s to t. Let $\pi\#e$ denote the frequency of the edge e appearing in π. The program execution time $\tau(\pi)$ can be estimated by the sum of the product of weights and frequencies of edges, i.e., $\tau(\pi) = \sum_{e \in E} w(e) \cdot \pi\#e$.

The goal of timing analysis is to find the path with the longest possible execution time of the program. Instead of explicitly enumerating all possible paths in the CFG, Puschner and Schedl [LM97] create a system of linear inequalities whose solutions over-approximate the set of possible paths and define an integer linear programming (ILP) problem for estimating WCET.

An ILP can be derived from the CFG $\langle V, E, s, t, tr \rangle$ as follows (we will use the notations from Section 2.3.1 for representing ILPs). Let $\mathfrak{f} \colon E \to \mathcal{X}$ be a function that associates variable symbols to CFG edges. In the system of linear equations $\mathcal{S}$, each feasible execution path P is associated with a solution k_π such that $k_\pi(\mathfrak{f}(e)) = \pi\#e$. Thus, linear constraints on variable obtained as $\mathfrak{f}(e)$ restrict frequency of e in all feasible paths. Moreover, $\sum_{e=\langle n_1,n_2 \rangle \in E, tr(n_1)=bb_i} k_\pi(\mathfrak{f}(e))$ is the number of times the basic block $bb_i \in BB$ was executed in π.

- We add $\sum_{e=\langle s,n \rangle \in E} \mathfrak{f}(e) = 1$ and $\sum_{e=\langle n,t \rangle \in E} \mathfrak{f}(e) = 1$ to $\mathcal{S}$, because the program is entered and exited exactly once.
- Except for s and t, each node is entered and exited the same number of times, so we add $\sum_{e=\langle n_1,n_2 \rangle \in E} \mathfrak{f}(e) - \sum_{e=\langle n_2,n_3 \rangle \in E} \mathfrak{f}(e) = 0$ for each $n_2 \in V \setminus \{s, t\}$.
- Any additional *flow facts* regarding the execution frequencies of program parts (e.g., loop execution counts) are added in the form $\sum_{e_{i,j} \in E} a_{i,j} \cdot \mathfrak{f}(e_{i,j}) \leq y_j$.
- For each edge $e \in E$, we also have $-\mathfrak{f}(e) \leq 0$, since the execution frequency is non-negative.

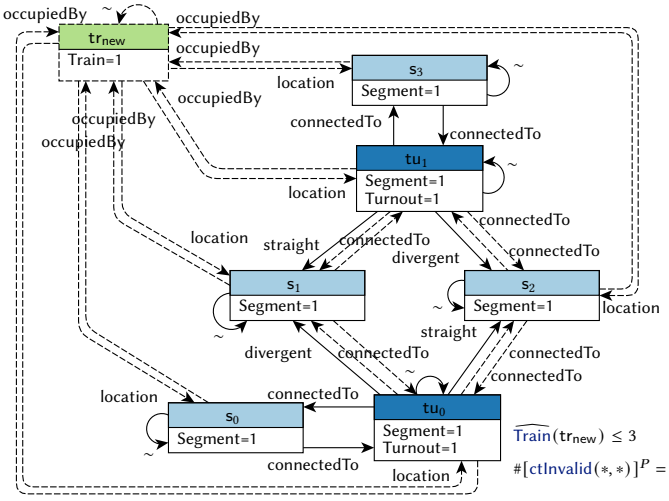


Figure 6.5: Example partial model

We have an ILP $\max \sum_{e \in E} w(e) \cdot \mathfrak{f}(e)$ *subject to* $\mathcal{S}$. The objective function $g(k)$ overapproximates the execution time of P. Therefore the value $g(k^*)$ of any solution k^* is a WCET bound.

6.2.2 Modeling and graph queries

In order to extend static analysis of data-driven graph query programs with domain-specific flow information, we will formally capture metamodels by a logic signature and their instance models as logic structures following [SNV18][j1].

We rely on *scoped partial models* from Section 2.3 to represent runtime models. In particular, runtime snapshots of a system are captured by *concrete (instance) models*, which contain no uncertainty or multi-objects and truth values are restricted to 1 and 0. During the reasoning about the WCET of graph query programs, scoped partial models that are not concrete

A metamodel may also include binary *attribute symbols* such as in [Búr+18]. However, their handling is analogous to binary relation symbols, thus their discussion is excluded from here.

> **Example 6.7** For the metamodel of Figure 6.2a, Train, Segment, Turnout $\in \Sigma_C$ are unary class predicates, and location, occupiedBy, connectedTo, straight, divergent $\in \Sigma_R$ are binary relation predicates. Out of these, Train is numerically tracked, i.e., Train $\in \Gamma_C$. We also have a binary predicate symbol ctInvalid $\in \Gamma_F \subseteq \Sigma_F$ (α(ctInvalid) = 2) that is numerically tracked.
> Figure 6.5 shows a partial model $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ conforming to the MoDeS3 metamodel. Objects are drawn as boxes with the values of the interpretations $\mathcal{I}_P(\mathsf{C}_i)$ of the class symbols written inside, while edges are drawn as arrows labelled with the relation symbols R_j and the equality symbol ∼. Solid edges correspond to 1 logic values, dashed edges are ½ logic values, and 0 logic values are omitted. Uncertain existence ε is shown with a dashed outline.
> The switching direction of the turnouts tu_0 and tu_1 is unknown. Additionally, there are no concrete trains on the track, but a *multi-object* $\mathsf{tr}_{\mathsf{new}}$ represents all trains and their potential locations. Formally, $\mathcal{I}_P(\varepsilon)(\mathsf{s}_0) = \mathcal{I}_P(\sim)(\mathsf{s}_0, \mathsf{s}_0) = \mathcal{I}_P(\mathsf{Segment})(\mathsf{s}_0) = \mathcal{I}_P(\mathsf{connectedTo})(\mathsf{s}_0, \mathsf{tu}_0) = 1$, but $\mathcal{I}_P(\mathsf{connectedTo})(\mathsf{tu}_0, \mathsf{s}_1) = ½$. Because $\mathcal{I}_P(\varepsilon)(\mathsf{tr}_{\mathsf{new}}) = \mathcal{I}_P(\varepsilon)(\mathsf{tr}_{\mathsf{new}}, \mathsf{tr}_{\mathsf{new}}) = ½$, $\mathsf{tr}_{\mathsf{new}}$ is a *multi-object* that may stand for any number of Train instances (even 0). The location of $\mathsf{tr}_{\mathsf{new}}$ is also uncertain, new trains may be located on any Segment or Turnout.
> The system of linear inequalities $\mathcal{S}_P = \{\widehat{\mathsf{Train}}(\mathsf{tr}_{\mathsf{new}}) \leq 3, \#[\mathsf{ctInvalid}(*, *)]^P = 0\}$ is shown in the lower right corner of the figure.

Concrete models are obtained from partial models by a series of *refinements* [SFC12], which add further information by setting unknown logic values ½ to 1 or 0, while known 1 and 0 values

remain unchanged. This is captured by the refinement relation $X \succcurlyeq Y \coloneqq (X = ½) \vee (X = Y)$. During model generation, refinements are carried out until a concrete model is reached.

Example 6.8 The concrete model M in Figure 6.2b is a refinement of the partial model P in Figure 6.5: $P \succcurlyeq_{abs} M$. The abstraction function maps $abs(\mathrm{tr}_0) = \mathrm{tr}_{\mathrm{new}}$ and $abs(\mathrm{tr}_1) = \mathrm{tr}_{\mathrm{new}}$, i.e., newly added trains are refinements of the train multi-object. Any other object is mapped by abs to itself, i.e., the identities of the rest of the objects remained unchained.
We may also see that the directions the turnouts were set, e.g., $\mathcal{I}_P(\mathsf{connectedTo})(\mathrm{tu}_0, \mathrm{s}_1) = ½ \succcurlyeq \mathcal{I}_M(\mathsf{connectedTo})(\mathrm{tu}_0, \mathrm{s}_1) = 0$. Moreover, $\mathcal{S}_M \vDash \mathcal{S}_P$.

The formal definitions of metamodel and instance model enable the formulation of first-order logic (FOL) predicates, which can be evaluated as *graph queries* over the logic structure of an instance model (Definition 2.29).

Note that in our context, a match of a query will typically represent a violation of a well-formedness constraint of the domain or a hazardous situation with respect to a safety property.

Definition 6.4 The *match set* of a query predicate φ with free variables $v_1, \ldots, v_n$ is the set $Matches(M, \varphi) = \{Z \colon \{v_1, \ldots, v_n\} \to O_M \mid [\![\varphi]\!]^M_Z = 1\}$. One element in this set is called a *match*, while $M\#\varphi = |Matches(M, \varphi)|$ denotes the size of the match set.

Note that if $M = \langle O_M, \mathcal{I}_M, \mathcal{S}_M \rangle$ is a concrete model over the signature $\langle \Sigma, \Gamma, \alpha \rangle$, and M compatible with the theory $\mathcal{T} = \langle d, \mathcal{E} \rangle$, then

- if $\varphi \in \mathcal{E}$, then $M\#\varphi = 0$; and
- if $\mathsf{F} \in \Gamma$ and $d(\mathsf{F}) = \varphi$, then $\mathcal{S}_M \vDash \#[\mathsf{F}(*, \ldots, *)]^M = M\#\varphi$.

Example 6.9 Consider the graph query φ_{CT} for the "close trains" hazard formalized as a FOL expression φ_{CT} in Figure 6.3c. In the concrete model M in Figure 6.2b, $[\![\varphi_{\mathrm{CT}}]\!]^M_{s \mapsto \mathrm{s}_2, e \mapsto \mathrm{s}_3} = 1$. In the partial model P in Figure 6.5, $[\![\varphi_{\mathrm{CT}}]\!]^P_{s \mapsto \mathrm{s}_2, e \mapsto \mathrm{s}_3} = 0$ due to the uncertain existence of the $\mathrm{tr}_{\mathrm{new}}$ multi-object. In M, φ_{CT} has two matches $Matches(M, \varphi_{\mathrm{CT}}) = \{\{s \mapsto \mathrm{s}_2, e \mapsto \mathrm{s}_3\}, \{s \mapsto \mathrm{s}_2, e \mapsto \mathrm{s}_3\}\}$. Therefore, $M\#\varphi_{\mathrm{CT}} = 2$.

6.2.3 Well-formedness and scope constraints

In order to make static analysis of data-driven monitors more precise, we can add additional domain-specific information into models as constraints to exclude impossible or irrelevant runtime snapshots from consideration. To this end, we will use *theories* of predicate symbol definitions and error patterns as defined in Definition 2.31.

Additionally, numerical *scope constraints* restrict the sizes of models to conform with allocation requirement in monitor programs and guide the analysis toward models that are relevant in practice (e.g., the size of the model and the ratios between the number of objects of given types match realistic scenarios).

Example 6.10 Consider the FOL predicate

$$\varphi_{\mathrm{ctInvalid}}(v_1, v_2) = \mathsf{connectedTo}(v_1, v_2) \wedge \neg \mathsf{connectedTo}(v_2, v_1)$$

and the theory $\mathcal{T} = \langle d, \emptyset \rangle$, where $d(\mathsf{ctInvalid}) = \varphi_{\mathrm{ctInvalid}}$.
The predicate $\varphi_{\mathrm{ctInvalid}}$ selects connectedTo links that do not have a corresponding link in the reverse direction. Since railway tracks can be traversed in both directions, we enforce a symmetric ctInvalid relation by a well-formedness constraint encoded as $(\widetilde{\mathsf{ctInvalid}} = 0) \in \mathcal{S}_P$. The concrete model M in Figure 6.2b is compatible with the theory $\mathcal{T}$. As shown in Lemma 2.30, M obeys the scope and well-formedness constraints prescribed in $\mathcal{S}_P$, since $P \succcurlyeq M$.

Our work relies on the model generator presented in [SNV18] and Chapter 4, which was proved to be complete and sound in [Var+18]. Informally, it is able to derive all concrete (instance)

Table 6.2: WCET analysis approaches for data-driven runtime monitor programs

	Inputs	Outputs
	Low-level analysis inputs	
CL	HW description + Query program	WCET estimate
	Value analysis inputs	
VAL	HW description + Query program + Concrete model + Memory image	WCET estimate for single memory image
	Domain-specific high-level analyis inputs	
DS_M	HW description + Query program + Concrete model	WCET est. for single model
DS_Σ	HW description + Query program + **Constraints**	WCET est. for all valid models + **Witness model**
DS_P	HW description + Query program + **Partial model** + **Constraints**	WCET est. for all refinements + **Witness model**

models in a domain (up to a designated size defined by the scopes) which satisfy the constraints by exploring a *state space* of possible partial models along refinements.

6.3 Timing analysis of query-based monitors

Estimating the WCET of query-based monitors is a highly complex task which involves multiple classic challenges of timing analysis. The runtime model of the system is a continuously changing data structure that captures an up to date snapshot of the underlying running system. Hence, it is not sufficient to analyze execution time on a single input model, but all models possible at runtime must be considered.

However, the space of possible models is enormous. For example, in a metamodel with 3 reference types, there may be up to $2^{3 \cdot 25 \cdot 25} = 2^{1875}$ models with 25 objects. Thus, explicit enumeration of graph models is intractable, which necessitates the use of abstractions.

Another major challenge is that *query execution time is heavily data-dependent*, i.e., the same control flow of a query program may have substantially different run times based upon the structural characteristics of the underlying graph model. Assuming some constraints on model size (e.g., capped by available memory) and some general restrictions on model scope (e.g., there are more segments than trains in any real model), a key open challenge is *how to provide a model where the execution time of a particular query program is maximal.* In this work, we provide *witness models* that maximize an estimate of the execution time, which aids in WCET analysis and in identifying bottlenecks in query execution.

Moreover, a single model may be represented in memory in several isomorphic ways (Section 6.1.2.2). During the runtime evolution of the graph, a particular snapshot might be reached in any of its possible in-memory representations. Thus, WCET estimation even for a single concrete input model must tackle the dependency of execution paths on the data representations. As a single model of n objects has $n!$ possible in-memory representations even if we only consider inserting the objects into a single continuous linear array of n elements, explicit enumeration is again intractable.

6.3.1 Comparison of timing analysis approaches

Table 6.2 illustrates the existing and proposed approaches of WCET analysis for query programs. *Classical* (CL) analysis is based on binary code of query program and the characteristics of the hardware platform, but does not consider structure and the well-formedness of the models.

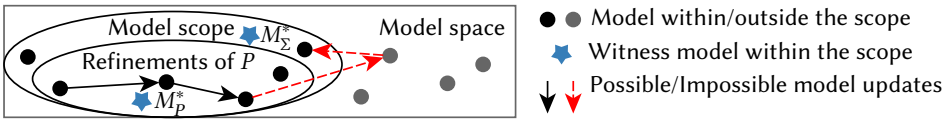


Figure 6.6: Classification of query input models and model updates from the perspective of WCET analysis

Value analysis (VAL) can derive more precise WCET estimates for executing a query on a single memory image (comprised of a single concrete model). However, it is unable to consider equivalent in-memory representations of the same concrete model (i.e., different parts of the model allocated to different spatial locations), or to cover all possible consistent concrete models, thus it is unsuitable for the analysis of data-driven monitors.

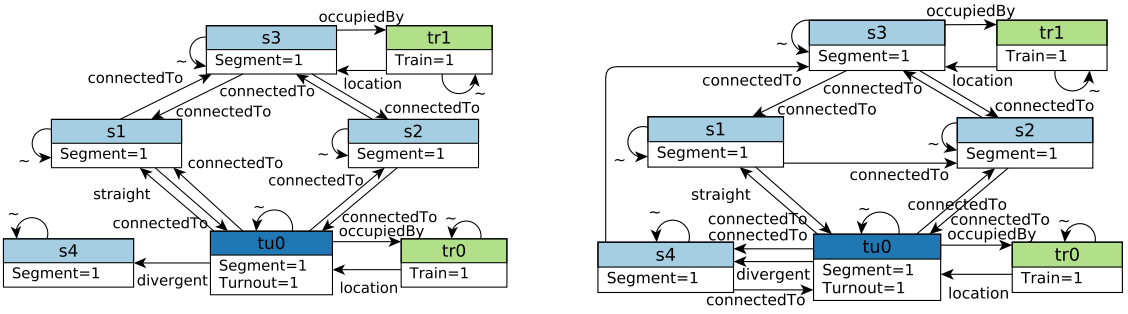
To alleviate this issue, we propose three *domain-specific* (DS) WCET analysis methods for data-driven monitors. We introduce the concept of *witness models*, which are consistent models that are *feasible inputs of the graph query program* and *maximize the WCET estimate for all models within the given scope*. They serve as representative data to calculate WCET for *any* model within the scope.

- First, we estimate WCET for a *single concrete model* (DS_M). The estimate is valid for all in-memory representations of a given concrete model M.
- In the next case, the set of possible runtime snapshots is specified with *metamodel* $\langle \Sigma, \alpha \rangle$ along with well-formedness and scope constraints (DS_Σ). This WCET estimate is valid for all possible runtime snapshots within the memory limits of the system, i.e., for all consistent instances of the metamodel up to the size specified by the scope constraints.
- Thirdly, an *initial partial model* P may specify the set of possible runtime snapshots (DS_P) including static (known and concrete) and dynamic (uncertain at design time) parts of the runtime model. The WCET estimate is valid for all possible refinements $P \succcurlyeq M$ of P.

Figure 6.6 sketches the *model space* of runtime graph models (represented with dots), i.e., the set of all input models. Possible changes made to a model at runtime (depicted as arrows) result in a new model. To obtain a safe and tight WCET estimate for query programs, we make some assumptions about realistic (and consistent) models captured in the form of a *model scope*. If an initial partial model P is provided, the analysis is further restricted to its (valid) refinements, thus inconsistent models are considered to be unrealistic. The witness model M^*_Σ for the consistent instances of the metamodel and M^*_P for the refinements of P are depicted as blue stars in Figure 6.6.

The witness models M^*_P may aid in iteratively refining the model scope. If the partial model corresponds to a situation that is impossible at runtime, it indicates that the model scope was specified in a too general way. We may exclude such situations by refining the partial model P [Var+18]. However, care must be taken to avoid excluding feasible inputs and overfitting the WCET estimate. If the witness model is a feasible input, it may be inspected to study the characteristics and bottlenecks of the graph query program.

> **Example 6.11** Figure 6.7a shows the witness model M^* for the WCET of the closeTrains query for well-formed models with up to 7 objects in total (as model scope), out of which up to 2 are Train instances. The corresponding WCET estimate is 1309 systicks. The model M' in Figure 6.7b has the same number of elements, but with a higher execution time estimate of 1325 systicks. However, M' lies outside the model scope, because it is malformed due to non-symmetric connectedTo references (e.g., s1 is connectedTo s2 but not vice versa).
> Classical (CL) WCET estimation techniques cannot exclude M' from the analysis, and they



(a) Witness model M^* yields execution estimate of 1309 for query closeTrains

(b) Malformed model M' yields execution estimate of 1325 for query closeTrains

Figure 6.7: Illustrating model generation problems for witness models

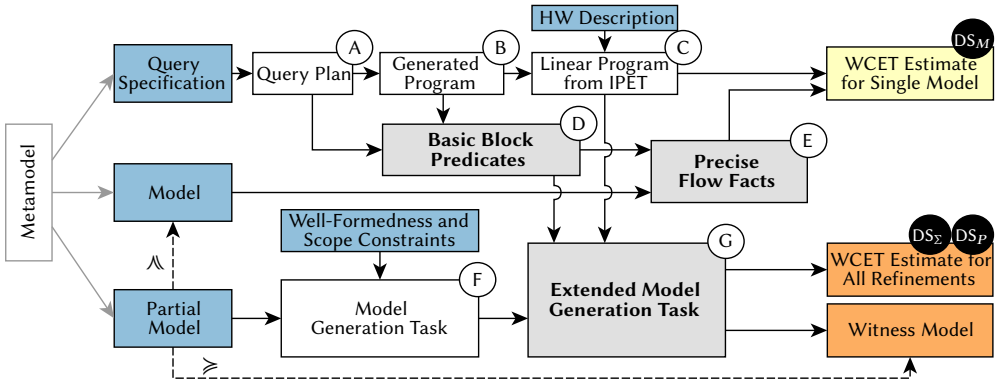


Figure 6.8: Workflow of WCET estimation for query-based monitors

would return a higher WCET estimate, while our novel DS_Σ technique can restrict the analysis to the model scope to return the correct estimate of 1309 systicks along with the witness M^*.

6.3.2 Architectural overview

Figure 6.8 presents the high-level description of our design time tasks to obtain a WCET estimate in the DS_M, DS_Σ and DS_P scenarios. The high-level inputs of the process include the *query specification*, the target *hardware description*, and the *well-formedness and scope constraints* of the domain.

First, a *query plan* (A) is constructed from the query specification, based on which the *query program* (B) is generated according to Section 6.1.3. Our approach is complementary to IPET-based WCET estimators and leverages the results of high- and low-level analysis in form of the CFG (possibly after some loop unrolling) and the corresponding linear program (C).

In case of WCET estimation for a concrete model M (DS_M), M is also provided as an input. Based on the query plan and the generated monitor code, *basic block predicates* (D) are derived, whose matches in the concrete model M correspond to executions of basic blocks in the monitor program. We leverage these matches to construct *precise flow facts* (E) for IPET analysis in Section 6.3.3. The resulting flow facts and WCET estimate consider all possible in-memory representations of M.

For WCET estimation for any valid instance of a metamodel $\langle \Sigma, \Gamma, \alpha \rangle$ (DS_Σ), the initial partial model P_{init} is constructed according to Section 4.2.2. When a partial model P is already provided as input (DS_P), it replaces P_{init} as the initial partial model. Hence, along with the $\mathcal{T}$ of well-formedness and scope constraints, we obtain a *model generation task* (F), whose solutions are

Algorithm 2: Basic block predicate construction

```
1  Function DerivePredicates(BB, lineTrace, planTrace) is
2      Ψ = ∅
3      for bb ∈ BB do
4          ψ_bb = 1
5          Let ln_b–ln_e be the source lines associated with bb in
             lineTrace
6          for if and for statements st containing ln_b–ln_e do
7              ψ_bb = ψ_bb ∧ StatementToLogic(st, planTrace)
8          Let v_1, ..., v_m be the free variables of ψ_bb
9          Ψ = Ψ ∪ {ψ_bb}
10         if bb is the header of the loop ℓ then
11             ψ'_bb = ψ_bb ∧ StatementToLogic(ℓ, planTrace)
12             Let v_1, ..., v_{m+1} be the free variables of ψ'_bb
13             Ψ = Ψ ∪ {ψ'_bb}
14     return Ψ
```

Algorithm 3: Translate search plan to logic

```
1  Function StatementToLogic(st, planTrace) is
2      Determine the constraint implemented by st from
         planTrace
3      if st implements extend ∃v_i : C(v_i) then
4          return C(v_i)
5      else if st implements extend ∃v_j : R(v_i, v_j) then
6          return R(v_i, v_j)
7      else if st implements check C(v_i) then
8          return C(v_i)
9      else if st implements check R(v_i, v_j) then
10         return R(v_i, v_j)
11     else if st implements check v_i = v_j then
12         return v_i = v_j
13     else if st implements check ¬C(v_i) then
14         return ¬C(v_i)
15     else if st implements check ¬R(v_i, v_j) then
16         return ¬R(v_i, v_j)
17     else if st implements check ¬(v_i = v_j) then
18         return ¬(v_i = v_j)
```

Algorithm 4: Precise flow fact construction for a single concrete model M

```
1  Function PreciseFlowFacts(BB, lineTrace, planTrace, CFG = ⟨V, E, s, t, w, tr⟩, f, M) is
2      S_flow = ∅, Ψ = DerivePredicates(BB, lineTrace, planTrace)
3      for bb ∈ BB do
4          if bb is a loop header then S_flow = S_flow ∪ {Σ_{e=⟨n1,n2⟩∈E, tr(n1)=bb} f(e) = M#ψ_bb + M#ψ'_bb} ;
5          else S_flow = S_flow ∪ {Σ_{e=⟨n1,n2⟩∈E, tr(n1)=bb} f(e) = M#ψ_bb} ;
6      return S_flow
```

partial models within the analyzed model scope. We incorporate the basic block predicates (D) into an extended signature $\langle \Sigma', \Gamma', \alpha' \rangle$ and an extended theory $\mathcal{T}'$ in Section 6.3.4, which forms an *extended model generation task* (G) for witness model generation along with the linear program (C). *Witness models* are systematically generated using a graph solver [SNV18][j1] as solutions of such tasks along refinements $P_{\text{init}} \geqslant M$ of the initial partial model. The cost associated with the witness model M^*, which is a solution of the IPET linear program (C) extended with domain-specific flow facts (E), is a safe and tight WCET estimate.

6.3.3 Approximating execution time with graph predicates

To derive precise flow facts for WCET analysis of a graph query program with concrete input model M and characterize its data-dependent execution time, we construct a *basic block predicate* ψ_{bb} for each basic block $bb \in BB$ of the query program. Free variables of ψ_{bb} correspond to program variables (bound by `for` loops). Due to the structure of the code generated from the query plans (Section 6.1.3.3), each execution of bb corresponds to a match $Z \in \mathit{Matches}(M, \psi_{bb})$ of ψ_{bb} in M.

For loop headers, we construct an additional $\psi'_{bb_{\ell,h}}$ where the matches of $\psi'_{bb_{\ell,h}}$ represent executions of the loop ℓ where the loop condition holds, while the matches of $\psi_{bb_{\ell,h}}$ correspond to the executions where the loop exits.

Algorithm 2 takes a data-driven monitor program generated from a graph query and constructs the basic block predicates. In addition to the set of basic blocks *BB*, the algorithm requires traceability information *lineTrace* (that connects basic blocks to source code lines) and *planTrace* (that connects source code lines to **extend** and **check** constraints in the query plan). The *lineTrace* is extracted from the IPET analysis tool (based on debug information in the compiled executable), while *planTrace* is the output of the query code generator.

As state-of-the-art WCET analysis tools [Bal+10] do not recommend analyzing programs compiled with advanced optimizations, we did not assess programs that use optimization. Therefore, source line, as well as **extend** and **check** constraint information in *lineTrace* remains valid after compilation. However, the following algorithms can be extended to support compiler optimizations, as long as a compiled basic block still corresponds to a single constraint and the control flow remains structured (comprised on loops and conditionals).

In line 4, ψ_{bb} is initialized to true, which has a single (trivial) match in any model to reflect that blocks not implementing any query plan constraints will be executed exactly once. Then, in line 5, we traverse *lineTrace* to extract the source lines corresponding to *bb*. The loop in lines 6–7 processes all `if` and `for` statements enclosing the source lines for *bb*. As a result, ψ_{bb} becomes the conjunction of atomic predicates, which correspond to the query plan constraints implemented by the processed statements. Lastly, in lines 10–13, if *bb* is a loop header, we also add the atomic predicate corresponding to the loop itself to obtain ψ'_{bb}, which characterizes executions when the loop condition holds.

Algorithm 3 implements translation of `for` and `if` statements to atomic logical predicates. The algorithm traverses *planTrace* to process the corresponding query plan constraint. For **extend** constraints (usually associated with `for` loops), the existential quantifier $\exists$ is removed so that the constraint introduces a new free variable to ψ_{bb}. **Check** constraints (associated with `if` statements) are returned as-is, because all their variables are already introduced by some enclosing **extend** operation. Thus, for a basic block *bb* enclosed by m statements with **extend** constraints will have ψ_{bb} with free variables $v_1, \ldots, v_m$. If *bb* is a loop header, ψ'_{bb} has an additional v_{m+1} free variable.

> **Example 6.12** Listing 6.4 shows the generated source code of the `closeTrains` graph query, while Figure 6.4 show the corresponding CFG (without any loop unrolling). The lineTrace information is depicted as line numbers next to the CFG nodes, and comments above the control structures contain planTrace. The basic block bb_9 corresponds to the loop header in line 17. Collecting the query plan constraints from the control structures enclosing line 17 with Algorithm 2, we find that
>
> $$\psi_{bb_9} = \mathsf{Train}(t) \wedge \mathsf{location}(t, s) \wedge \mathsf{connectedTo}(s, m)$$
> $$\psi'_{bb_9} = \mathsf{Train}(t) \wedge \mathsf{location}(t, s) \wedge \mathsf{connectedTo}(s, m) \wedge \mathsf{connectedTo}(m, e).$$
>
> In the concrete model M^* in Figure 6.7a, executions of bb_9 are represented by the 4 matches $\mathit{Matches}(M, \psi_{bb_9}) = \{\{t \mapsto \mathrm{tr0}, s \mapsto \mathrm{tu0}, m \mapsto \mathrm{s1}\}, \{t \mapsto \mathrm{tr0}, s \mapsto \mathrm{tu0}, m \mapsto \mathrm{s2}\}, \{t \mapsto \mathrm{tr1}, s \mapsto \mathrm{s3}, m \mapsto \mathrm{s1}\}, \{t \mapsto \mathrm{tr0}, s \mapsto \mathrm{s3}, m \mapsto \mathrm{s2}\}\}$ of ψ_{bb_9}, as well as the 8 matches of ψ'_{bb_9} obtained by extending each ψ_{bb_9} match by the two possible segments connectedTo the value of m as the value of the variable e. Each match describes the values of the program variables when entering bb_9.

Algorithm 4 constructs precise domain-specific flow facts for a concrete model M. In addition to the basic blocks *BB* and the traceability information, the algorithm reads the control flow graph $\mathit{CFG} = \langle V, E, s, t, w, tr \rangle$ and the function $\mathfrak{f}\colon E \to \mathcal{X}$ associating CFG edges with linear equation variables. Line 2 initializes the empty system of linear equations $\mathcal{S}_{\text{flow}}$ and constructs the basic block predicates Ψ. Leveraging the CFG traceability function $tr\colon E \to BB$, expressions $\sum_{e=\langle n_1,n_2\rangle \in E, tr(n_1)=bb} \mathfrak{f}(e)$ are built, which represent the number of times a basic block *bb* is executed (Section 6.2.1). For a loop header, this number is equal to the number of ψ_{bb} and ψ'_{bb} matches in M (line 4), while for other blocks, only matches of ψ_{bb} are counted (line 5).

The resulting set of linear equations $\mathcal{S}_{\text{flow}}$ serve as flow facts in IPET analysis. More precisely, by incorporating $\mathcal{S}_{\text{flow}}$ into the analysis, we may obtain a safe and tight estimate for the execution time of a graph query program on the concrete model M.

Algorithm 5: Witness generation task construction for a partial model P

1 **Function** WitnessGen(BB, *lineTrace*, *planTrace*, *CFG* $= \langle V, E, s, t, w, tr\rangle, \langle \Sigma, \Gamma, \alpha\rangle, P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P\rangle, \mathcal{T} = \langle d, \mathcal{E}\rangle$) **is**
2 Construct the IPET max $\sum_{\mathfrak{x}_i \in \mathcal{X}_{\text{IPET}}} c_i \cdot \mathfrak{x}_i$ *subject to* $\mathcal{S}_{\text{IPET}}$ for *CFG* with $\mathfrak{f}\colon E \to \mathcal{X}_{\text{IPET}}$ (w.l.o.g. $\mathcal{X}_{\text{IPET}} \cap \mathcal{X}_P = \emptyset$);
3 $\Sigma' = \Sigma, \Gamma' = \Gamma, \alpha' = \alpha, d' = d, \mathcal{I}_{P'} = \mathcal{I}_P, \mathcal{S}_{P'} = \mathcal{S}_P, \Psi = \text{DerivePredicates}(BB, \textit{lineTrace}, \textit{planTrace})$;
4 $P' = \langle O_P, \mathcal{I}_{P'}, \mathcal{S}_{P'}\rangle$ over the theory $\langle \Sigma', \Gamma', \alpha'\rangle, \mathcal{T}' = \langle d', \mathcal{E}\rangle$;
5 **foreach** $\mathfrak{x} \in \mathcal{X}_{\text{IPET}}$ **do** $\Gamma'_{\mathsf{X}} = \Gamma'_{\mathsf{X}} \cup \{\mathsf{X}_{\mathfrak{x}}\}, \alpha' = \alpha' \cup \{\mathsf{X}_{\mathfrak{x}} \mapsto 0\}$;
6 **foreach** $\left[\sum_{\mathfrak{x}_i \in \mathcal{X}_{\text{IPET}}} a_i \cdot \mathfrak{x}_i \leq b\right] \in \mathcal{S}_{\text{IPET}}$ **do** $\mathcal{S}_{P'} = \mathcal{S}_{P'} \cup \left\{\sum_{\mathfrak{x}_i \in \mathcal{X}_{\text{IPET}}} a_i \cdot \hat{\mathsf{X}}_{\mathfrak{x}_i} \leq b\right\}$;
7 $\Gamma'_{\mathsf{X}} = \Gamma'_{\mathsf{X}} \cup \{\mathsf{X}^*\}, \alpha' = \alpha' \cup \{\mathsf{X}^* \mapsto 0\}, \mathcal{S}_{P'} = \mathcal{S}_{P'} \cup \left\{\hat{\mathsf{X}}^* = \sum_{\mathfrak{x}_i \in \mathcal{X}_{\text{IPET}}} c_i \cdot \hat{\mathsf{X}}_{\mathfrak{x}_i}\right\}$;
8 **foreach** $bb \in BB$ **do**
9 **if** bb is a loop header **then**
10 $\Sigma'_{\mathsf{F}} = \Sigma'_{\mathsf{F}} \cup \{\mathsf{Y}_{bb}, \mathsf{Y}'_{bb}\}, \Gamma'_{\mathsf{F}} = \Gamma'_{\mathsf{F}} \cup \{\mathsf{Y}_{bb}, \mathsf{Y}'_{bb}\}, \alpha' = \alpha' \cup \{\mathsf{Y}_{bb} \mapsto |\text{Free}\,\psi_{bb}|, \mathsf{Y}'_{bb} \mapsto |\text{Free}\,\psi'_{bb}|\}$, $d' = d' \cup \{\mathsf{Y}_{bb} \mapsto \psi_{bb}, \mathsf{Y}'_{bb} \mapsto \psi'_{bb}\}, \mathcal{I}_{P'}(\mathsf{Y}_{bb}) \equiv \mathcal{I}_{P'}(\mathsf{Y}'_{bb}) \equiv \tfrac{1}{2}$;
11 $\mathcal{S}_{P'} = \mathcal{S}_{P'} \cup \left\{\sum_{e=\langle n_1, n_2\rangle \in E, tr(n_1) = bb} \hat{\mathsf{X}}_{\mathfrak{f}(e)} = \#[\mathsf{Y}_{bb}(*, \ldots, *)]^{P'} + \#[\mathsf{Y}'_{bb}(*, \ldots, *)]^{P'}\right\}$;
12 **else**
13 $\Sigma'_{\mathsf{F}} = \Sigma'_{\mathsf{F}} \cup \{\mathsf{Y}_{bb}\}, \Gamma'_{\mathsf{F}} = \Gamma'_{\mathsf{F}} \cup \{\mathsf{Y}_{bb}\}, \alpha' = \alpha' \cup \{\mathsf{Y}_{bb} \mapsto |\text{Free}\,\psi_{bb}|\}, d' = d' \cup \{\mathsf{Y}_{bb} \mapsto \psi_{bb}\}$;
14 $\mathcal{I}_{P'}(\mathsf{Y}_{bb}) \equiv \tfrac{1}{2}, \mathcal{S}_{P'} = \mathcal{S}_{P'} \cup \left\{\sum_{e=\langle n_1, n_2\rangle \in E, tr(n_1) = bb} \hat{\mathsf{X}}_{\mathfrak{f}(e)} = \#[\mathsf{Y}_{bb}(*, \ldots, *)]^{P'}\right\}$;
15 **return** max X^* *subject to* $\langle P', \mathcal{T}'\rangle$;

Proposition 6.5 Let τ be the execution time of the query q on the concrete model M,

$$CL = \max \sum_{\mathfrak{x}_i \in \mathcal{X}} c_i \cdot \mathfrak{x}_i \textit{ subject to } \mathcal{S}_{\text{IPET}}, \qquad DS_M = \max \sum_{\mathfrak{x}_i \in \mathcal{X}} c_i \cdot \mathfrak{x}_i \textit{ subject to } \mathcal{S}_{\text{IPET}} \cup \mathcal{S}_{\text{flow}},$$

where CL is the classical IPET estimate obtained from q, and DS_M is the domain-specific estimate with flow facts derived from M using Algorithm 4. Then $\tau \leq DS_M \leq CL$.

6.3.4 Witness Generation of Worst-Case Execution Time

To estimate the WCET of some query program q over a set of models, we specify the *model scope* of interest as the *solutions*$(P, \mathcal{T})$ of a model generation task. We construct an extended model generation problem max X^* *subject to* $\langle P', \mathcal{T}'\rangle$ in Algorithm 5, where P' extends P with the results of IPET analysis and $\mathcal{T}'$ incorporates the basic block predicates obtained from q. We use the notation Free ψ to denote the set of *free variables* of the logic formula ψ.

In line 2, the algorithm builds an IPET integer program max $\sum_{\mathfrak{x}_i \in \mathcal{X}_{\text{IPET}}} c_i \cdot \mathfrak{x}_i$ *subject to* $\mathcal{S}_{\text{IPET}}$ based on the provided *CFG*. Then, in line 3, we invoke Algorithm 2 to obtain the basic block predicates Ψ used to derive precise flow facts for graph models.

We create a new signature $\langle \Sigma', \Gamma', \alpha'\rangle$ that extends the signature $\langle \Sigma, \Gamma, \alpha\rangle$ by new numerically tracked symbols. In line 5, we add a new auxiliary variable symbol $\mathsf{X}_{\mathfrak{x}}$ for each linear inequality variable $\mathfrak{x} \in \mathcal{X}_{\text{IPET}}$ of the IPET problem $\mathcal{S}_{\text{IPET}}$. The auxiliary variable symbol X^* added in line 7 will hold the value of the cost function $\sum_{\mathfrak{x}_i \in \mathcal{X}_{\text{IPET}}} c_i \cdot \mathfrak{x}_i$ of the IPET problem. Moreover, we add predicate symbols Y_{bb} and Y'_{bb} for basic block and loop header predicates ψ_{bb} and ψ'_{bb} in lines 13 and 10, respectively. The extended theory $\mathcal{T} = \langle d', \mathcal{E}\rangle$ assigns these predicates to the corresponding predicate symbols.

In the extended partial model P', we initialize the interpretations of basic block and predicate symbols to have the truth value $\tfrac{1}{2}$ everywhere. This allows us to discover the values of these predicates in the future by refinement without reducing model scope initially by new constraints.

The extended system of linear inequalities $\mathcal{S}_{P'}$ incorporates the IPET linearinequalities $\mathcal{S}_{\text{IPET}}$ after replacing each linear inequality variable $\mathfrak{x}_i$ with the corresponding auxiliary symbol $\hat{\mathsf{X}}_{\mathfrak{x}_i}$ in line 6. Line 7 sets the value of X^* symbol to be equal to the IPET objective function. The precise flow facts from Algorithm 4 are added to $\mathcal{S}_{P'}$ in lines 11 and 14. Instead of referring to the number of matches $M\#\psi_{bb}$ and $M\#\psi'_{bb}$ in a concrete model directly, we replace then with the

count aggregations $\#[\mathsf{Y}_{bb}(*,\ldots,*)]^{P'}$ and $\#[\mathsf{Y}'_{bb}(*,\ldots,*)]^{P'}$, respectively, to refer to the *potential* number of matches in the partial model P' (and thus in any concrete refinement $P' \succcurlyeq M$).

The resulting extended model generation task provides a safe and tight estimate for the query program WCET with theory $\mathcal{T}$.

Proposition 6.6 (Safety and tightness) Let $\tau(M)$ be the execution time of a query program q on a concrete model M, P be regular scoped partial model, $\mathcal{T}$ be a theory, and

$$CL = \max \sum_{\mathfrak{x}_i \in \mathcal{X}} c_i \cdot \mathfrak{x}_i \textit{ subject to } \mathcal{S}_{\text{IPET}}, \qquad DS_P = \max \mathsf{X}^* \textit{ subject to } \langle P', \mathcal{T}' \rangle,$$

where CL is the classical IPET estimated obtained from q, and DS_P is the domain-specific estimate based on the extended graph generation problem form Algorithm 5. Then $\tau(M) \leq DS_P \leq CL$ for all $M \in \textit{solutions}(P, \mathcal{T})$.

Optimal solutions are *witness models*, which maximize the domain-specific of WCET for concrete refinements of the input partial model P compatible with the theory $\mathcal{T}$. As the witness model M^* is in the model scope, it is a feasible (as opposed to spurious) input of the query program.

Definition 6.7 (Projection of an extended partial model) Let $\langle \Sigma, \Gamma, \alpha \rangle$ be a signature, $\langle \Sigma', \Gamma', \alpha' \rangle$ be the corresponding extended signature output by Algorithm 5, and $P' = \langle \mathcal{O}_{P'}, \mathcal{I}_{P'}, \mathcal{S}_{P'} \rangle$ be a regular scoped partial model over $\langle \Sigma', \Gamma', \alpha' \rangle$. Then $\lfloor P' \rfloor = \langle \mathcal{O}_{P'}, \mathcal{I}_{\lfloor P' \rfloor}, \mathcal{S}_{\lfloor P' \rfloor} \rangle$ is the *projection of the extended partial model* P' back to $\langle \Sigma, \Gamma, \alpha \rangle$, which is formed by omitting the interpretations of any symbols $\Sigma' \setminus \Sigma$ from $\mathcal{I}_{P'}$ and eliminating any symbols in $\Gamma' \setminus \Gamma$ from $\mathcal{S}_{P'}$ via Fourier–Motzkin elimination [Sch98, pp. 155-157].

Note that in practice, linear inequalities mentioning numerically tracked symbols in $\Gamma' \setminus \Gamma$ in $\mathcal{S}_{P'}$ do not contain any other symbols from Γ, because no such inequalities are added by Algorithm 5 or in any other step of the model generation procedure. Therefore, these inequalities can be simply discarded instead of having to execute Fourier–Motzkin elimintation.

Proposition 6.8 (Witness model) Let $DS_M(M)$ be the domain-specific WCET estimate of a query program q obtained by Algorithm 4 for a concrete model M, DS_P be the domain-specific WCET estimate of q for a regular scoped partial model P and theory $\mathcal{T}$ by Algorithm 5, and M^* be the witness model for the WCET of q, i.e., the optimal solution of DS_P. Then $\lfloor M^* \rfloor \in \textit{solutions}(P, \mathcal{T})$ and $DS_M(M) \leq DS_M(\lfloor M^* \rfloor) = DS_P$ for all $M \in \textit{solutions}(P, \mathcal{T})$.

Moreover, refinements of the partial model $P \succcurlyeq Q$ may be used to tighten the WCET estimate by reducing the model scope under discussion.

Proposition 6.9 (Tightening by refinement) Let $DS_P(P, \mathcal{T})$ denote the domain-specific WCET estimate of a query program q for a regular scoped partial model P and theory $\mathcal{T}$ obtained by Algorithm 5 and $P \succcurlyeq Q$ for some regular scoped partial model Q. Then $DS_P(Q, \mathcal{T}) \leq DS_P(P, \mathcal{T})$. In particular, if $P = P_{init}$ is the initial partial model for a metamodel $\langle \Sigma, \Gamma, \alpha \rangle$ from Section 4.2.2, then we may see that the WCET estimate for any partial model conforming to the metamodel is at least as tight as the DS_Σ estimate for the metamodel.

Example 6.13 Figure 6.9 shows a simplified execution of the graph generator. Suppose that Algorithm 5 has output an extended graph generation task $\max \mathsf{X}^*$ *subject to* $\langle P_0, \mathcal{T}' \rangle$, where the objective function is defined as $\left[\hat{\mathsf{X}}^* = 250 \cdot \hat{\mathsf{X}}_1\right] \in \mathcal{S}_{P_0}$ and we have a flow fact $\left[\hat{\mathsf{X}}_1 = \#[\mathsf{Y}_{bb}(*,*)]^{P_0}\right] \in \mathcal{S}_{P'}$. Moreover, suppose that the extended theory $\mathcal{T}' = \langle d', \mathcal{E} \rangle$ has $d(\mathsf{Y}_{bb}) = \varphi_{\text{CT}}$ from Figure 6.3c. This amounts to a witness generation task where there is a single baseic block *bb* that takes 250 systicks and is entered once for every match of the φ_{CT}

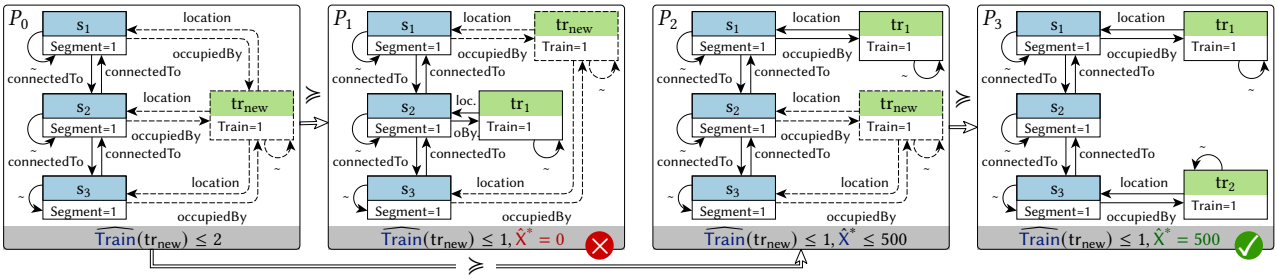


Figure 6.9: Execution of the example graph generation task max X^* *subject to* $\langle P', \mathcal{T}' \rangle$.

predicate. The inequality $\#[\text{Train}(*)]^{P_0} = \widehat{\text{Train}}(\text{tr}_{\text{new}}) \leq 2$ prescribes a *type scope* of at most 2 Train instances.

Two non-isomorphic partial models P_1, P_2 can be obtained from $P_0 = P'$. In P_1, the generator placed a train tr_1 on the middle segment s_2 of the track. Therefore, the multi-object tr_{new} represents at most one additional train ($\widehat{\text{Train}}(\text{tr}_{\text{new}}) \leq 1$). The φ_{CT} predicate cannot match ($\#[\text{Y}_{bb}(*,*)]^{P_1} = 0$), since it is impossible to place a new train on both s_1 and s_3. Any possible concrete refinement $P_1 \succcurlyeq M$ of P_1 has an objective value $\hat{X}^* = 250 \cdot \hat{X}_1 = 250 \cdot \#[\text{Y}_{bb}(*,*)]^M = 250 \cdot 0 = 0$.

If we place a new train on s_1, we obtain P_2. Placing a train on s_3 results in an isomorphic model, so it is sufficient to only consider P_2 instead. Here, there can be at most 2 matches of φ_{CT} ($\#[\text{Y}_{bb}(*,*)]^{P_2} = \hat{\text{Y}}_{bb}(\text{tr}_1, \text{tr}_{\text{new}}) + \hat{\text{Y}}_{bb}(\text{tr}_{\text{new}}, \text{tr}_1) \leq 2$). Indeed, if we place an additional train on s_3, we obtain the concrete model $M^* = P_3$ with 2 matches of φ_{CT} ($\#[\text{Y}_{bb}(*,*)]^{P_3} = \hat{\text{Y}}_{bb}(\text{tr}_1, \text{tr}_2) + \hat{\text{Y}}_{bb}(\text{tr}_{\text{new}}, \text{tr}_1) \leq 2$). The corresponding objective value is $\hat{X}^* = 250 \cdot 2 = 500$. No larger objective value is possible by any concrete refinement of P_1, since $\mathcal{S}_{P_1} \vDash \hat{X}^* = 0$. Hence we may discard P_1 along with its potential refinements, and output M^* as the witness model obtained as the optimal solution of model generation task.

Assuming that the (simplified) objective X^* is the domain-specific WCET estimate of the query program, 500 is our WCET estimate, which is the execution time bound that is expected to be reached (according to the low-level IPET analysis) when executing the query program over the witness model M^* as input.

6.4 Evaluation

We conducted experiments to address the following research questions related to the WCET of query programs. For each research question, we investigate the scenario (a) DS_Σ, where only the metamodel and the relevant well-formedness and scope constraints are known; and (b) DS_P, when an initial partial model (describing a track layout but not its runtime state) is also provided.

RQ1 How difficult is it to find witness models?

RQ2 How safe and tight are WCET estimates w.r.t. existing approaches and real execution times?

RQ3 How does query program complexity impact the overestimation of computed WCET bounds?

RQ1 aims at determining whether our model generation based approach can find the witness model in practical time and whether it constitutes an improvement over random search. The rest of the experiments study the quality of WCET bounds, which is a key factor in the applicability of our approach. In particular, RQ2 attempts to compare our computed WCET estimates with

the state of the art in challenging settings with partial runtime information, while RQ3 presents increasingly challenging query programs to our approach.

6.4.1 Evaluation overview and setup

Queries

To address these research questions, we use graph queries from the domain of the MoDeS3 CPS demonstrator [Vör+18b]. This demonstrator uses high-level runtime monitoring rules captured as graph queries, and showcases synthesized monitoring programs executing these queries over the runtime graph model of the underlying running system. Our experiments focus only on query evaluation, and updates to the runtime model are out of scope for the current paper. Therefore, we ran the query programs on various snapshots of runtime graph models. We evaluated the following queries adopted from [Búr+18]:

- **Close trains** (**ct**): This is the query introduced in the running example of Section 6.1.3.
- **End of siding** (**eos**): This query finds trains that are dangerously close (one segment distance) to an end of the track.
- **Misaligned turnout** (**mt**): Pairs of trains and turnouts are the objectives of this query, where the train would derail if it reached the turnout because it is switched in a different direction.
- **Train locations** (**tl**): A simple query to find pairs of trains and segments that describe the locations of each train.

The calculation of query search plans is out of scope of the current paper, but they were created and optimized based on the typical model statistics of runtime model snapshots in the MoDeS3 system. For example, the search plan presented in Table 6.1 is the one used by the program executing the query Close trains.

WCET algorithms and WCET tools

To compare the results produced by our WCET estimation approaches DS_Σ and DS_P with estimates produced by other tools, we used the commercial aiT [FH04] (version 20.10i) and the open-source OTAWA [Bal+10] (version V1.2.0) tools. For aiT, we used a *high precision* configuration with pipeline-level analysis and full (up to the determined loop bound) loop unrolling, as well as a *low precision* configuration with only basic block-level analysis and no unrolling. To incorporate the results of low-level analysis into DS_Σ and DS_P, we extracted the IPET linear equations from the low-level configuration of aiT manually, as no facility was available for automatic export or accessing the high precision system of linear equations directly. We also extracted the IPET linear equations from OTAWA, which have BB execution context information (paths of length two).

Graph models

In the following, we describe how we obtained a variety of models to assess the impact of models with different characteristics on query evaluation times.

Using the metamodel in the MoDeS3 case study, we generated *witness models* M^*_Σ for each monitoring query and for both low-level analyses (aiT, OTAWA) such that the query is estimated to have the longest possible execution time according the low-level analysis. For all of these models, we used the same model scope inspired by the railway domain: up to 20% of the objects

can be Trains and up to 20% of the objects can be Turnouts. The rest of the objects are Segments; we capped the maximum number of objects at 25. The resulting models are syntactically valid and they can represent a realistic railway system thanks to the domain-specific well-formedness constraints.

To obtain a *realistic model* M_{real}, we manually captured a detailed runtime model snapshot of MoDeS3 that is similar to the one presented in Figure 6.2b with a total of 25 objects. Then, we removed all Train objects from M_{real} and unset all turnout directions, and used the resulting (partial) track layout P to find specific placements of trains and switching of turnouts such that the run times of the queries are maximized on the generated M_P^* witness models.

To assess the execution times of the query programs on random models, we generated models conforming to the MoDeS3 metamodel with up to a total of 25 objects. Due to the large space of possible graph models, representative sampling from the model space is an open question [JSS13; Sem+20c]. Nevertheless, we generated 250 models with the EMF random model generator[2] (**Rand**) with up to 5 Turnouts and up to 5 Trains, but none of them represents a railway setting that can occur because they all violated well-formedness constraints due to the completely random construction.

We also generated 250 models with the VIATRA Generator (**VG**) without an optimization objective, which satisfy all well-formedness and scope constraints used for generating witness models. However, the state exploration heuristics of the generator may lead to a biased sample.

Hardware setup

We use the Infineon Relax Lite Kit-V1 Board[3] to execute the query programs. This board has an XMC4500 F100-K1024 microcontroller and it is driven by a 120MHz system clock. This microcontroller is considered to be a mature industrial microcontroller and has an ARM Cortex-M4 core. For the present evaluation, the instruction cache on the device is not used as our primary focus is on the impact of domain-specific information about high-level program flow rather than microarchitectural effects.

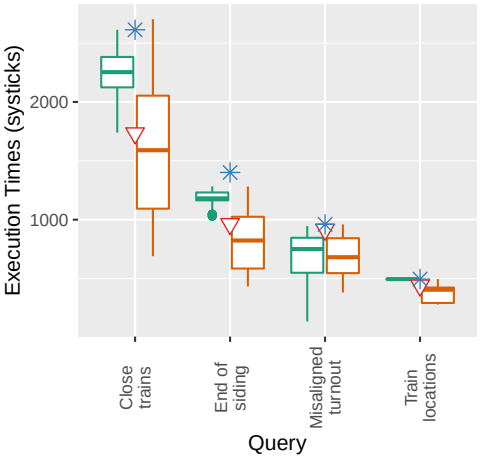
The bare-metal query programs are compiled with GCC compiler for ARM version 7.2.1 with `-O0` and `-g3` flags in debug mode. These programs run on the microcontroller while no other tasks (e.g., interrupts) are running. We rely on the cycle counter feature of the Data Watchpoint and Trace Unit in the device to extract the execution times of each query using a debugger. The embedded code used for the experiments as well as compiler and other configurations are available online[4].

6.4.2 Evaluation results

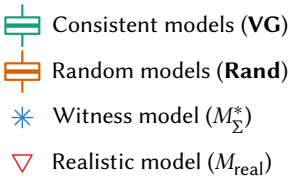
Research Question 1 (a)

We investigate if a witness model for a query can be obtained from simpler graph generation approaches, and we do this by measuring the execution times of queries over various models. Our results are presented in Figure 6.10a. The run times over models by **VG** is captured by the green boxes, while the orange ones show the run times over models by **Rand**. Each query was evaluated on the same two sets of models. Additionally, the respective query execution time over each witness model M_Σ^* is added to these figures for comparison, where M_Σ^* is the witness model generated using the objective function built from the low-level analysis results of aiT. Moreover, the run time over the hand-crafted M_{real} model is also presented.

[2] `https://github.com/atlanmod/mondo-atlzoo-benchmark`
[3] `http://www.infineon.com/xmc-dev`
[4] `https://imbur.github.io/cps-query/`



(a) Measured query times over consistent models, random models, witness models, and realistic models



Query	States visited	% of total
Close trains	992 681	19 %
End of siding	878 243	17 %
Misaligned turnout	875	< 1 %
Train locations	144	< 1 %
Total	5 273 100	100 %

(b) Results of state space exploration during witness generation from partial model using IPET linear equations adapted from aiT low-precision

Figure 6.10: Query execution times on fully random models and realistic models

Findings For each consistent model considered, queries exhibited the longest observed execution times on their respective witness models. In fact, for two queries, eos and mt, the execution time on the witness is longer than the maximum measured execution time over any other consistent model, which highlights the importance of our witness model generation technique.

Maximum run times over models generated by **Rand** can be both higher and lower than on witness models. For example, query ct takes 2% shorter over a random model than over its witness model, but the random model does not represent a realistic railway. On the contrary, query eos takes at least 8% longer to complete on the witness model than on any model generated by **Rand**. *Therefore, computing safe and tight WCET estimates of queries which execute over well-formed models (1) is infeasible by collecting run times over random models, and (2) necessitates finding witness models by employing sophisticated model generation approaches.*

Research Question 1 (b)

All possible refinements of P constitute a potentially large model space where finding M^*_P can be challenging and requires the graph solver to apply suitable abstractions for optimization. In our case, there are $3^5 \cdot \sum_{i=0}^{5} \binom{20}{i} = 5\,273\,100$ models (Total row in Figure 6.10b) in the space of well-formed concrete refinements of the initial partial model P containing the (selected) track layout, because each of the 5 turnouts can be in three different states, and there can be up to 5 trains which must be located on different segments. Thus, explicit enumeration of all models is possible for such a track layout, although it is computationally expensive.

As described in Section 6.4.1, we used the graph generator to add trains to a model with an empty track layout such that the expected query run times are maximized. Figure 6.10b presents the number of states explored by the graph generator compared to the space of all refinements.

Findings The number of states visited by the generator increases with the complexity of the query. For the most complex query ct, the generator was able to find the model with the highest estimated WCET after visiting 19% of the model space. For the least complex tl query, it visited only 144 states, which allowed the witness generation to finish almost instantly. In conclusion, the generator explores a fraction of the state space, making it more favorable than explicit enumeration.

Table 6.3: Query code complexity, measured execution time, and WCET estimates in systicks

Query	CC	Exec. time over M^*_Σ	$\mathbf{DS_\Sigma}$		aiT		
			w/aiT	**w/OTAWA**	low precision	high precision	OTAWA
Close trains	7	2652	**3133**	**3430**	3563	3038	4210
End of siding	6	1395	**1757**	**1820**	1757	1477	1860
Misaligned t.	5	939	**1097**	**1370**	1097	987	1370
Train locations	3	489	**592**	**695**	592	507	695

Table 6.4: Query code complexity, measured execution time, and WCET estimates with a partial model

Query	CC	Exec. time over M^*_P	$\mathbf{DS_P}$		aiT with partial memory image	
			w/aiT	**w/OTAWA**	low precision	high precision
Close trains	7	2544	**3079**	**3338**	3706	3091
End of siding	6	1275	**1554**	**1636**	1813	1523
Misaligned t.	5	969	**1097**	**1370**	1065	950
Train locations	3	504	**592**	**695**	586	506

Research Question 2

Our goal is to compare the computed WCETs obtained from different tools with our own techniques. For this RQ, we restrict our investigation to the scenario DS_Σ where only the metamodel and the well-formedness and scope constraints are known (i.e., case **(a)**), because only our technique but not the baseline tools (aiT, OTAWA) support processing a partial model as in DS_P (i.e., case **(b)**). Table 6.3 shows the WCET estimates for the 4 queries along with measured execution time (expressed in systicks) over the respective witness model.

Findings In the case of ct, our WCET estimation approach produces estimates 14% tighter than the one by aiT (low precision analysis). It is also important to point out that even without context-sensitive BB timings, our low precision approach provides only 3% higher estimates than aiT's high precision mode, which indicates that it is able to automatically identify infeasible paths in the program based on high-level domain-specific information. For OTAWA, improvements of the WCET estimate achieved in two cases: ct has a 23%, while eos has a 2% tighter estimate. For the rest of the queries, the analysis yields the same results as aiT low precision mode or OTAWA.

Therefore, WCET estimates by DS_Σ were at least as tight as those obtained by low-level IPET analysis. Thus, domain-specific analysis can improve WCET estimates while simultaneously synthesizing witness models to study query program behavior. Conceptually, it would be possible to formulate more precise DS_Σ estimates by incorporating low-level analysis results from the high precision mode of aiT as shown in Section 6.3.4, but such equations cannot be obtained from aiT.

Research Question 3 (a)

With this RQ, we look at the impact of query complexity on the computed WCET bounds, so that we can give recommendations on where our approach offers the greatest benefits. The execution times of queries over M^*_Σ in Table 6.3 provide a lower bound to the actual WCET (i.e., the longest possible execution time of the program over inputs which represent well-formed models in the model scope), while the CC columns shows query cyclomatic complexity. Since

the actual WCET of the program is unknown (but it must lay between the measured execution time and the WCET estimates produced by the analyses), we use the measured execution time over witness models as the baseline when discussing overestimation in WCET estimates.

Findings The biggest visible advantage of DS_Σ is in the case of the most complex query ct: the overestimation is 18% with BB timings from aiT, while the aiT low precision analysis computes a 34% higher value. In other cases, it produces the same result as aiT, with overestimates being between 16% (query mt) and 26% (query eos). We come to the same conclusion using BB timings from OTAWA, although these timings are slightly more conservative. The high precision analysis available in aiT is able to leverage the microarchitectural properties and thus provide the most precise estimates with the overestimation being 14% (observed for query ct). The overestimation increases with CC of the query code only in the case of high precision aiT analysis.

In general, DS_Σ computes a safe WCET bound and additionally provides a witness model. Moreover, it is able to discover additional infeasible paths the WCET estimate for the most complex query, thus provide a tighter estimate.

Research Question 3 (b)

The goal of this RQ is to conclude if providing an initial graph model with elements of the graph model known upfront can lower the WCET, thus provide an execution time estimate for M_P^* lower than the one computed for M_Σ^*. The execution times of queries over M_P^* are presented in Table 6.4 similarly to Table 6.3. The estimates computed by DS_P are valid and safe for any possible in-memory representation of the refinements of P (concrete models containing the designated track layout).

Additionally, Table 6.4 shows estimates from aiT computed by value analysis (VAL) over the initial track layout model provided to aiT as a *partial memory image*, where the unknown parts of P (locations of trains and the directions of turnouts) are left uninitialized. While WCET estimates based on this input are *not safe*, as they do not consider all possible in-memory representations of the refinements of P, we still added these numbers to Table 6.4, because they correspond to the most likely usage of existing WCET analysis tools with partial input.

Findings Comparing DS_Σ and DS_P, we discover that providing an initial model tightens WCET estimates for the two more complex queries, but does not change the estimate for the two less complex ones. This is due to the nature of the initial model and query search plans. There is no arrangement of trains on the initial track layout such that the WCET estimate from DS_Σ for ct and eos run times is reached, whereas the evaluation of mt and tl depends less on the track layout.

Additional observations For mt and tl, observed query run times on the witness models M_P^* were higher than on M_Σ^*, which can be attributed to the placement of data in memory as mentioned in Section 6.1.2.2. Nevertheless, they were still within the WCET estimates from both DS_Σ and DS_P.

Unexpectedly, the partially specified data yields higher WCET estimates by both analysis modes of aiT for the two more complex queries, ct and eos, when compared with the estimates in Table 6.3. Thus, for these queries, it is not possible to tighten WCET estimates for partial input data even with the caveat that all objects in the partial input are statically allocated. We have reported our observation to the developers of aiT at AbsInt GmbH, and they have confirmed that the discrepancy in the estimates is due to the differences in the placement of data in the two binaries. Note that these analyses solve a less general challenge compared to DS_P since they only consider one possible physical layout of the partial data in memory (and not all possible in-memory layouts), thus they are highlighted in gray in Table 6.4.

On the other hand, aiT high-precision mode produces a *lower estimate* based on the provided initial model than what is measured over M_P^* (highlighted in red in Table 6.4). The underlying reason for this is that the tool relies on the exact placement of data in memory rather than the abstract graph model the data encodes. Eventually, in M_P^*, the model objects were stored in the memory in different order which resulted in a higher runtime (see note about data placement in Section 6.1.2.2).

In general, providing partial input data allows for tightening WCET estimated for complex queries even in cases where value analysis with partial input data is not safe. For less complex queries, supplying a partial memory image can tighten the results of value analysis considerably, but requires committing to a specific in-memory representation of the partial data statically.

6.4.3 Threats to validity

Internal validity The current evaluation was performed on a device where the executing binary only included the query-based monitor, so we can assume that the measurements presented here precisely show the execution times of queries. Since the exact WCET of the program is unknown, we used the longest observed execution time to assess the overestimation of the computed WCETs. In reality, this overestimation might be lower than what is reported here, which would make our WCET estimates tighter than presented.

External validity We carried out the evaluation using one specific hardware and compiler, thus the presented results may not generalize to other platforms. Furthermore, the presented approach is applicable to any query-based monitor generated with Algorithm 1. However, evaluation of the WCET estimation techniques using additional case-studies with query-based runtime monitors from different domains could further improve the confidence in the evaluation results.

6.5 Related work

Numerous static and probabilistic WCET analysis methods have been discussed in [Wil+08; Koz16]. Abella et al. [Abe+15] compares the most common WCET estimation approaches for programs in real-time systems and highlights their strengths and limitations. Based on the categorization of approaches of this latter work, our approach is a *high-level, static deterministic timing analysis (SDTA)* which provides *safe* execution bounds for embedded programs executing complex graph queries. Furthermore, measurement-based WCET estimations [Wen+05; LB16] and probabilistic methods [HHM09; Cuc+12] are out of scope for our work. Nevertheless, we focus on *semantic-aware WCET estimation* [Mai+17], which aims at providing safe and tight estimates for programs where there are some semantic limitations on the input data, which cannot be automatically explored and exploited by current analysis techniques, and often times manual annotations of the code are necessary. In our case, control flow is often constrained by complex rules which are based on various properties of graph models. We provide an overview of existing work related to *graph-based programs*, *runtime monitoring* and *program flow analysis*.

Graph query programs: Existing platforms Graph models and queries have been often used in design models and tools of real-time systems [Jür03; Gie+03; Bur+04]. Furthermore, graph-based techniques are used in various IoT and edge computing applications [Xie+21; Li+19]. However, due to the soft real-time requirements of such applications, the WCET analysis aspect is often neglected. The focus of our work is to provide safe and tight WCET of such programs, and thus extend their application area.

Graph query programs: WCET estimation One of the few related works that investigates real-time properties of graph-based techniques is [Bur+05]. Motivated by the expressiveness of *story diagrams* [Fis+98], the authors evaluate the applicability of this high-level modeling formalism to recognize hazardous situations in real-time systems. Their work investigates worst-case execution times of imperative programs generated from such story diagrams by executing measurements of manually created worst-case inputs. In contrast, our work aims to automatically synthesize worst-case well-formed input models as part of static analysis.

Runtime monitoring: Hard real-time embedded systems One of the earlier works in the field is The Temporal Rover [Dru00]. This framework can generate monitoring code from temporal logic formulae with low overhead, but the verification of properties is done in a large part on a powerful remote host, while our method does not rely on any external component. The concept of *predictable monitoring* was introduced in [ZDG09] where static scheduling techniques were used to show that a monitor fits its allocated time frame, but the analysis of monitoring tasks is out of its scope which is the topic of this current paper. Finally, synchronous component execution and observable program states are the main assumptions made in [Pik+10] to support sampling-based monitoring of input streams in real-time systems, whereas our work targets monitors executing complex queries over a graph model capturing contextual information on a high-level of abstraction.

Runtime monitoring: Real-time database queries In real-time databases [OS95], access to data has strict time constraints. The work in [HOT89] presents a data sampling-based statistical method to evaluate aggregate queries in a database. There is a trade-off between time available for query execution and the precision of the estimate. Such estimations would not be acceptable in a monitoring setting where precise query results are expected. The real-time object-oriented database RODAIN [TR96], which targets telecommunication applications, does not support *hard real-time* transaction (i.e., query) types, because it is considered too costly for the target domain. However, our objective is exactly to provide such guarantees over graph models to support hard real-time applications.

Program flow analysis Timing analyzers for program flow analysis often employ some version of the implicit path enumeration technique (IPET) [LM97]. The general idea behind this method is to use the control flow graph (CFG) of the program to create an integer linear program (ILP) where each variable encodes the number of executions of a corresponding basic blocks, and the objective function is to maximize their total execution time. Besides the IPET method, several *tree-based* methods exist which use a tree representation of the program (obtained from the source code or compiled binary) and apply some traversal to find the longest path in a program [Lim+95; CB02; BFL17]. In any case, the effectiveness of these methods rely on precise program *flow facts* (e.g., loop bounds, infeasible paths) to be able to determine a safe and tight WCET estimate. Although there are several advanced (semi-)automated techniques available today to derive additional constraints on the program flow and thus improve the precision of the WCET estimate [Gus+06; Erm+07; CJ11; KKZ13; Lis14], there is still a significant manual effort needed to specify flow facts [Abe+15]. Most closely related to our current work is [KKZ13], which uses abstraction refinement to reduce WCET estimates by *squeezing*. However, this approach cannot exclude longest execution paths from the program which are infeasible due to complex domain-specific constraints on the inputs.

6.6 Conclusions

In this chapter, we presented a method to *provide safe and tight WCET bounds for runtime monitoring programs derived from graph queries* to enable their use in real-time systems. We

provided a static WCET estimate by incorporating low-level analysis results from traditional IPET-based tools and high-level domain-specific constraints into the objective function of an advanced graph solver. In addition to a tight WCET estimate, the result also entails a witness graph model where the query-based monitoring program execution time is expected to be the longest.

We carried out extensive evaluation of our approach on an industry-grade hardware platform using a variety of graph models as inputs for query programs, and assessed the tightness of computed WCET by comparing it to the results produced by two different tools. We constructed witness models for highest estimated execution times of queries as well as random graph models as inputs for graph query programs as an attempt to showcase high execution times. While we have no formal guarantee that worst-case timing behavior is exhibited on witness models as inputs, in all our experiments, the longest execution times were always measured on such witness models.

In the short run, the proposed approach can be improved by passing the results of high-precision IPET analysis (including CFG unrolling and pipeline analysis) to the graph solver, while the evaluation of the approach should be done on a different hardware platforms as well. As a part of a long-term future research agenda, our approach could be extended to provide witness models with specific data placement in memory where the execution time equals to the WCET of the program.

Chapter 7

Summary of contributions

As per university regulations, contributions in this section are formally proposed using the *first-person singular* ("I"). The highlighted contributions, published in the works cited therein, are the work of the author of this thesis alone. The rest of the thesis follows the conventions of the field by using the *first-person plural* ("we").

7.1 Partial modeling for quantitative extra-functional analysis

In order to tackle **Challenges 3** and **5**, the first group of contributions in this thesis provides extended FOL structures to compute quantitative extra-functional metrics of incomplete or inconsistent architecture models.

In *Contribution 1.1.1*, we proposed *4-valued partial models* [j2], which adopt the 4-valued Belnap-Dunn logic [Bel77; KO17] to represent *incomplete* or *inconsistent* models. We also introduced *multi-objects* to compactly represent from zero to many potential model elements with a single object by interpreting the existence and equality of objects with 4-valued logic. Object equality can be also used to denote the *merging* of elements from multiple sources of information [SE06; CNS12], where inconsistency-tolerance is exploited to mark merge conflicts and conflicting executions of composite view transformations [c7; d21] (**Challenge 3**).

In *Contribution 1.1.2*, we proposed an *abstraction for numerical attribute values and graph metrics* [j2] (**Challenge 5**) in partial models by exploiting *interval abstraction* [Kul09] to combine partial modeling with *value analysis* [FFJ12].

Lastly, in *Contribution 1.2*, we proposed *scoped partial models* [j1] as a conceptual core for evaluating *model size constraints* over partial models (**Challenge 5**). By employing *polyhedron abstraction* [BHZ08], they offer finer-grained abstraction over the number of model elements represented by multi-objects. This allows representing extra-functional constraints on the number of model elements (e.g., model size constraints, linear cost functions) within the partial model. As a limitation, scoped partial models rely on *3-valued logic* [SV17] and lack inconsistency-tolerance.

Contribution group 1 I defined *4-valued* and *scoped* extensions of the partial graph model formalism for the quantitative extra-functional analysis of system architecture models with unknown properties and pending design decisions.

Contribution 1.1 I formalized *4-valued partial models* as a conceptual core to introduce *inconsistency-tolerance* into architecture models and other graph models [j2; c7; d21].

1.1.1. I proposed the use of 4-valued Belnap–Dunn logic to explicitly capture both *unknown properties* and *pending design decisions* (paracompleteness), as well as *inconsi-*

tencies (paraconsistency).

1.1.2. I proposed the use of *interval abstraction* to handle *unknown* and *uncertain attribute values*, which enables the *under-* and *over-approximation* of logic formulas over models containing *complex numerical constraints*.

Contribution 1.2 I formalized *scoped partial models* as a conceptual core for model generation with linear numerical constraints, proposing the use of *polyhedron abstraction* to express constraints on model size, number of graph pattern matches, and costs [j1].

Added value The contributions in this contribution group provide abstractions for incomplete and inconsistent architecture models for evaluating structural and numerical constrains and quantitative extra-functional metrics. As such, they serve as a theoretical basis for the contributions in **Contributions groups 2–3**.

In particular, the under- and over-approximation of constraint satisfaction enables the use of the abstract DPLL [NOT06; Bra+13] algorithms for the synthesis of architecture models. They aid in detecting surely violated functional and extra-functional requirements. Hence, when designing an architecture, mistakes and inconsistencies can be detected even before the full model is ready [CNS12; SV17][c7]. In automated synthesis, the detection of inconsistencies results in backtracking in the search space [SNV18; Var+18][j1] to speed up model generation.

To our best knowledge, these contributions are the first to address the sound and complete abstract DPLL synthesis of candidate architecture models.

Applications and related contributions Contributions in this contribution group were published as part of joint work with Dániel Varró, Oszkár Semeráth, and Aren A. Babikian.

Joint work also with Zoltán Micskei, András Vörös, Zoltán Szatmári, and Csaba Hajdu [c8] proposed an end-to-end framework for the run-time monitoring and testing of autonomous vehicles with coverage metrics based on qualitative graph abstractions. Building on this framework, joint work also with Anqi Li [j3; c9; d20] investigated the synthesis of test cases for data processing systems and self-driving vehicles by adopting *constant abstraction* to represent unknown attribute values, which are then filled by an SMT solver [MB08]. This approach introduces *refinement units* as a means to unify various reasoning techniques with partial graph models and abstract domains.

7.2 Reasoning with partial models

The second group of contributions in this thesis is aimed at providing reasoning capabilities over partial models for extra-functional analyses.

Contribution 2.1 [c7; d21] proposes a novel *view transformation language* and a *transformation engine* as an open source prototype implementation primarily aimed at the construction of analysis models from architecture models.

The transformation language provides *parallel composition* of transformation to enable the integration of knowledge (such as the dependability attributes of various system components in **Challenge 3**) from multiple experts. We introduced *relation-based composition*, where the composition can be fine-tuned by providing a *glue transformation* without having to modify the composed transformations.

Thanks to the change-driven execution of the underlying Viatra *Query* and *Transformation* framework [Ujh+15], a *reactive* and target *incremental* transformation engine is obtained. Exploiting the *inconsistency-tolerance* of 4-valued partial models from *Contribution 1.1*, the proposed transformation engine remains *validating* (**Challenge 2**) and explicitly marks inconsistencies even if the composed transformations or the state of the source model are not consistent. An

implicitly constructed *traceability model* links source and target elements so that analysis results can be *back-annotated* to the architecture model.

Contribution 2.2 [j1] targets the synthesis of design candidates with multiplicity constraints (i.e., structural constraints with lower or upper bounds on the number of model objects or relationships) in **Challenge 5**. The proposed model generator also supports an extended version of *class scopes*, which are a popular approach for constraining model generation problems introduced by Alloy [Jac02].

More complex logical well-formedness constraints, as in **Challenge 2**, can also be handled by manual translation to linear inequalities and scoped partial models (*Contribution 1.2*).

Numerical reasoning is provided by Linear Programming (LP) and Integer Linear Programming (ILP) solvers, such as [Clp; Cbc]. Based on these low-level background theories, we provide *scope propagation* operators for scoped partial models that *refine* under- and over-approximations for partial model metrics. The refined approximations for metrics are used to *prune* the search space (**Challenge 1**) in an abstract DPLL [NOT06; Bra+13] decision procedure.

Contribution group 2 I proposed two reasoning techniques for deriving quantitative extra-functional metrics from partial models of systems and considering multiplicity constraints on architecture models.

Contribution 2.1 I proposed a fully compositional view transformation language and developed a reactive, incremental, validating, and inconsistency-tolerant view transformation engine for executing view transformations using 4-valued partial models [c7; d21].

2.1.1. I identified the levels of *parallel composition* support in view transformation languages, as well as the *consistent* and *validating* properties of view transformation engines.

2.1.2. I defined a *fully compositional view transformation language*.

2.1.3. I proposed a *reactive, incremental, validating, and inconsistecy-tolerant* transformation engine for unidirectional view transformations based on 4-valued partial models.

2.1.4. I evaluated the practical applicability and scalability of the approach using the open source Train Benchmark framework.

Contribution 2.2 I proposed a model generation approach that combines a DPLL-like decision procedure based on partial modeling with LP and ILP background solvers to synthesize and optimize models according to objective functions defined as linear programs, such as *multiplicity constraints*, *total model size*, and *cost functions*, using scoped partial models [j1].

2.2.1. I defined a mapping of *structural* and *well-formedness constraints* into linear numerical constraints that can *under-* and *over-approximated* on scoped partial models to efficiently guide model generation.

2.2.2. I extended an open-source graph solver with scoped partial model support by integrating LP and ILP solvers for numerical reasoning.

2.2.3. I evaluated the effectiveness of the approach using multiple industrial case studies, including a satellite constellation synthesis task from NASA JPL.

Added value To our best knowledge, the presented transformation engine is the first to support *reactive*, *validating*, *inconsistency-tolerant* execution of view transformations with *parallel composition* [c7]. This allows for the collaboration of multiple domain experts in the design of transformations, and the combination of transformations relating to multiple viewpoints, while still enabling the validation of target model structural constraints and the incremental maintenance of the target model according to source model updates.

In our empirical evaluations, reasoning with scoped partial models provided a *7-fold reduction in running time* for model generation in problems with complex multiplicity constraints [j1]. Even in domains without such constraints, the overhead added by scope propagation is minimal. While

unsatisfiable problems usually pose a difficulty to model generators based on partial modeling, scope propagation operators can efficiently detect unsatisfiable multiplicity constraints with LP and ILP background solvers, leading to much better scalability.

Applications and related contributions Contributions in this contribution group were published as part of joint work with Dániel Varró and Oszkár Semeráth.

The proposed ViewModel view transformation engine is available as an open source project under the Eclipse Public License 1.0 at `https://github.com/ftsrg/viewmodel` and in [d21]. The tools has been applied for automatically constructing Stochastic Petri Net (SPN) reliability analysis models for design candidates of a redundant automotive subsystem in an industrial project with *thyssenkrupp Hungary Kft.*

The implementation of the proposed scope propagation strategies is available as part of the open source Viatra-Generator model generation framework under the Eclipse Public License 1.0 at `https://github.com/viatra/VIATRA-Generator`.

Joint work also with Gábor Szárnyas, Aren A. Babikian, Boqi Chen, and Chuning Li investigated the generation of *realistic test cases* for modelling tools by focusing the generator towards practically relevant corner cases [j4]. To ensure the realistic distribution of model elements by type, we relied on numerical reasoning techniques introduced in this contribution group.

Related contributions by my student Máté Földiák focused on numerical reasoning with reliability and performability metrics *directly over partial models* (instead of constructing separate analysis models) by computing under- and over-approximations. An initial prototype [c10; d22] was created that synthesizes optimal satellite constellation mission architectures [Her+17] according to performability objectives based on the *refinement unit* framework [j3; c9].

Joint work also with Boqi Chen and Sebastian Pilarski investigated the use logic reasoning with graphs for ensuring consistency in image recognition [c11].

7.3 Model-based quantitative extra-functional analysis

The third group of contributions presents concrete applications of partial model based reasoning for extra-functional analysis problems.

Contribution 3.1 [c12; l18; r19] provides a model-based technique for the automatic performability evaluation (**Challenge 3**) of reconfigurable system architectures based on Stochastic Petri Net (SPN) analysis models. As a specification language for performability properties, we adopt the incremental view transformations from *Contribution 2.1* to derive SPN models for each possible configuration.

We introduced *mission automata* as an extension of the *Graph Transformation Abstract State Machine* (GT+ASM) [VB07] formalism. The possible system failures are described by the *observable* attributes of a *run-time* version of the architecture model, which are defined based on the state of the SPN model and the traceability relationships between the architecture and SPN models. The mission automaton reacts to failures by changing the structure or the *controllable* attributes of the run-time model with graph transformations. We construct a *Phased-Mission System* (PMS) [MB99] from the SPN analysis models, where the reachable state space of the mission automaton serves as the high-level phase model.

Contribution 3.2 [j5] presents *witness model synthesis* in the domain-specific *Worst-Case Execution Time* (WCET) analysis for *graph query based runtime monitor programs* (**Challenge 4**). This allows us to tackle (i) WCET based on a single runtime snapshot, (ii) WCET based on the metamodel and well-formedness constraints of the runtime snapshots, and (iii) WCET based on a *partial* runtime snapshot.

Our analysis combines low-level *Implicit Path Enumeration Technique* (IPET) WCET analysis with a high-level, *domain specific analysis* that takes into account the well-formedness constraints

of runtime models. We adapted the low-level IPET analyses from the OTAWA [Bal+10] and aiT WCET analysis tools.

To obtain a high-level analysis, we associate *auxiliary graph queries* based on the runtime monitor program with the variables of the IPET Integer Program to form a graph-based description of the WCET analysis problem. We synthesize the witness model using the extended generator from *Contribution 2.2*.

Contribution group 3 I proposed analysis methods for the reliability and worst-case execution time estimation based on reasoning with partial models.

Contribution 3.1 I proposed a model-based technique for automatically deriving phased-mission stochastic Petri net models for complex reconfigurable systems based on fully compositional view transformations [c12; l18; r19].

3.1.1. I defined *mission automata*, a formalism that leverages graph transformation abstract state machines (GT+ASM), as well as *observable* and *controllable* runtime features in architecture models, for a high-level description of runtime reconfigurations in cyber-physical systems.

3.1.2. I proposed a technique to construct *phased-mission stochastic Petri nets* for the dependability analysis of reconfigurable cyber-physical systems, where the reconfigurations are captured by a mission automaton, while the reliability processes of the system are described by individual stochastic Petri nets automatically constructed from the system architecture models via a view transformation.

3.1.3. I evaluated the practical applicability and scalability of the approach on a case study based on the analysis of a reconfigurable production cell.

Contribution 3.2 I proposed an approach for finding *witness models* of worst-case execution times of query-based runtime monitor programs in critical embedded systems by model generation using scoped partial models with linear programming [j5].

3.2.1. I proposed a *high-level static analysis* technique for query-based runtime monitor programs to estimate execution time on a given runtime model snapshot that combines *domain-specific* information from query plans with state-of-the-art IPET *low level analysis* output about the microarchitectural characteristics of the target execution platform.

3.2.2. I formulated the *witness generation* problem as a model generation task with a linear program objective.

3.2.3. I evaluated the practical applicability and scalability of the proposed approach by generating witness models for queries from the open source Train Benchmark in the context of the MoDeS3 CPS demonstrator.

Added value Model-based phased-mission system generation is an efficient, architecture-based analysis method for reconfigurable CPS. In our experiments, analysis models with up to 1028 different configurations could be constructed by exploring the state space of mission automata within 20 sec.

In the MoDeS3 CPS demonstrator, domain-specific WCET analysis reduced the metamodel-based WCET estimate with up to 12% compared to aiT and 25% compared to OTAWA on queries where the well-formedness constraints of the possible runtime snapshots impacted the execution time. It also managed to provide partial model based WCET estimates, while aiT provided an incorrect (lower than the actual program execution time of hardware) estimate due to its lack of partial model based estimate support.

Applications and related contributions Contributions in this contribution group were published as part of joint work with Dániel Varró, Brett H. Meyer, and Márton Búr. The concept

of a *witness model* [j5] was introduced by Búr [Búr21].

Our background work related to dependability and performability analysis published jointly also with Miklós Telek, Tamás Bartha, Dániel Darvas, Ákos Hajdu, Attila Klenik, and Vince Molnár focused on the numerical solution of *Generalized Stochastic Petri Net* (GSPN) models. In [c13; c14], we proposed a configurable GSPN analysis framework based on the *block Kronecker decomposition*. In [c15], we proposed a symbolic state-space exploration engine for GSPN models. The analysis framework is available as part of the PetriDotNet [j6] tool at `https://inf.mit.bme.hu/en/research/tools/petridotnet` and was applied for the reliability analysis of a redundant automotive subsystem in an industrial project with *thyssenkrupp Hungary Kft.*

Related contributions by my student Simon Nagy investigated hierarchical reliability modeling and analysis based on statecharts and probabilistic programming [c16]. The resulting tool is available as part of the open source *Gamma Statechart Composition Framework* under the Eclipse Public License 1.0 at `https://github.com/FTSRG/gamma`. The tool was also used in our industrial project and a case study model adapted from the project is available in the *Models for Formal Analysis of Real Systems* (MARS) repository at `http://mars-workshop.org/repository/028-EPAS.html`. Related contributions by my student Dániel Szekeres investigated efficient numeric solution of large *Continuous-Time Markov Chains* (CTMC) arising from *Static Fault Tree* (SFT) analysis using the *Tensor Train* decomposition for large matrices [c17]. An open source prototype implementation of the analysis tool is available under the Apache License 2.0 at `https://github.com/ftsrg/StoATT`.

7.4 Future work

Our long-term research goal is to *facilitate the design of complex system architectures* according to various, stringent functional and extra-functional requirements by *unifying and developing reasoning techniques* for their analysis and synthesis.

In particular, we highlight the following open challenges for future research:

- *Integration of reasoning techniques within sound and complete design-space exploration.* While this work addressed the use of 4-valued logic [Bel77][c7; j2] to explicitly highlight inconsistencies in architecture and analysis models, as well as polyhedron abstraction [CH78; BHZ08][j1] to reason about model size constraints and other linear inequalities, the simultaneous use of such abstractions remains unresolved. More broadly, the use of relational abstractions [Min04] or other abstract domains could lead to more efficient design-space exploration by pruning unsuitable design candidates earlier.

- *Integration of external extra-functional analysis and reasoning tools.* Partial model based analysis approaches can reason about many variants of a given design at once by explicitly encoding design decisions that are yet to be made. Hence, feedback about candidate architectures can be provided early in the design process. An initial attempt at integrating existing analyses as *refinement units* was made in [j3; c9], where external SMT solvers [MB08] are used to determine attribute values, and in [c10], where the performability of architectures is estimated with Markov chains. Nevertheless, further research is needed to improve analysis performance and explore the set of extra-functional analyses which can be executed on partial architectures.

- *Reasoning about system behavior during adaptation at runtime.* Self-adaptive strategies for run-time adaptation, where new system configurations are synthesized at runtime in response to failure events of the system, *require quantitative verification of the configurations at runtime* [Cal+12]. Sound and complete design-space exploration techniques for configuration synthesis could support provably sound adaptation strategies.

Publications

Number of publications	23
Number of peer-reviewed journal papers (written in English)	6
Number of articles in journals indexed by WoS or Scopus	5
Number of publications (in English) with at least 50% contribution	5
Number of peer-reviewed publications	21
Number of independent citations (as of January 2023)	98

Publications related to the contributions

	Journal papers	International conference and workshop papers	Reports and artifacts
Group 1	[j1]*, [j2], [j3]	[c7]*, [c8], [c9]	[d20]
Group 2	[j1]*, [j4]	[c7]*, [c10], [c11]	[d21], [d22]
Group 3	[j5], [j6]	[c12], [c13], [c14], [c15], [c16], [c17]	[l18], [r19]

*Publication related to multiple contribution groups

This classification follows the PhD publication scoring system of the faculty.

Journal papers

[j1] **Kristóf Marussy**, Oszkár Semeráth, and Dániel Varró. "Automated Generation of Consistent Graph Models with Multiplicity Reasoning". In: *IEEE Trans. Softw. Eng.* 48.5 (2022), pp. 1610–1629. DOI: 10.1109/TSE.2020.3025732

[j2] **Kristóf Marussy**, Oszkár Semeráth, Aren A. Babikian, Dániel Varró. "A Specification Language for Consistent Model Generation based on Partial Models". In: *J. Obj. Technol.* 19.3, 12 (2021). DOI: 10.5381/jot.2020.19.3.a12

[j3] Aren A. Babikian, Oszkár Semeráth, Anqi Li, **Kristóf Marussy**, Dániel Varró. "Automated Generation of Consistent Models by Combining Geometric and Qualitative Reasoning". In: *Softw. Syst. Model.* (2021). DOI: 10.1007/s10270-021-00918-6

[j4] Oszkár Semeráth, Aren A. Babikian, Boqi Chen, Chuning Li, **Kristóf Marussy**, Gábor Szárnyas, Dániel Varró. "Automated generation of consistent, diverse and structurally realistic graph models". In: *Softw. Syst. Model.* (2021). DOI: 10.1007/s10270-021-00884-z

[j5] Márton Búr, **Kristóf Marussy**, Brett H. Meyer, Dániel Varró. "Worst-Case Execution Time Calculation for Query-Based Monitors by Witness Generation". In: *ACM Trans. Embed. Comput. Syst.* 20.6 (2021), pp. 1–36. DOI: 10.1145/3471904. arXiv: 2102.03116 [cs.SE]

[j6] András Vörös, Dániel Darvas, Ákos Hajdu, Attila Klenik, **Kristóf Marussy**, Vince Molnár, Tamás Bartha, István Majzik. "Industrial applications of the PetriDotNet modelling and analysis tool". In: *Sci. Comput. Program.* 157 (2018), pp. 17–40. DOI: 10.1016/j.scico.2017.09.003

International conference and workshop papers

[c7] **Kristóf Marussy**, Oszkár Semeráth, and Dániel Varró. "Incremental View Model Synchronization Using Partial Models". In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 2018, pp. 323–333. DOI: 10.1145/3239372.3239412

[c8] István Majzik, Oszkár Semeráth, Csaba Hajdu, **Kristóf Marussy**, Zoltán Szatmári, Zoltán Miskei, András Vörös, Aren A. Babikian, Dániel Varró. "Towards System-Level Testing with Coverage Guarantees for Autonomous Vehicles". In: *ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems*. IEEE, 2019. DOI: 10.1109/MODELS.2019.00-12

[c9] Oszkár Semeráth, Aren A. Babikian, Anqi Li, **Kristóf Marussy**, Dániel Varró. "Automated generation of consistent models with structural and attribute constraints". In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 2020, pp. 187–199. DOI: 10.1145/3365438.3410962

[c10] Máté Földiák, **Kristóf Marussy**, Dániel Varró, István Majzik. "System Architecture Synthesis for Performability by Logic Solvers". In: *Proceedings of the 25th ACM / IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 2022. DOI: 10.1145/3550355.3552448

[c11] Boqi Chen, **Kristóf Marussy**, Sebastian Pilarski, Oszkár Semeráth, Dániel Varró. "Consistent Scene Graph Generation by Constraint Optimization". In: *37th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2022. DOI: 10.1145/3551349.3560433. Forthcoming

[c12] **Kristóf Marussy** and István Majzik. "Constructing Dependability Analysis Models of Reconfigurable Production Systems". In: *IEEE 14th International Conference on Automation Science and Engineering*. IEEE, 2018. DOI: 10.1109/COASE.2018.8560551

[c13] **Kristóf Marussy**, Attila Klenik, Vince Molnár, András Vörös, István Majzik, Miklós Telek. "Efficient Decomposition Algorithm for Stationary Analysis of Complex Stochastic Petri Net Models". In: *PETRI NETS 2016*. LNCS 9698. Springer, 2016, pp. 281–300. DOI: 10.1007/978-3-319-39086-4_17

[c14] **Kristóf Marussy**, Attila Klenik, Vince Molnár, András Vörös, Miklós Telek, István Majzik. "Configurable numerical analysis for stochastic systems". In: *2016 International Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR)*. IEEE, 2016. DOI: 10.1109/SNR.2016.7479383

[c15] **Kristóf Marussy**, Vince Molnár, András Vörös, István Majzik. "Getting the Priorities Right: Saturation for Prioritised Petri Nets". In: *PETRI NETS 2017*. LNCS 10258. Springer, 2017, pp. 223–242. DOI: 10.1007/978-3-319-57861-3_14

[c16] Simon József Nagy, Bence Graics, **Kristóf Marussy**, András Vörös. "Simulation-based Safety Assessment of High-level Reliability Models". In: *Proceedings of the 4th Workshop on Models for Formal Analysis of Real Systems*. EPTCS 316. 2020. DOI: 10.4204/EPTCS.316.9. arXiv: 2004.13290 [cs.SE]

[c17] Dániel Szekeres, **Kristóf Marussy**, and István Majzik. "Tensor-based reliability analysis of complex static fault trees". In: *17th European Dependable Computing Conference*. IEEE, 2021. DOI: 10.1109/EDCC53658.2021.00012

Local workshop paper

[l18] **Kristóf Marussy** and István Majzik. "Architecture-based Dependability Analysis of Reconfigurable and Adaptive Systems". In: *Proceedings of the 26th Minisymposium of the Department of Measurement and Information Systems*. BME Méréstechnika és Információs Rendszerek Tanszék, 2019

Technical report (not peer reviewed)

[r19] **Kristóf Marussy** and István Majzik. *Constructing Phased-Mission Systems for Dependability Analysis of Reconfigurable Production Systems.* Tech. rep. 2018. URL: http://doi.org/10.5281/zenodo.1290661

Artifacts (peer reviewed as part of the corresponding publications)

[d20] Oszkár Semeráth, Aren A. Babikian, Anqi Li, **Kristóf Marussy**, Dániel Varró. *Artifacts for "Automated Generation of Consistent Models with Structural and Attribute Constraints".* Approved as *reuseable* by the MODELS '20 Artifact Evaluation Committee [c9]. 2020. DOI: 10.5281/zenodo.3950552

[d21] **Kristóf Marussy**, Oszkár Semeráth, and Dániel Varró. *ViewModel Tool and Benchmark Results for "Incremental View Model Synchronization Using Partial Models".* Approved by the MODELS '18 Artifact Evaluation Committee [c7]. 2018. DOI: 10.5281/zenodo.1318156

[d22] Máté Földiák, **Kristóf Marussy**, Dániel Varró, István Majzik. *Artifacts for "System Architecture Synthesis for Performability by Logic Solvers".* Approved as *functional* by the MODELS '22 Artifact Evaluation Committee [c10]. 2022. DOI: 10.5281/zenodo.6974248

Additional publications (not related to the contributions)

Book chapter

[b23] Nenad Tomašev, Krisztián Buza, **Kristóf Marussy**, Piroska Buzáné Kis. "Hubness-Aware Classification, Instance Selection and Feature Construction: Survey and Extensions to Time-Series". In: *Feature Selection for Data and Pattern Recognition.* SCI 584. Springer, 2015, pp. 231–261. DOI: 10.1007/978-3-662-45620-0_11

International conference and workshop papers

[c24] **Kristóf Marussy** and Krisztián Buza. "SUCCESS: A New Approach for Semi-supervised Classification of Time-Series". In: *ICAISC 2013.* LNCS 7894. Springer, 2013, pp. 437–447. DOI: 10.1007/978-3-642-38658-9_39

[c25] Krisztián Buza, Júlia Koller, and **Kristóf Marussy**. "PROCESS: Projection-Based Classification of Electroencephalograph Signals". In: *ICAISC 2015.* LNCS 9120. Springer, 2015, pp. 91–100. DOI: 10.1007/978-3-319-19369-4_9

[c26] **Kristóf Marussy**, Ladislav Peska, and Krisztián Buza. "Recommendations of Unique Items Based on Bipartite Graphs". In: *Proceedings of the 9th Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications.* Kyushu University, 2015

Local workshop paper

[l27] **Kristóf Marussy** and Krisztián Buza. "Hubness-based indicators for semi-supervised time-series classification". In: *8th Japanese-Hungarian Symposium on Discrete Mathematics and Its Applications.* BME, 2013

Poster

[a28] **Kristóf Marussy** and Krisztián Buza. "PROGRESS: Projection-Based Gene Expression Classification". In: *Innovations in Medicine Conference.* Akadémiai Kiadó, 2014

References

[Abd+14] Hani Abdeen et al. “Multi-objective optimization in rule-based design space exploration”. In: *ASE ’14*. ACM, 2014, pp. 289–300. DOI: 10.1145/2642937.2643005.

[Abd+18] Raja Ben Abdessalem et al. “Testing Autonomous Cars for Feature Interaction Failures Using Many-Objective Search”. In: *33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 143–154. DOI: 10.1145/3238147.3238192.

[Abe+15] Jaume Abella et al. “WCET analysis methods: Pitfalls and challenges on their trustworthiness”. In: *10th IEEE International Symposium on Industrial Embedded Systems*. IEEE, 2015, pp. 39–48. DOI: 10.1109/SIES.2015.7185039.

[ACG17] Mohammed Al-Refai, Walter Cazzola, and Sudipto Ghosh. “A Fuzzy Logic Based Approach for Model-Based Regression Test Selection”. In: *MODELS*. IEEE, 2017, pp. 55–62. DOI: 10.1109/MODELS.2017.17.

[ADW16] Ahmad Salim Al-Sibahi, Aleksandar S. Dimovski, and Andrzej Wasowski. “Symbolic execution of high-level transformations”. In: *SLE*. Springer, 2016, pp. 207–220.

[AG16] Colin Atkinson and Ralph Gerbig. “Flexible Deep Modeling with Melanee”. In: *Modellierung Workshopband*. Vol. P-255. LNI. GI, 2016, pp. 117–122. URL: https://dl.gi.de/20.500.12116/843.

[Agr+02] Aditya Agrawal et al. “Generative Programming via Graph Transformations in the Model-Driven Architecture”. In: *Workshop on Generative Techniques in the Context of Model Driven Architecture, OOPSLA*. 2002.

[Ajm+94] Marco Ajmone Marsan et al. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1994. ISBN: 0-471-93059-8.

[AK01] Colin Atkinson and Thomas Kühne. “The Essence of Multilevel Metamodeling”. In: *UML*. Vol. 2185. LNCS. Springer, 2001, pp. 19–33. DOI: 10.1007/3-540-45441-1_3.

[Ale+09] Aldeida Aleti et al. “ArcheOpterix: An extendable tool for architecture optimization of AADL models”. In: *MOMPES*. IEEE, 2009, pp. 61–71. DOI: 10.1109/MOMPES.2009.5069138.

[Alk+20] Bader Alkhazi et al. “Multi-criteria test cases selection for model transformations”. In: *Autom. Softw. Eng.* (2020). DOI: 10.1007/s10515-020-00271-w.

[Ana+10] Kyriakos Anastasakis et al. “On challenges of model transformation from UML to Alloy”. In: *Softw. Syst. Model.* 9.1 (2010), pp. 69–86.

[Anj+14] Anthony Anjorin et al. “Efficient Model Synchronization with View Triple Graph Grammars”. In: *ECMFA 2014*. Springer, 2014. DOI: 10.1007/978-3-319-09195-2_1.

[Anj+17] Anthony Anjorin et al. “BenchmarX Reloaded: A Practical Benchmark Framework for Bidirectional Transformations”. In: *BX@ETAPS*. Vol. 1827. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 15–30. URL: http://ceur-ws.org/Vol-1827/paper12.pdf.

[APV09] Saswat Anand, Corina S. Păsăreanu, and Willem Visser. “Symbolic execution with abstraction”. In: *Int. J. Softw. Tools Technol. Transf.* 11.1 (2009), pp. 53–67. DOI: 10.1007/s10009-008-0090-1.

[Arc+18] Davide Arcelli et al. “EASIER: An Evolutionary Approach for Multi-objective Software ArchItecturE Refactoring”. In: *ISCA*. IEEE, 2018, pp. 105–114. DOI: 10.1109/ICSA.2018.00020.

[Are+10] Thorsten Arendt et al. “Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations”. In: *MODELS 2010*. Springer, 2010, pp. 121–135. DOI: 10.1007/978-3-642-16145-2_9.

[Bak+13] Kacper Bak et al. “Clafer: unifying class and feature modeling”. In: *Softw. Syst. Model.* (2013), pp. 1–35.

[Bal+10] Clément Ballabriga et al. “OTAWA: An Open Toolbox for Adaptive WCET Analysis”. In: *Software Technologies for Embedded and Ubiquitous Systems*. LNCS 6399. Springer, 2010, pp. 35–46. DOI: 10.1007/978-3-642-16256-5_6.

[Bal+15] Paolo Ballarini et al. “HASL: A new approach for performance evaluation and model checking from concepts to experimentation”. In: *Perform. Eval.* 90 (2015), pp. 53–77. DOI: 10.1016/j.peva.2015.04.003.

[Ban+13] Kshitij Bansal et al. “Structural Counter Abstraction”. In: *TACAS 2013*. Vol. 7795. LNCS. Springer, 2013, pp. 62–77. DOI: 10.1007/978-3-642-36742-7_5.

[Bar+12] Edd Barrett et al. “Virtual machine warmup blows hot and cold”. In: *Proc. ACM Program. Lang.* 1.OOPSLA (2012). Article No.: 52. DOI: 10.1145/3133876.

[Bar+18] Ezio Bartocci et al. “Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications”. In: *Lectures on Runtime Verification*. Springer, 2018, pp. 135–175.

[BBF09] Gordon S. Blair, Nelly Bencomo, and Robert B. France. “Models@run.time”. In: *IEEE Computer* 42.10 (2009), pp. 22–27. DOI: 10.1109/MC.2009.326.

[BC12] Fabian Büttner and Jordi Cabot. “Lightweight String Reasoning for OCL”. In: *ECMFA*. Vol. 7349. LNCS. Springer, 2012, pp. 244–258.

[BCK01] Paolo Baldan, Andrea Corradini, and Barbara König. “A Static Analysis Technique for Graph Transformation Systems”. In: *CONCUR*. Vol. 2154. LNCS. Springer, 2001, pp. 381–295. DOI: 10.1007/3-540-44685-0_26.

[BDD04] Simona Bernardi, Susanna Donatelli, and Giovanna Dondossola. “Towards a Methodological Approach to Specification and Analysis of Dependable Automation Systems”. In: Springer, 2004, pp. 36–51. DOI: 10.1007/978-3-540-30206-3_5.

[Bea+10] Olivier Beaudoux et al. “Active Operations on Collections”. In: *MODELS 2010*. Springer, 2010. DOI: 10.1007/978-3-642-16145-2_7.

[Bel77] Nuel D. Belnap. “A Useful Four-Valued Logic”. In: *Modern Uses of Multiple-Valued Logic*. Springer, 1977, pp. 5–37. DOI: 10.1007/978-94-010-1161-7_2.

[Ben+15] Amine Benelallam et al. “Distributed model-to-model transformation with ATL on MapReduce”. In: *SLE*. ACM, 2015, pp. 37–48. DOI: 10.1145/2814251.2814258.

[Ben+18] R. Ben Abdessalem et al. “Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms”. In: *ICSE*. 2018, pp. 1016–1026. DOI: 10.1145/3180155.3180160.

[Ber+11] Gábor Bergmann et al. “A Graph Query Language for EMF models”. In: *ICMT*. LNCS 6707. Springer, 2011, pp. 167–182. DOI: 10.1007/978-3-642-21732-6_12.

[Ber+12a] Gábor Bergmann et al. “Change-driven model transformations”. In: *Softw. Syst. Model.* 11.3 (2012), pp. 431–461. DOI: 10.1007/s10270-011-0197-9.

[Ber+12b] Gábor Bergmann et al. “Incremental Pattern Matching for the Efficient Computation of Transitive Closure”. In: *ICGT 2012*. Springer, 2012, pp. 386–400. DOI: 10.1007/978-3-642-33654-6.

[Ber+15] Gábor Bergmann et al. “VIATRA 3: A Reactive Model Transformation Platform”. In: *ICMT*. Springer, 2015, pp. 101–110.

[BFK19] Axel Busch, Dominik Fuchss, and Anne Koziolek. "PerOpteryx: Automated Improvement of Software Architectures". In: *ICSA*. IEEE, 2019, pp. 162–165. DOI: 10.1109/ICSA-C.2019.00036.

[BFL17] Clément Ballabriga, Julien Forget, and Giuseppe Lipari. "Symbolic WCET computation". In: *ACM Trans. Embedded Comput. Syst.* 17.2 (2017). DOI: 10.1145/3147413.

[BFS00] Peter Buneman, Mary Fernandez, and Dan Suciu. "UnQL: a query language and algebra for semistructured data based on structural recursion". In: *VLDB J.* 9.1 (2000). DOI: 10.1007/s007780050084.

[BG01] Jean Bézivin and Olivier Gerbé. "Towards a Precise Definition of the OMG/MDA Framework". In: *Proc. 16th IEEE Int. Conf. Autom. Softw. Eng.* IEEE, 2001. DOI: 10.1109/ASE.2001.989813.

[BHZ08] Roberto Bagnara, Particia M. Hill, and Enea Zaffanella. "The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems". In: *Sci. Comput. Program.* 72.1-2 (2008), pp. 3–21. DOI: 10.1016/j.scico.2007.08.001.

[BKR08] Steffen Becker, Heiko Koziolek, and Ralf Reussner. "The Palladio component model for model-driven performance prediction". In: *J. Sys. Softw.* 82.1 (2008), pp. 3–22. DOI: 10.1016/j.jss.2008.03.066.

[BKS02] B. Beckert, U. Keller, and P. H. Schmitt. "Translating the Object Constraint Language into First-order Predicate Logic". In: *Proc. VERIFY, Workshop at FLoC*. 2002.

[BMM99] Andrea Bondavalli, Ivan Mura, and István Majzik. "Automatic Dependability Analysis for Supporting Design Decisions in UML". In: *Proc. 4th IEEE Int. Symp. High-Assur. Syst. Eng.* IEEE, 1999, pp. 64–74. DOI: 10.1109/HASE.1999.809476.

[BMP12] Simona Bernardi, José Merseguer, and Dorina C. Petriu. "Dependability modeling and analysis of software systems specified with UML". In: *ACM Comput. Surv.* 45.1 (2012). DOI: 10.1145/2379776.2379778.

[BPF15] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. "νZ – An Optimizing SMT Solver". In: *TACAS*. Vol. 9035. LNCS. Springer, 2015, pp. 194–199. DOI: 10.1007/978-3-662-46681-0_14.

[Bra+13] Martin Brain et al. "An Abstract Interpretation of DPLL(T)". In: *VMCAI 2013*. LNCS 7737. Springer, 2013, pp. 455–475. DOI: 10.1007/978-3-642-35873-9_27.

[Bru+15] Hugo Brunelière et al. "EMF Views: A View Mechanism for Integrating Heterogeneous Models". In: *ER 2015*. Springer, 2015, pp. 317–325. DOI: 10.1007/978-3-319-25264-3_23.

[Bru+17] Hugo Brunelière et al. "A feature-based survey of model view approaches". In: *Softw. Syst. Model.* (2017). DOI: 10.1007/s10270-017-0622-9.

[BSC10] David Benavides, Sergio Segura, and Antonio Ruiz Cortés. "Automated analysis of feature models 20 years later: A literature review". In: *Inf. Syst.* 35.6 (2010), pp. 615–636. DOI: 10.1016/j.is.2010.01.001.

[Bur+04] Sven Burmester et al. "Incremental design and formal verification with UML/RT in the FUJABA real-time tool suite". In: *Proceedings of the International Workshop on Specification and Validation of UML Models for Real Time and Embedded Systems, SVERTS2004*. 2004.

[Bur+05] Sven Burmester et al. "Worst-case execution time optimization of story patterns for hard real-time systems". In: *3rd International Fujaba Days*. 2005, pp. 71–78.

[Búr+18] Márton Búr et al. "Distributed graph queries for runtime monitoring of cyber-physical systems". In: LNCS 10802. 2018, pp. 111–128. DOI: 10.1007/978-3-319-89363-1_7.

[Búr+20] Márton Búr et al. "Distributed graph queries over models@run.time for runtime monitoring of cyber-physical systems". In: *Int. J. Softw. Tools Technol. Transf.* 22 (2020), pp. 79–102. DOI: 10.1007/s10009-019-00531-5.

[Búr21] Márton Búr. "Query-Based Runtime Monitoring in Real-Time and Distributed Systems". PhD thesis. McGill University, 2021. URL: https://imbur.github.io/phd/marton-bur-thesis.pdf.

[Büt+12] Fabian Büttner et al. “Verification of ATL Transformations Using Transformation Models and Model Finders”. In: *ICFEM*. Springer, 2012, pp. 198–213.

[BZJ21] Alexandru Burdusel, Steffen Zschaler, and Stefan John. “Automatic generation of atomic multiplicity-preserving search operators for search-based model engineering”. In: *Softw. Syst. Model.* 20.6 (2021), pp. 1857–1887. DOI: 10.1007/s10270-021-00914-w.

[BZS18] Alexandru Burdusel, Steffen Zschaler, and Daniel Strüber. “MDEoptimiser: A Search Based Model Engineering Tool”. In: *MODELS*. ACM, 2018, pp. 12–16. DOI: 10.1145/3270112.3270130.

[CA05] Krzysztof Czarnecki and Michał Antkiewicz. “Mapping Features to Models: A Template Approach Based on Superimposed Variants”. In: *GPCE*. Springer, 2005, pp. 422–437. DOI: 10.1007/11561347_28.

[Cal+12] Radu Calinescu et al. “Self-Adaptive Software Needs Quantitative Verification at Runtime”. In: *Comm. ACM* 55.9 (2012), pp. 69–77. DOI: 10.1145/2330667.2330686.

[Cal+17] Radu Calinescu et al. “RODES: A Robust-Design Synthesis Tool for Probabilistic Systems”. In: *QEST*. Vol. 10503. LNCS. Springer, 2017, pp. 304–308. DOI: 10.1007/978-3-319-66335-7_20.

[CB02] Antoine Colin and Guillem Bernat. “Scope-tree: A program representation for symbolic worst-case execution time analysis”. In: *Proceedings 14th Euromicro Conference on Real-Time Systems. Euromicro RTS 2002*. IEEE, 2002, pp. 50–59.

[Cbc] COIN-OR. *Cbc*. URL: https://github.com/coin-or/Cbc.

[CCR07] Jordi Cabot, Robert Clarisó, and Daniel Riera. “UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming”. In: *ASE*. ACM, 2007, pp. 547–548. DOI: 10.1145/1321631.1321737.

[CCR08] J. Cabot, R. Clariso, and D. Riera. “Verification of UML/OCL Class Diagrams using Constraint Programming”. In: *ICSTW*. 2008, pp. 73–80.

[CCR14] Jordi Cabot, Robert Clarisó, and Daniel Riera. “On the verification of UML/OCL class diagrams using constraint programming”. In: *J. Syst. Softw.* 93 (2014), pp. 1–23. DOI: 10.1016/j.jss.2014.03.023.

[CET18] Vittorio Cortellessa, Romina Eramo, and Michele Tucci. “Availability-Driven Architectural Change Propagation Through Bidirectional Model Transformations Between UML and Petri Net Models”. In: *ICSA*. IEEE, 2018. DOI: 10.1109/ICSA.2018.00022.

[CET20] Vittorio Cortellessa, Romina Eramo, and Michele Tucci. “From software architecture to analysis models and back: Model-driven refactoring aimed at availability improvement”. In: *Inf. Softw. Technol.* 127 (2020), p. 106362. DOI: 10.1016/j.infsof.2020.106362.

[CGC19] Robert Clarisó, Carlos A. González, and Jordi Cabot. “Smart Bound Selection for the Verification of UML/OCL Class Diagrams”. In: *IEEE Trans. Softw. Eng.* 45.4 (2019), pp. 412–426. DOI: 10.1109/TSE.2017.2777830.

[CH06] K. Czarnecki and S. Helsen. “Feature-based survey of model transformation approaches”. In: *IBM Syst. J.* 45.3 (2006), pp. 621–645. DOI: 10.1147/sj.453.0621.

[CH78] Patrick Cousot and Nicolas Halbwachs. “Automatic discovery of linear restraints among variables of a program”. In: *POPL*. ACM, 1978, pp. 84–96. DOI: 10.1145/512760.512770.

[Che+11a] Taolue Chen et al. “Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications”. In: *Log. Method. Comput. Sci.* 7.1 (2011). DOI: 10.2168/LMCS-7(1:12)2011.

[Che+11b] Betty H. C. Cheng et al. “Using Models at Runtime to Address Assurance for Self-Adaptive Systems”. In: *Models@run.time*. LNCS 8378. Springer, 2011, pp. 101–136. DOI: 10.1007/978-3-319-08915-7_4.

[Cic+10] Antonio Cicchetti et al. “JTL: A Bidirectional and Change Propagating Transformation Language”. In: *SLE 2010*. Springer, 2010, pp. 183–202. DOI: 10.1007/978-3-642-19440-5_11.

[CJ11] Duc Hiep Chu and Joxan Jaffar. "Symbolic simulation on complicated loops for WCET path analysis". In: *Proc. of the 9th ACM International Conference on Embedded Software*. IEEE, 2011, pp. 319–328. DOI: 10.1145/2038642.2038692.

[CK96] Kong-Rim Choi and Kyung-Chang Kim. "T*-tree: a main memory database index structure for real time applications". In: *3rd International Workshop on Real-Time Computing Systems and Applications*. IEEE, 1996, pp. 81–88. DOI: 10.1109/RTCSA.1996.554964.

[Clp] COIN-OR. *Clp*. URL: https://github.com/coin-or/Clp.

[CMP15] Vittorio Cortellessa, Raffaela Mirandola, and Pasqualina Potena. "Managing the evolution of a software architecture at minimal cost under performance and reliability constraints". In: *Sci. Comput. Program.* 98 (2015), pp. 439–463. DOI: 10.1016/j.scico.2014.06.001.

[CNS12] Marsha Chechik, Shiva Nejati, and Mehrad Sabetzadeh. "A relationship-based approach to model integration". In: *Innov. Syst. Softw. Eng.* 8.1 (2012), pp. 3–18. DOI: 10.1007/s11334-011-0155-2.

[CS06] Hugues Cassé and Pascal Sainrat. "OTAWA, a framework for experimenting WCET computations". In: *3rd European Congress on Embedded Real-Time*. 2006, pp. 1–8.

[CT93] Gianfranco Ciardo and Kishor S. Trivedi. "A decomposition approach for stochastic reward net models". In: *Perform. Eval.* 18.1 (1993), pp. 37–59. DOI: 10.1016/0166-5316(93)90026-Q.

[Cuc+12] L. Cucu-Grosjean et al. "Measurement-Based Probabilistic Timing Analysis for Multi-path Programs". In: *2012 24th Euromicro Conference on Real-Time Systems*. 2012, pp. 91–101. DOI: 10.1109/ECRTS.2012.31.

[DBB18] Wei Dou, Domenico Bianculli, and Lionel Briand. "Model-Driven Trace Diagnostics for Pattern-Based Temporal Specifications". In: *21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 2018, pp. 278–288. DOI: 10.1145/3239372.3239396.

[Deb+14] Csaba Debreceni et al. "Query-driven incremental synchronization of view models". In: *VAO '14*. ACM, 2014, pp. 31–38. DOI: 10.1145/2631675.2631677.

[DGC14] Juan De Lara, Ester Guerra, and Jesús Sánchez Cuadrano. "When and How to Use Multilevel Modelling". In: 24.2 (2014), pp. 1–46. DOI: 10.1145/2685615.

[DHS09] Susanna Donatelli, Serge Haddad, and Jeremy Sproston. "Model Checking Timed and Stochastic Properties with CSL^{TA}". In: *IEEE Tran. Softw. Eng.* 35.2 (2009), pp. 224–240. DOI: 10.1109/TSE.2008.108.

[DLL62] Martin Davis, George Logemann, and Davis Loveland. "A machine program for theorem-proving". In: *C. ACM* 5.7 (1962), pp. 394–397. DOI: 10.1145/368273.368557.

[Dru00] Doron Drusinsky. "The temporal rover and the ATG rover". In: *SPIN Model Checking and Software Verification*. LNCS 1885. 2000, pp. 323–330. DOI: 10.1007/10722468_19.

[DXC11] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. "Specifying Overlaps of Heterogeneous Models for Global Consistency Checking". In: *MODELS 2010*. Springer, 2011, pp. 165–179. DOI: 10.1007/978-3-642-21210-9_16.

[Edu+18] Sergey Edunov et al. "Generating Synthetic Social Graphs with Darwini". In: *ICDCS*. IEEE, 2018, pp. 567–577. DOI: 10.1109/ICDCS.2018.00062.

[Ehr+06] H. Ehrig et al. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006. ISBN: 978-3-540-31187-4. DOI: 10.1007/3-540-31188-2.

[Ehr+99] H. Ehrig et al., eds. *Handbook of Graph Grammars and Computing by Graph Transformation*. Vol. 1. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1999, pp. 163–246.

[Eme11] Daniel Emery. "Headways on high speed lines". In: *9th World Congress on Railway Research*. 2011, pp. 22–26.

[Eng+97] Gregor Engels et al. "A Combined Reference Model- and View-Based Approach to System Specification". In: *Int. J. Softw. Eng. Knowl. Eng.* 7.4 (1997), pp. 457–477. DOI: 10.1142/S0218194097000266.

[ENS10] Brandon K. Eames, Sandeep Neema, and Rohit Saraswat. "DesertFD: a finite-domain constraint based tool for design space exploration". In: *Des. Autom. Embed. Syst.* 14.2 (2010), pp. 43–74. DOI: 10.1007/s10617-009-9049-z.

[Epi+09] Ilenia Epifani et al. "Model evolution by run-time parameter adaptation". In: *ICSE*. IEEE, 2009, pp. 111–121. DOI: 10.1109/ICSE.2009.5070513.

[Erm+07] Andreas Ermedahl et al. "Loop bound analysis based on a combination of program slicing, abstract interpretation, and invariant analysis". In: *7th International Workshop on Worst-Case Execution Time Analysis (WCET'07)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[ES03] Niklas Eén and Niklas Sörensson. "An extensible SAT-solver". In: *ICTAST*. Springer. 2003, pp. 502–518.

[Fam+13] Michalis Famelis et al. "Transformation of models containing uncertainty". In: *MODELS*. Springer. 2013, pp. 673–689.

[FB16] Simon Fürst and Markus Bechter. "AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform". In: *DSN-W*. IEEE, 2016. DOI: 10.1109/DSN-W.2016.24.

[FD16] Peter Feiler and Julien Delange. "Automated Fault Tree Analysis from ADL Models". In: *ACM SIGAda Ada Letters* 36.2 (2016), pp. 39–46. DOI: 10.1145/3092893.3092900.

[Fel03] Massimo Felici. "Taxonomy of evolution and dependability". In: *Proc. 2nd Workshop Unanticip. Softw. Evol.* 2003, pp. 95–104.

[FFJ12] Pietro Ferrara, Rafael Fuchs, and Uri Juhasz. "TVAL+: TVLA and value analyses together". In: *SEFM*. ACM, 2012, pp. 63–77. DOI: 10.1007/978-3-642-33826-7_5.

[FH04] Christian Ferdinand and Reinhold Heckmann. "aiT: Worst-Case Execution Time Prediction by Static Program Analysis". In: *Building the Information Society*. IFIPAICT 156. Springer, 2004, pp. 377–383. DOI: 10.1007/978-1-4020-8157-6_29.

[Fis+98] Thorsten Fischer et al. "Story diagrams: A new graph rewrite language based on the unified modeling language and java". In: *International Workshop on Theory and Application of Graph Transformations*. Springer. 1998, pp. 296–309. DOI: 10.1007/978-3-540-46464-8_21.

[FSC12] Michais Famelis, Rick Salay, and Marsha Chechik. "Partial models: Towards modeling and reasoning with uncertainty". In: *ICSE '12*. IEEE, 2012. DOI: 10.1109/ICSE.2012.6227159.

[FTW16] Martin Fleck, Javier Troya, and Manuel Wimmer. "Search-Based Model Transformations with MOMoT". In: *ICMT@STAF*. LNCS 9765. Springer, 2016, pp. 79–87. DOI: 10.1007/978-3-319-42064-6_6.

[Gal06] Brian Gallagher. "Matching structure and semantics: A survey on graph-based pattern matching". In: *AAAI FS* 6 (2006), pp. 45–53.

[GBR05] Martin Gogolla, Jörn Bohling, and Mark Richters. "Validating UML and OCL models in USE by automatic snapshot generation". In: *Softw. Syst. Model.* 4 (2005), pp. 386–398.

[GDM14] Hamid Gholizadeh, Zinovy Diskin, and Tom Maibaum. "A Query Structured Approach for Model Transformation". In: *Workshop on Analysis of Model Transformations*. Vol. 1277. CEUR Workshop Proceedings. CEUR-WS.org, 2014, pp. 54–63. URL: http://ceur-ws.org/Vol-1277/6.pdf.

[Get+18] Sinem Getir et al. "Supporting semi-automatic co-evolution of architecture and fault tree models". In: *J. Syst. Softw.* 142 (2018), pp. 115–135. DOI: 10.1016/j.jss.2018.04.001.

[Gha+17] Majdi Ghadhab et al. "Model-Based Safety Analysis for Vehicle Guidance Systems". In: *SAFECOMP*. Springer, 2017, pp. 3–19. DOI: 10.1007/978-3-319-66266-4_1.

[GHL14] Holger Giese, Stephan Hildebrandt, and Leen Lambers. "Bridging the gap between formal semantics and implementation of triple graph grammars". In: *Softw. Syst. Model.* 13.1 (2014), pp. 273–299. DOI: 10.1007/s10270-012-0247-y.

[Gie+03] Holger Giese et al. "Towards the compositional verification of real-time UML designs". In: *ACM SIGSOFT Symposium on the Foundations of Software Engineering*. 2003, pp. 38–47. DOI: 10.1145/940071.940078.

[Gil+10] Stephen Gilmore et al. "Non-functional properties in the model-driven development of service-oriented systems". In: *Software & Systems Modeling* 10.3 (2010), pp. 287–311. ISSN: 1619-1366. DOI: 10.1007/s10270-010-0155-y.

[GK07] Joel Greenyer and Ekkart Kindler. "Reconciling TGGs with QVT". In: *MODELS 2007*. Springer, 2007, pp. 16–30. DOI: 10.1007/978-3-540-75209-7_2.

[Gon+12] Carlos A. González et al. "EMFtoCSP: a tool for the lightweight verification of EMF models". In: *FormSERA*. 2012, pp. 44–50. DOI: 10.1109/FormSERA.2012.6229788.

[Gop+04] Denis Gopan et al. "Numeric Domains with Summarized Dimensions". In: *TACAS 2004*. Vol. 2988. LNCS. Springer, 2004, pp. 512–529. DOI: 10.1007/978-3-540-24730-2_38.

[Gre06] Joel Greenyer. "A study of technologies for model transformation: Reconciling TGGs with QVT". Diplomarbeit. Universität Paderborn, 2006.

[GRR09] Hans Grönniger, Jan Oliver Ringert, and Bernhard Rumpe. "System Model-Based Definition of Modeling Language Semantics". In: *FORTE*. Vol. 5522. LNCS. Springer, 2009, pp. 152–166.

[GTC15] Simos Gerasimou, Giordano Tamburrelli, and Radu Calinescu. "Search-Based Synthesis of Probabilistic Models for Quality-of-Service Software Engineering". In: *ASE*. IEEE, 2015. DOI: 10.1109/ASE.2015.22.

[Gus+06] Jan Gustafsson et al. "Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution". In: *Proc. of Real-Time Systems Symposium*. 2006, pp. 57–66. DOI: 10.1109/RTSS.2006.12.

[Hal18] Terry A. Halpin. "Object-Role Modeling". In: *Encyclopedia of Database Systems, Second Edition*. Ed. by Ling Liu and M. Tamer Özsu. Springer, 2018. DOI: 10.1007/978-1-4614-8265-9_251.

[Har+19] Thomas Hartmann et al. "GREYCAT: Efficient what-if analytics for data in motion at scale". In: *Information Systems* 83 (2019), pp. 101–117.

[Hav15] Klaus Havelund. "Rule-based runtime verification revisited". In: *Int. J. Software Tools Technol. Trans.* 17.2 (2015), pp. 143–170. DOI: 10.1007/s10009-014-0309-2.

[Heg+16] Ábel Hegedüs et al. "Query-driven soft traceability links for models". In: *Softw. Syst. Model.* 15.3 (2016), pp. 733–756. DOI: 10.1007/s10270-014-0436-y.

[Her+17] Sebastian J. I. Herzig et al. "Model-transformation-based computational design synthesis for mission architecture optimization". In: *IEEE Aerospace Conference*. IEEE, 2017. DOI: 10.1109/AERO.2017.7943953.

[HHM09] Jeffery Hansen, Scott Hissam, and Gabriel A Moreno. "Statistical-based wcet estimation and validation". In: *9th International Workshop on Worst-Case Execution Time Analysis*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2009.

[HHV15] Ábel Hegedüs, Ákos Horváth, and Dániel Varró. "A model-driven framework for guided design space exploration". In: *Autom. Softw. Eng.* 22 (2015), pp. 399–436. DOI: 10.1007/s10515-014-0163-1.

[Hid+11] Soichiro Hidaka et al. "GRoundTram: An integrated framework for developing well-behaved bidirectional model transformations". In: *ASE 2011*. IEEE, 2011. DOI: 10.1109/ASE.2011.6100104.

[Hil16] Nicolas Hili. "A Metamodeling Framework for Promoting Flexibility and Creativity Over Strict Model Conformance". In: *FlexMDE@MODELS*. Vol. 1694. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 2–11. URL: http://ceur-ws.org/Vol-1694/FlexMDE2016%5C_paper%5C_6.pdf.

[HLR06] David Hearnden, Michael Lawley, and Kerry Raymond. "Incremental Model Transformation for the Evolution of Model-Driven Systems". In: *MODELS 2006*. Springer, 2006, pp. 321–335. DOI: 10.1007/11880240_23.

[HM19] Dávid Honfi and Zoltán Micskei. "Classifying generated white-box tests: an exploratory study". In: *Softw. Qual. J.* 27 (2019), pp. 1339–1380. DOI: 10.1007/s11219-019-09446-5.

[Hoa+17] Xuan-Luu Hoang et al. "Generation and impact analysis of adaptation options for automated manufacturing machines". In: *22nd IEEE Int. Conf. Emerg. Technol. Fact. Autom.* IEEE, 2017. DOI: 10.1109/ETFA.2017.8247572.

[HOT89] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeo K. Taneja. "Processing Aggregate Relational Queries with Hard Time Constraints". In: *ACM SIGMOD Rec.* 18.2 (1989), pp. 68–77. DOI: 10.1145/66926.66933.

[HR02] Klaus Havelund and Grigore Rosu. "Synthesizing Monitors for Safety Properties". In: *Tools and Algorithms for the Construction and Analysis of Systems*. LNCS 2280. 2002, pp. 342–356. DOI: 10.1007/3-540-46002-0_24.

[HR09] Jörg Herter and Jan Reineke. "Making Dynamic Memory Allocation Static to Support WCET Analysis". In: *9th International Workshop on Worst-Case Execution Time Analysis*. 2009.

[HS17] Nicolas Hili and Jean-Sébastien Sottet. "The Conformance Relation Challenge: Building Flexible Modelling Frameworks". In: *FlexMDE@MODELS*. Vol. 2019. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 418–423. URL: http://ceur-ws.org/Vol-2019/flexmde%5C_6.pdf.

[HT16] Sochiro Hidaka and Massimo Tisi. *Partial Bidirectionalization of Model Transformation Languages*. Tech. rep. 2016. URL: https://hidaka.cis.k.hosei.ac.jp/research/papers/scp2016.pdf.

[Iqb+15] Muhammad Zohaib Iqbal et al. "Applying UML/MARTE on industrial projects: challenges, experiences, and guidelines". In: *Softw. Syst. Model.* 14.4 (2015), pp. 1367–1385. DOI: 10.1007/s10270-014-0405-5.

[IW17] M. Usman Iftikhar and Danny Weyns. "ActivFORMS: A Runtime Environment for Architecture-Based Adaptation with Guarantees". In: *ICSAW*. IEEE, 2017. DOI: 10.1109/ICSAW.2017.21.

[Jac02] Daniel Jackson. "Alloy: a lightweight object modelling notation". In: *ACM Trans. Softw. Eng. Methodol.* 11.2 (2002), pp. 256–290. DOI: 10.1145/505145.505149.

[JDR17] Axel Jantsch, Nikil Dutt, and Amir M. Rahmani. "Self-awareness in systems on chip – A survey". In: *IEEE Design & Test* 34.6 (2017), pp. 8–26.

[JGS13] Ethan K. Jackson, Simko Gabor, and Janos Sztipanovits. *Diversely Enumerating System-Level Architectures*. Tech. rep. MSR-TR-2013-56. 2013.

[JKS06] Johannes Jakob, Alexander Königs, and Andy Schürr. "Non-materialized Model View Specification with Triple Graph Grammars". In: *ICGT 2006*. Springer, 2006, pp. 321–355. DOI: 10.1007/11841883_23.

[JLB11] Ethan K Jackson, Tihamer Levendovszky, and Daniel Balasubramanian. "Reasoning about metamodeling with formal specifications and automatic proofs". In: *MODELS*. Springer, 2011, pp. 653–667.

[JM09] Bertrand Jeannet and Antoine Miné. "Apron: A Library of Numerical Abstract Domains for Static Analysis". In: *CAV*. LNTCS 5643. Springer, 2009, pp. 661–667. DOI: 10.1007/978-3-642-02658-4_52.

[JN16] Manfred A. Jeusfeld and Bernd Neumayr. "DeepTelos: Multi-level Modeling with Most General Instances". In: *ER*. Vol. 9974. LNCS. 2016, pp. 198–211. DOI: 10.1007/978-3-319-46397-1_15.

[Joh+19] Stefan John et al. "Searching for Optimal Models: Comparing Two Encoding Approaches". In: *J. Object Technol.* 18.3 (2019), 6:1–22. DOI: 10.5381/jot.2019.18.3.a6.

[Jou+08] Frédéric Jouault et al. "ATL: A Model Transformation Tool". In: *Sci. Comput. Program.* 72.1-2 (2008), pp. 31–39. DOI: 10.1016/j.scico.2007.08.002.

[JS06] Ethan K. Jackson and Janos Sztipanovits. "Towards a Formal Foundation for Domain Specific Modeling Languages". In: *EMSOFT*. Seoul, Korea: ACM, 2006, pp. 53–62.

[JS07] Ethan K Jackson and Janos Sztipanovits. "Constructive techniques for meta- and model-level reasoning". In: *MODELS*. Springer, 2007, pp. 405–419.

[JSS13] Ethan K. Jackson, Gabor Simko, and Janos Sztipanovits. "Diversely enumerating system-level architectures". In: *ACM International Conference on Embedded Software*. IEEE, 2013.

[JT10] Frédéric Jouault and Massimo Tisi. "Towards Incremental Execution of ATL Transformations". In: *ICMT 2010*. Springer, 2010, pp. 123–137. DOI: 10.1007/978-3-642-13688-7_9.

[Jür03] Jan Jürjens. "Developing safety-critical systems with UML". In: LNCS 2863. 2003, pp. 360–372. DOI: 10.1007/978-3-540-45221-8_31.

[JVB17] Anjali Joshi, Steve Vestal, and Pam Binns. "Automatic generation of static fault trees from AADL models". In: *DSN Workshops*. Springer, 2017.

[Kat+16] Guy Katz et al. "Lazy proofs for DPLL(T)-based SMT solvers". In: *FMCAD*. IEEE, 2016, pp. 93–100. DOI: 10.1109/FMCAD.2016.7886666.

[KB09] Heiko Koziolek and Franz Brosch. "Parameter Dependencies for Component Reliability Specifications". In: *Elec. Note. Theor. Comput. Sci.* 253 (1 2009), pp. 23–38. DOI: 10.1016/j.entcs.2009.09.026.

[Ker+13] Aleksandr A Kerzhner et al. "Architecting cellularized space systems using model-based design exploration". In: *AIAA SPACE 2013 Conference and Exposition*. 2013, p. 5371.

[KHG11] Mirco Kuhlmann, Lars Hamann, and Martin Gogolla. "Extensive Validation of OCL Models by Integrating SAT Solving into USE". In: *TOOLS*. Vol. 6705. LNCS. 2011, pp. 290–306.

[KJS10] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. "An Approach for Effective Design Space Exploration". In: *Monterey Workshop*. Vol. 6662. LNCS. Springer, 2010, pp. 33–54. DOI: 10.1007/978-3-642-21292-5_3.

[KK06] Barbara König and Vitali Kozioura. "Counterexample-Guided Abstraction Refinement for the Analysis of Graph Transformation Systems". In: *TACAS*. Vol. 3920. LNCS. Springer, 2006, pp. 197–211. DOI: 10.1007/11691372_13.

[KK08] Barbara König and Vitali Kozioura. "Augur 2 – A New Version of a Tool for the Analysis of Graph Transformation Systems". In: *Electr. Notes Theor. Comput. Sci.* 211 (2008). GT-VMT 2006, pp. 201–210.

[KKZ13] Jens Knoop, Laura Kovács, and Jakob Zwirchmayr. "WCET squeezing". In: *International Conference on Real-Time Networks and Systems*. ACM, 2013, pp. 161–170. DOI: 10.1145/2516821.2516847.

[KM04] Sarfraz Khurshid and Darko Marinov. "TestEra: Specification-Based Testing of Java Programs Using SAT". In: *Autom. Softw. Eng.* 11.4 (2004), pp. 403–434.

[KO17] Norihiro Kamide and Hitoshi Omori. "An Extended First-Order Belnap-Dunn Logic with Classical Negation". In: *LORI*. Springer, 2017, pp. 79–93. DOI: 10.1007/978-3-662-55665-8_6.

[Kol+13] Dimitrios S. Kolovos et al. "Programmatic Muddle Management". In: *XM@MODELS*. Vol. 1089. CEUR Workshop Proceedings. CEUR-WS.org, 2013, pp. 2–10. URL: http://ceur-ws.org/Vol-1089/1.pdf.

[Koz10] Heiko Koziolek. "Performance evaluation of component-based software systems: A survey". In: *Perform. Eval.* 67.8 (2010), pp. 634–658. DOI: 10.1016/j.peva.2009.07.007.

[Koz16] V. P. Kozyrev. "Estimation of the execution time in real-time systems". In: *Programming and Computer Software* 42.1 (2016), pp. 41–48. DOI: 10.1134/S0361768816010059.

[KP09] Ekkart Kindler and Laure Petrucci. "Towards a Standard for Modular Petri Nets: A Formalisation". In: *PETRI NETS*. 2009, pp. 43–62. DOI: 10.1007/978-3-642-02424-5_5.

[KPP] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. "The Epsilon Transformation Language". In: *ICMT 2008*. Springer, pp. 46–60. DOI: 10.1007/978-3-540-69927-9_4.

[KPP06] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. "Merging Models with the Epsilon Merging Language (EML)". In: Springer, 2006, pp. 215–229. DOI: 10.1007/11880240_16.

[KR08] Heiko Koziolek and Ralf Reussner. "A Model Transformation from the Palladio Component Model to Layered Queueing Networks". In: (2008), pp. 58–57. DOI: 10.1007/978-3-540-69814-2_6.

[KR11] Anne Koziolek and Ralf Reussner. "Towards a generic quality optimisation framework for component-based system models". In: *CBSE*. ACM, 2011, pp. 103–108. DOI: 10.1145/2000229.2000244.

[KS07] Thomas Kühne and Daniel Schreiber. "Can programming be liberated from the two-level style: multi-level programming with deepjava". In: *OOPSLA*. ACM, 2007, pp. 229–244. DOI: 10.1145/1297027.1297044.

[KSH12] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. "A Description Logic Primer". In: *CoRR* abs/1201.4089 (2012). arXiv: 1201.4089.

[Küh+10] Thomas Kühne et al. "Explicit Transformation Modeling". In: *MODELS 2009*. Springer, 2010, pp. 240–255. DOI: 10.1007/978-3-642-12261-3_23.

[Küh18] Thomas Kühne. "Exploring Potency". In: *MODELS*. ACM, 2018, pp. 2–12. DOI: 10.1145/3239372.3239411.

[Kul09] Ulrich W. Kulisch. "Complete Interval Arithmetic and Its Implementation of the Computer". In: *Numerical Validation in Current Hardware Architectures*. LNCS. Springer, 2009, pp. 7–26. DOI: 10.1007/978-3-642-01591-5_2.

[KZH16] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu. "BiGUL: a formally verified core language for putback-based bidirectional programming". In: *PEPM '16*. ACM, 2016, pp. 61–72. DOI: 10.1145/2847538.2847544.

[Lau+12] Marius Lauder et al. "Bidirectional Model Transformation with Precedence Triple Graph Grammars". In: *ECMFA 2012*. Springer, 2012, pp. 287–303. DOI: 10.1007/978-3-642-31491-9_22.

[LB16] S. Law and I. Bate. "Achieving Appropriate Test Coverage for Reliable Measurement-Based Timing Analysis". In: *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. 2016, pp. 189–199. DOI: 10.1109/ECRTS.2016.21.

[Leb+17] Erhan Leblebici et al. "Leveraging Incremental Pattern Matching Techniques for Model Synchronisation". In: *ICGT 2017*. Springer, 2017, pp. 179–195. DOI: 10.1007/978-3-319-61470-0_11.

[Lev06] Tihamér Levendovszky. "Applying metamodels in software model transformation methods". PhD thesis. Budapest University of Technology and Economics, 2006.

[LG10] Juan de Lara and Esther Guerra. "Deep Meta-modelling with MetaDepth". In: *TOOLS*. Vol. 6141. LNCS. Springer, 2010, pp. 1–20. DOI: 10.1007/978-3-642-13953-6_1.

[Li+07] Xianfeng Li et al. "Chronos: A timing analyzer for embedded software". In: *Science of Computer Programming* 69.1-3 (2007), pp. 56–67.

[Li+11] Rui Li et al. "An evolutionary multiobjective optimization approach to component-based software architecture design". In: *IEEE CEC*. IEEE, 2011, pp. 432–439. DOI: 10.1109/CEC.2011.5949650.

[Li+14] Yi Li et al. "Symbolic optimization with SMT solvers". In: *POPL*. ACM, 2014, pp. 607–618. DOI: 10.1145/2535838.2535857.

[Li+19] Xiaocui Li et al. "Aggregated multi-attribute query processing in edge computing for industrial IoT applications". In: *Computer Networks* 151 (2019), pp. 114–123. DOI: 10.1016/j.comnet.2019.01.022.

[Lia+15] Jia Hui (Jimmy) Liang et al. "SAT-based analysis of large real-world feature models is easy". In: *SPLC*. ACM, 2015, pp. 91–100. DOI: 10.1145/2791060.2791070.

[Lie+14] Grischa Liebel et al. "Assessing the State-of-Practice of Model-Based Engineering in the Embedded Systems Domain". In: *MODELS 2014*. Vol. 8767. LNCS. Springer, 2014, pp. 166–182. DOI: 10.1007/978-3-319-11653-2_11.

[Lim+95] Sung-Soo Lim et al. "An accurate worst case timing analysis for RISC processors". In: *IEEE Transactions on Software Engineering* 21.7 (1995), pp. 593–604.

[Lis14] Björn Lisper. "SWEET – a tool for WCET flow analysis". In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer. 2014, pp. 482–485.

[LLC05] Tihamér Levendovszky, László Lengyel, and Hassan Charaf. "A UML class diagram-based pattern language for model transformation systems". In: *SEPADS*. ACM, 2005. DOI: 10.5555/1365774.1365793.

[LM97] Y.-T.S. Li and Sharad Malik. "Performance analysis of embedded software using implicit path enumeration". In: *IEEE T. Comput. Aid. D.* 16.12 (1997), pp. 1477–1487. DOI: 10.1109/43.664229.

[LMC04] Juan Pablo López-Grao, José Merseguer, and Javier Campos. "From UML activity diagrams to Stochastic Petri nets: application to software performance engineering". In: *WOSP*. ACM, 2004, pp. 25–36. DOI: 10.1145/974043.974048.

[LP10] Daniel Le Berre and Anne Parrain. "The Sat4j library, release 2.2". In: *J. Satisfiability Boolean Model. Comput.* 7 (2010), pp. 59–64.

[Mag+07] Stephen Magill et al. "Arithmetic Strengthening for Shape Analysis". In: *SAS*. Vol. 4634. LNCS. Springer, 2007, pp. 419–436. DOI: 10.1007/978-3-540-74061-2_26.

[Mai+17] Claire Maiza et al. "The W-SEPT Project: Towards Semantic-Aware WCET Estimation". In: *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*. Vol. 57. OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. DOI: 10.4230/OASIcs.WCET.2017.9.

[Mar+10] Anne Martens et al. "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms". In: *Proc. 1st Joint WOSP/SIPEW Int. Conf. Perf. Eng.* ACM, 2010, pp. 105–116. DOI: 10.1145/1712605.1712624.

[Mar+98] Florian Martin et al. "Analysis of loops". In: *Lecture Notes in Computer Science* 1383 (1998), pp. 80–94. ISSN: 16113349.

[MB01] I. Mura and A. Bondavalli. "Markov Regenerative Stochastic Petri Nets to Model and Evaluate Phased Mission Systems Dependability". In: *IEEE Comput.* 50.12 (2001), pp. 1337–1351. DOI: 10.1109/TC.2001.970572.

[MB08] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *TACAS*. Vol. 4963. LNCS. Springer, 2008, pp. 337–340.

[MB15] Alexis Marechal and Didier Buchs. "Generalizing the Compositions of Petri Nets Modules". In: *Fundam. Inform.* 137.1 (2015), pp. 87–116. DOI: 10.3233/FI-2015-1171.

[MB99] I. Mura and A. Bondavalli. "Hierarchical modeling and evaluation of phased-mission systems". In: *IEEE Tran. Reliability* 48.4 (1999), pp. 360–368. DOI: 10.1109/24.814518.

[MCB84] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems". In: *ACM Trans. Comput. Syst.* 2.2 (1984). DOI: 10.1145/190.191.

[MCH16] David Mosteller, Lawrence Cabac, and Michael Haustermann. "Integrating Petri Net Semantics in a Model-Driven Approach: The Renew Meta-Modeling and Transformation Framework". In: *TOPNOC XI*. Vol. 9930. LNCS. Springer, 2016, pp. 92–113. DOI: 10.1007/978-3-662-53401-4_5.

[Mel+05] Sergey Melnik et al. "Supporting executable mappings in model management". In: *SIGMOD '05*. ACM, 2005, pp. 167–178. DOI: 10.1145/1066157.1066177.

[Men+17] Baoluo Meng et al. "Relational Constraint Solving in SMT". In: *CADE*. Vol. 10395. LNCS. Springer, 2017, pp. 148–165.

[Mey16] Bart Meyers. *A Multi-Paradigm Modelling Approach to Design and Evolution of Domain-Specific Modelling Languages*. 2016. URL: http://msdl.cs.mcgill.ca/people/bart/publ/thesis.pdf.

[Mez+19] Gergely Mezei et al. "Towards Flexible, Rigorous Refinement in Metamodeling". In: *MODELS-C*. IEEE, 2019, pp. 455–459. DOI: 10.1109/MODELS-C.2019.00073.

[Mic+12] Zoltán Micskei et al. "A Concept for Testing Robustness and Safety of the Context-Aware Behaviour of Autonomous Systems". In: *KES-AMSTA*. Vol. 7327. LNCS. Springer, 2012, pp. 504–513. DOI: 10.1007/978-3-642-30947-2_55.

[Mil+07] Aleksandar Milicevic et al. "Korat: A Tool for Generating Structurally Complex Test Inputs". In: *ICSE*. 2007, pp. 771–774. DOI: 10.1109/ICSE.2007.48.

[Mil+15] Aleksandar Milicevic et al. "Alloy*: A General-Purpose Higher-Order Relational Constraint Solver". In: *ICSE*. IEEE, 2015, pp. 609–619.

[Min04] Antoine Miné. "Weakly Relational Numerical Abstract Domains". PhD thesis. École Normale Supérieure, 2004.

[MOF] The Object Management Group. *Object Constraint Language, v2.5.1*. 2017. URL: https://www.omg.org/spec/MOF/2.5.1.

[MOG] The MOGENTES project. *Model-based generation of tests for dependable embedded systems, 7th EU Framework Programme*. 2011. URL: http://mogentes.eu/.

[MPB02] István Majzik, András Pataricza, and Andrea Bondavalli. "Stochastic Dependability Analysis of System Architecture Based on UML Models". In: *Architecting Dependable Systems*. Springer, 2002, pp. 219–244. DOI: 10.1007/3-540-45177-3_10.

[MRS10] Bill McCloskey, Thopas Reps, and Mooly Sagiv. "Statically Inferring Complex Heap, Array, and Numeric Invariants". In: *SAS*. Vol. 6337. LNCS. Springer, 2010, pp. 71–99. DOI: 10.1007/978-3-642-15769-1_6.

[MTD17] Salvador Martínez, Massimo Tisi, and Rémi Douence. "Reactive model transformation with ATL". In: *Sci. Comp. Prog.* 136 (2017), pp. 1–16. DOI: 10.1016/J.SCICO.2016.08.006.

[MVS07] Panagiotis Manolios, Daron Vroon, and Gayatri Subramanian. "Automating component-based system assembly". In: *ISSTA*. ACM, 2007, pp. 61–72. DOI: 10.1145/1273463.1273473.

[MW12] Maarten McKubre-Jordens and Zach Weber. "Real Analaysis in Paraconsistent Logic". In: *J. Phil. Logic* 41.5 (2012), pp. 901–922. DOI: 10.1007/s10992-011-9210-6.

[MWC09] Marcílio Mendonça, Andrzej Wasowski, and Krzysztof Czarnecki. "SAT-based analysis of feature models is easy". In: *SPLC*. ACM, 2009, pp. 231–240.

[Ndi+16] Moulaye Ndiaye et al. "Practical Use of Coloured Petri Nets for the Design and Performance Assessment of Distributed Automation Architectures". In: *PNSE*. CEUR-WS, 2016, pp. 113–131. URL: http://ceur-ws.org/Vol-1591/paper10.pdf.

[Nin15] Jordan Ninin. "Global Optimization Based on Contractor Programming: An Overview of the IBEX Library". In: *MACIS*. Vol. 9582. LNCS. Springer, 2015, pp. 555–559. DOI: 10.1007/978-3-319-32859-1_47.

[NIST17] Cyber-Physical Systems Public Working Group. *Framework for Cyber-Physical Systems: Volume 1, Overview*. NIST Special Publication 1500-201. National Institute of Standards and Technology, 2017. DOI: 10.6028/NIST.SP.1500-201.

[NNZ00] Ulrich Nickel, Jörg Niere, and Albert Zündorf. "The FUJABA environment". In: *ICSE*. 2000, pp. 742–745. DOI: 10.1145/337180.337620.

[NO79] Greg Nelson and Derek C. Oppen. "Simplification by Cooperating Decision Procedures". In: *ACM Trans. Program. Lang. Syst.* 1.2 (1979), pp. 245–257. DOI: 10.1145/357073.357079.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesar Tinelli. "Solving SAT and SAT Modulo Theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T)". In: *J. ACM* 53.6 (2006), pp. 937–977. DOI: 10.1145/1217856.1217859.

[OCL] The Object Management Group. *Object Constraint Language, v2.4*. 2014. URL: https://www.omg.org/spec/OCL/2.4.

[OS95] Gultekin Ozsoyoglu and Richard T. Snodgrass. "Temporal and real-time databases: A survey". In: *IEEE Trans. Knowl. Data Eng.* 7.4 (1995).

[Pek+20] Christian Pek et al. "Using online verification to prevent autonomous vehicles from causing accidents". In: *Nature Machine Intelligence* 2.9 (2020), pp. 518–528. DOI: 10.1038/s42256-020-0225-y.

[Pen08] Karl-Heinz Pennemann. "Resolution-like theorem proving for high-level conditions". In: *ICGT*. Vol. 5214. LNCS. Springer, 2008, pp. 289–304.

[Pik+10] Lee Pike et al. "Copilot: A Hard Real-Time Runtime Monitor". In: *Runtime Verification*. Vol. 6418. 2010, pp. 345–359. DOI: 10.1007/978-3-642-16612-9_26.

[PS97] Peter P. Puschner and Anton V. Schedl. "Computing Maximum Task Execution Times - A Graph-Based Approach". In: *Real-Time Systems* 13.1 (1997), pp. 67–91. DOI: 10.1023/A:1007905003094.

[Que+12] Anna Queralt et al. "OCL-Lite: Finite reasoning on UML/OCL conceptual schemas". In: *Data Knowl. Eng.* 73 (2012), pp. 1–22.

[QVT] Object Management Group. *MOF Query/View/Transformation Specification*. Version 1.3. 2016. URL: http://www.omg.org/spec/QVT/1.3/.

[RD06] Arend Rensink and Dino Distefano. "Abstract graph transformation". In: *Electr. Notes Theor. Comput. Sci.* 157.1 (2006), pp. 39–59.

[REJ09] Derek Rayside, H.-Cristian Estler, and Daniel Jackson. *The guided improvement algorithm for exact, general-purpose, many-objective combinatorial optimization*. Tech. rep. MIT-CSAIL-TR-2009-033. Massachsetts Institue of Technology, 2009.

[Ren+12] Arend Rensink et al. *User Manual for the GROOVE Tool Set*. 2012.

[Ren06] Arend Rensink. "Isomorphism Checking in GROOVE". In: *GaBaTS*. Vol. 4549. LNCS. Springer, 2006.

[Rey+13] Andrew Reynolds et al. "Finite Model Finding in SMT". In: *CAV*. Vol. 8044. LNCS. Springer, 2013, pp. 640–655. DOI: 10.1007/978-3-642-39799-8_42.

[Rie17] Leanna Rierson. *Developing Safety-Critical Software*. Vol. 7. 4. CRC Press, 2017, pp. 22–27. DOI: 10.1201/9781315218168.

[RST89] Andrew L. Reibman, Roger Smith, and Kishor S. Trivedi. "Markov and Markov reward model transient analysis: An overview of numerical approaches". In: *Eur. J. Oper. Res.* 4.2 (1989), pp. 257–267. DOI: 10.1016/0377-2217(89)90335-4.

[RSV17] Sebastian Rehberger, Lucas Spreiter, and Birgit Vogel-Heuser. "An agent-based approach for dependable planning of production sequences in automated production systems". In: *Automatisierungstechnik* 65.11 (2017), pp. 766–778. DOI: 10.1515/auto-2017-0040.

[RSW04] Thomas W Reps, Mooly Sagiv, and Reinhard Wilhelm. "Static program analysis via 3-valued logic". In: *CAV*. 2004, pp. 15–30.

[RTC11] RTCA, Inc. *DO-330 Sofware Tool Qualification Considerations*. 2011.

[SAB09] Seyyed M. A. Shah, Kyriakos Anastasakis, and Behzad Bordbar. "From UML to Alloy and back again". In: *MoDeVVa*. ACM, 2009.

[Sal+15] Rick Salay et al. "A Methodology for Verifying Refinements of Partial Models". In: *Journal of Object Technology* 14.3 (2015), 3:1–31.

[SB16] Jean-Sébastien Sottet and Nicolas Biri. "JSMF: a Javascript Flexible Modelling Framework". In: *FlexMD@MODELS*. Vol. 1694. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 42–51. URL: http://ceur-ws.org/Vol-1694/FlexMDE2016%5C_paper%5C_5.pdf.

[SC15] Rick Salay and Marsha Chechik. "A Generalized Formal Framework for Partial Modeling". English. In: *FASE*. Vol. 9033. LNCS. Springer Berlin Heidelberg, 2015, pp. 133–148.

[Sch+07] Pierre-Yves Schobbens et al. "Generic semantics of feature diagrams". In: *Comput. Networks* 51.2 (2007), pp. 456–479. DOI: 10.1016/j.comnet.2006.08.008.

[Sch95] Andy Schürr. "Specification of Graph Translators with Triple Graph Grammars". In: *WG 1994*. Springer, 1995, pp. 151–163. DOI: 10.1007/3-540-59071-4_45.

[Sch98] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & sons, 1998. ISBN: 978-0-471-98232-6.

[SE06] Mehrdad Sabetzadeh and Steve Easterbrook. "View merging in the presence of incompleteness and inconsistency". In: *Requir. Eng.* 11.3 (2006), pp. 174–193. DOI: 10.1007/s00766-006-0032-y.

[Sem+17] Oszkár Semeráth et al. "Formal Validation of Domain-Specific Languages with Derived Features and Well-Formedness Constraints". In: *Softw. Syst. Model* 16.2 (2017), pp. 357–392. DOI: 10.1007/s10270-015-0485-x.

[Sem+19] Oszkár Semeráth et al. "VIATRA Solver: A Framework for the Automated Generation of Consistent Domain-Specific Models". In: *ICSE Demo*. IEEE, 2019, pp. 43–46.

[Sem+20c] Oszkár Semeráth et al. "Diversity of Graph Models and Graph Generators in Mutation Testing". In: *Int. J. Softw. Tools Technol. Transf.* 22.1 (2020), pp. 57–78.

[SFC12] Rick Salay, Michalis Famelis, and Marsha Chechik. "Language Independent Refinement Using Partial Modeling". In: *FASE*. Vol. 7212. LNCS. Springer, 2012, pp. 224–239.

[SLO17] Sven Schneider, Leen Lambers, and Fernando Orejas. "Symbolic model generation for graph properties". In: *FASE*. Vol. 10202. LNCS. Springer, 2017, pp. 226–243.

[SNP13] Jaroslaw Skaruz, Artur Niewiadomski, and Wojciech Penczek. "Evolutionary Algorithms for Abstract Planning". In: *PPAM*. LNTCS 8384. Springer, 2013, pp. 392–401. DOI: 10.1007/978-3-642-55224-3_37.

[SNV18] Oszkár Semeráth, András Szabolcs Nagy, and Dániel Varró. "A Graph Solver for the Automated Generation of Consistent Domain-Specific Models". In: *ICSE '18*. ACM, 2018, pp. 969–980.

[Soe+10] Mathias Soeken et al. "Verifying UML/OCL models using Boolean satisfiability". In: *DATE*. IEEE, 2010, pp. 1341–1344.

[Son+11] Hui Song et al. "Instant and Incremental QVT Transformation for Runtime Models". In: *MODELS 2011*. Springer, 2011, pp. 273–288. DOI: 10.1007/978-3-642-24485-8_20.

[SPV18] Gagandeep Singh, Markus Püschel, and Martin Vechev. "A Practical Construction for Decomposing Numerical Abstract Domains". In: *Proc. ACM Program. Lang.* 2.POPL (2018). Article no. 2. DOI: 10.1145/3158143.

[Spy05] Peter Spyns. "Adapting the Object Role Modelling Method for Ontology Modelling". In: *ISMIS*. Vol. 3488. LNCS. Springer, 2005, pp. 276–284. DOI: 10.1007/11425274_29.

[SRA92] A. K. Somani, J. A. Ritcey, and S. H. L. Au. "Computationally-efficient phased-mission reliability analysis for systems with variable configurations". In: *IEEE Tran. Reliability* 41.4 (1992), pp. 504–511. DOI: 10.1109/24.249576.

[SSB17] G. Soltana, M. Sabetzadeh, and L. C. Briand. "Synthetic data generation for statistical testing". In: *ASE*. 2017, pp. 872–882. DOI: 10.1109/ASE.2017.8115698.

[SSB20] Ghanem Soltana, Mehrdad Sabetzadeh, and Lionel C. Briand. "Practical Constraint Solving for Generating System Test Data". In: *ACM Trans. Softw. Eng. Methodol.* 29.2 (2020), 11:1–11:48. DOI: 10.1145/3381032.

[Ste+09] Dave Steinberg et al. *EMF: Eclipse Modeling Framework*. 2nd ed. Addison-Wesley Prof., 2009.

[Ste10] Perdita Stevens. "Bidirectional model transformations in QVT: semantic issues and open questions". In: *Soft. Syst. Model.* 9.7 (2010). DOI: 10.1007/s10270-008-0109-9.

[Ste14] Perdita Stevens. "Bidirectionally tolerating inconsistency: partial transformations". In: *FASE*. Springer, 2014, pp. 32–46. DOI: 10.1007/978-3-642-54804-8_3.

[Sur+10] Neeraj Suri et al. "A software integration approach for designing and assessing dependable embedded systems". In: *J. Syst. Softw.* 83 (2010), pp. 1780–1800. DOI: 10.1016/j.jss.2010.04.063.

[SV17] Oszkár Semeráth and Dániel Varró. "Graph Constraint Evaluation over Partial Models by Constraint Rewriting". In: *ICMT*. 2017, pp. 138–154. DOI: 10.1007/978-3-319-61473-1_10.

[SVV16] Oszkár Semeráth, András Vörös, and Dániel Varró. "Iterative and Incremental Model Generation by Logic Solvers". In: *FASE*. 2016, pp. 87–103. DOI: 10.1007/978-3-662-49665-7_6.

[SysML2] The Object Management Group. 2022. URL: https://github.com/Systems-Modeling/SysML-v2-Release/tree/2022-11.

[SysMod] *System Modeling course*. Budapest Univ. of Technology and Economics. URL: https://portal.vik.bme.hu/kepzes/targyak/VIMIAA00/en/.

[SZ13] Michael Szvetits and Uwe Zdun. "Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime". In: *Software & Systems Modeling* 15.1 (2013), pp. 31–69. ISSN: 1619-1366. DOI: 10.1007/s10270-013-0394-9.

[Szá+14] Gábor Szárnyas et al. "IncQuery-D: A Distributed Incremental Model Query Framework in the Cloud". In: *MODELS*. Vol. 8767. LNCS. Springer, 2014, pp. 653–669. DOI: 10.1007/978-3-319-11653-2_40.

[Szá+17] Gábor Szárnyas et al. "The Train Benchmark: cross-technology performance evaluation of continuous model queries". In: *Softw. Syst. Model.* (2017). DOI: 10.1007/s10270-016-0571-8.

[TGS06] Matthias Tichy, Holger Giese, and Andreas Seibel. "Story diagrams in real-time software". In: *Proc. of the 4th International Fujaba Days*. 2006, pp. 15–22.

[TH19] Joze Tavcar and Imre Horvath. "A review of the principles of designing smart cyber-physical systems for run-time adaptation: Learned lessons and open issues". In: *IEEE Trans. Syst. Man Cybern. Syst.* 49.1 (2019), pp. 145–158. DOI: 10.1109/TSMC.2018.2814539.

[TJ07] Emina Torlak and Daniel Jackson. "Kodkod: A relational model finder". In: *TACAS*. Springer, 2007, pp. 632–647.

[TM15] Zoltán Theisz and Gergely Mezei. "An Algebraic Instantiation Technique Illustrated by Multilevel Design Patterns". In: *MULTI@MODELS*. Vol. 1505. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 53–62. URL: http://ceur-ws.org/Vol-1505/p6.pdf.

[TR96] Juha Taina and Kimmo Raatikainen. "Rodain: A real-time object-oriented database system for telecommunications". In: *International Conference on Information and Knowledge Management*. 1996, pp. 10–14. DOI: 10.1145/352302.352306.

[Ujh+15] Zoltán Ujhelyi et al. "EMF-IncQuery: An integrated development environment for live model queries". In: *Sci. Comput. Program.* 98.1 (2015), pp. 80–99. DOI: 10.1016/j.scico.2014.01.004.

[UML] The Object Management Group. *Object Constraint Language, v2.5.1*. 2017. URL: https://www.omg.org/spec/UML/2.5.1.

[UTM18] Dániel Urbán, Zoltán Theisz, and Gergely Mezei. "Self-describing Operations for Multi-level Meta-modeling". In: *MODELSWARD*. SciTePress, 2018, pp. 519–527. DOI: 10.5220/0006656105190527.

[Var+06] Dániel Varró et al. "Termination Analysis of Model Transformations by Petri Nets". In: *ICGT*. Vol. 4178. LNCS. Springer, 2006, pp. 260–274. DOI: 10.1007/11841883_19.

[Var+15] Gergely Varró et al. "An algorithm for generating model-sensitive search plans for pattern matching on EMF models". In: *Softw. Syst. Model.* (2015), pp. 597–621. DOI: 10.1007/s10270-013-0372-2.

[Var+16] Dániel Varró et al. "Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework". In: *Softw. Syst. Model.* 15.3 (2016), pp. 609–629. DOI: 10.1007/s10270-016-0530-4.

[Var+18] Dániel Varró et al. "Towards the Automated Generation of Consistent, Diverse, Scalable and Realistic Graph Models". In: *Graph Transformation, Specifications, and Nets – In Memory of Hartmut Ehrig*. LNCS 10800. Springer, 2018, pp. 285–312. DOI: 10.1007/978-3-319-75396-6_16.

[VB07] Dániel Varró and András Balogh. "The Model Transformation Language of the VIATRA2 Framework". In: *Sci. Comput. Program.* 68.3 (2007), pp. 214–234. DOI: 10.1016/j.scico.2007.05.004.

[Vog+15] Birgit Vogel-Heuser et al. “Evolution of software in automated production systems: Challenges and research directions”. In: *J. Syst. Softw.* 110 (2015), pp. 54–84. DOI: 10.1016/j.jss.2015.08.026.

[Vör+18b] András Vörös et al. “MoDeS3: Model-Based Demonstrator for Smart and Safe Cyber-Physical Systems”. In: *NASA Formal Methods*. 2018, pp. 460–467.

[VP03] Dániel Varró and András Pataricza. “VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML (The Mathematics of Metamodeling is Metamodeling Mathematics)”. In: *Softw. Syst. Model.* 2.3 (2003), pp. 187–210. DOI: 10.1007/s10270-003-0028-8.

[WA21] Nils Weidmann and Anthony Anjorin. “Schema Compliant Consistency Management via Triple Graph Grammars and Integer Linear Programming”. In: *Formal Aspects Comput.* 33.6 (2021), pp. 1115–1145. DOI: 10.1007/s00165-021-00557-0.

[Wag+22] D.A. Wagner et al. “Ontological Metamodeling and Analysis Using openCAESAR”. In: *Handbook of Model-Based Systems Engineering*. Springer, 2022. DOI: 10.1007/978-3-030-27486-3_78-1.

[Wec+18] Markus Weckesser et al. “Mathematical Programming for Anomaly Analysis of Clafer Models”. In: *MODELS*. ACM, 2018, pp. 34–44. DOI: 10.1145/3239372.3239398.

[Wen+05] I. Wenzel et al. “Measurement-based worst-case execution time analysis”. In: *Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)*. 2005, pp. 7–10. DOI: 10.1109/SEUS.2005.12.

[Wil+08] Reinhard Wilhelm et al. “The worst-case execution-time problem-overview of methods and survey of tools”. In: *Transactions on Embedded Computing Systems* 7.3 (2008). DOI: 10.1145/1347375.1347389.

[Wil17] Edward D. Willink. “The Micromapping Model of Computation; The Foundation for Optimized Execution of Eclipse QVTc/QVTr/UMLX”. In: *ICMT 2017*. Springer, 2017, pp. 51–65. DOI: 10.1007/978-3-319-61473-1_4.

[WT07] Dazhi Wang and Kishor S. Trivedi. “Reliability Analysis of Phased-Mission System With Independent Component Repairs”. In: *IEEE Tran. Reliability* 56.3 (2007), pp. 540–551. DOI: 10.1109/TR.2007.903268.

[Wu16] Hao Wu. “An SMT-based Approach for Generating Coverage Oriented Metamodel Instances”. In: *Int. J. Inf. Syst. Model. Design* 7.3 (2016).

[Xia+11] Jianwen Xiang et al. “Automatic Synthesis of Static Fault Trees from System Models”. In: *SSIRI*. IEEE, 2011. DOI: 10.1109/SSIRI.2011.32.

[Xie+21] Cheng Xie et al. “Multilayer Internet-of-Things Middleware Based on Knowledge Graph”. In: *IEEE Internet of Things Journal* 8.4 (2021), pp. 2635–2648. DOI: 10.1109/JIOT.2020.3019707.

[Xin07] Liudong Xing. “Reliability Evaluation of Phased-Mission Systems With Imperfect Fault Coverage and Common-Cause Failures”. In: *IEEE Tran. Reliability* 56.1 (2007), pp. 58–68. DOI: 10.1109/TR.2006.890900.

[Yak] Yakindu Statechart Tools. *Yakindu*.

[YBP07] Fang Yu, Tevfik Bultan, and Erik Peterson. “Automated Size Analysis for OCL”. In: *ESEC / FSE*. ACM, 2007, pp. 331–340. DOI: 10.1145/1287624.1287671.

[ZDG09] Haitao Zhu, Matthew B. Dwyer, and Steve Goddard. “Predictable runtime monitoring”. In: *Euromicro Conference on Real-Time Systems* 2 (2009), pp. 173–183.

Appendix A

Proofs of propositions from Chapter 2

A.1 4-valued partial models

Proposition 2.9 Refinement of 4-valued partial models is transitive, i.e., if P, Q, R are regular partial models, $P \succcurlyeq_{ref_1} Q$, and $Q \succcurlyeq_{ref_2} R$, then $P \succcurlyeq_{ref_3} R$ for some refinement ref_3.

Proof. We will set $ref_3 = ref_2 \circ ref_1 = \{\langle p, r\rangle \mid \langle p, q\rangle \in ref_1, \langle q, r\rangle \in ref_2\}$ and show that it satisfies the properties of a refinement relation.

VR1: Let $p \in O_P$ and $\mathcal{I}_P(\varepsilon)(p) \preccurlyeq 1$. By VR1, there is some $\langle p, q\rangle \in ref_1$ and $\mathcal{I}_P(\varepsilon)(p) \succcurlyeq \mathcal{I}_Q(\varepsilon)(q)$. This means that $\mathcal{I}_Q(\varepsilon)(q) \preccurlyeq 1$. Therefore, by VR1, there is some $\langle q, r\rangle \in ref_2$ and $\mathcal{I}_Q(\varepsilon)(q) \succcurlyeq \mathcal{I}_R(\varepsilon)(r)$. Hence, we conclude that $\langle p, r\rangle \in ref_3$ and $\mathcal{I}_P(\varepsilon)(p) \succcurlyeq \mathcal{I}_Q(\varepsilon)(q) \succcurlyeq \mathcal{I}_R(\varepsilon)(r)$ as required.

VR2: Let $r \in O_R$ with $\mathcal{I}_R(\varepsilon)(r) \succcurlyeq 1$. By VR2, there is some $q \in O_Q$ such that $\langle q, r\rangle \in ref_2$ and $\mathcal{I}_Q(\varepsilon)(q) \succcurlyeq 1$. Thus, there is some $p \in O_P$ such that $\langle p, q\rangle \in ref_1$ and $\mathcal{I}_P(\varepsilon)(p) \succcurlyeq 1$. Hence, we conclude that $\langle p, r\rangle \in ref_3$ as required.

VR3: Let $\varsigma \in \Sigma \setminus \{\varepsilon\}$, $\langle p_1, r_1\rangle, \ldots, \langle p_{\alpha(\varsigma)}, r_{\alpha(\varsigma)}\rangle \in ref_3$. Then there are $q_1, \ldots, q_{\alpha(\sigma)} \in O_Q$ such that $\langle p_1, q_1\rangle, \ldots, \langle p_{\alpha(\varsigma)}, q_{\alpha(\varsigma)}\rangle \in ref_1$ and $\langle q_1, r_1\rangle, \ldots, \langle q_{\alpha(\varsigma)}, r_{\alpha(\varsigma)}\rangle \in ref_2$. By VR3,

$$\mathcal{I}_P(\varsigma)(p_1, \ldots, p_{\alpha(\varsigma)}) \succcurlyeq \mathcal{I}_Q(\varsigma)(q_1, \ldots, q_{\alpha(\varsigma)}) \succcurlyeq \mathcal{I}_R(\varsigma)(r_1, \ldots, r_{\alpha(\varsigma)})$$

due to the transitivity of the information ordering relation $\succcurlyeq$ of 4-valued logic values.

VR4: Let $\langle p, q\rangle \in ref_1$ and $\langle q, r\rangle \in ref_2$. Then we have $\langle p, r\rangle \in ref_3$ according to the definition of ref_3 and also $\mathcal{V}_P(p) \supseteq \mathcal{V}_Q(q) \supseteq \mathcal{V}_R(r)$ by VR4 as required. □

Lemma 2.11 The semantics of expressions obeys partial model refinement, i.e., if $P = \langle O_P, \mathcal{I}_P, \mathcal{V}_P\rangle$ and $Q = \langle O_Q, \mathcal{I}_Q, \mathcal{V}_Q\rangle$ are 4-valued partial models over the signature $\langle \Sigma, \alpha\rangle$, $P \succcurlyeq_{ref} Q$ for some refinement ref, φ is a logic formula (resp. μ is a numerical expression) with free variables $V = \{v_1, \ldots, v_k\}$, $Z : V \to O_P$ and $Y : V \to O_Q$ are variable bindings, and $\langle Z(v), Y(v)\rangle \in ref$ for all $v \in V$, then $[\![\varphi]\!]^P_Z \succcurlyeq [\![\varphi]\!]^Q_Y$ (resp. $(\!|\mu|\!)^P_Z \supseteq (\!|\mu|\!)^Q_Y$).

Proof. We will proceed by induction over formulas φ and expressions μ. In our induction hypothesis, we will assume that Lemma 2.11 holds for all sub-formulas (sub-expressions) of φ and μ, as well as all formulas with strictly fewer transitive closure operations ς^+ or strictly fewer existential quantifiers $\forall$ than φ. Since all formulas contain finitely many sub-formulas, quantifiers, and transitive closure operators, this induces a well-founded partial order on formulas and expressions (i.e., all downward chains terminate in an atomic formula or a numerical literal

as the base case). Therefore, we may prove Lemma 2.11 by proving the induction hypothesis via case analysis:

$\underline{\varsigma(v_1, \ldots, v_{\alpha(\varsigma)})}$, $\underline{v_1 \sim v_2}$: The base cases of atomic formulas are satisfied due to Condition VR3.

$\underline{\neg\varphi_1}$, $\underline{\varphi_1 \vee \varphi_2}$, $\underline{\varphi_1 \wedge \varphi_2}$: We appeal to the induction hypothesis on φ_1 and φ_2, as well as to the monotonicity of the $\neg^4$, $\vee^4$, $\wedge^4$ connectives of 4-valued logic w.r.t. the information ordering $\succcurlyeq$.

$\underline{\exists v\colon \varphi_1}$: If $[\![\exists v\colon \varphi_1]\!]^P_Z = ½$, then $[\![\exists v\colon \varphi_1]\!]^Q_Y \preccurlyeq [\![\exists v\colon \varphi_1]\!]^P_Z = ½$ trivially holds.

If $[\![\exists v\colon \varphi_1]\!]^P_Z = 1$, then there must be some $p \in O_P$ such that $\mathcal{I}_P(\varepsilon)(p) \wedge^4 [\![\varphi_1]\!]^P_{Z,v\mapsto p} \preccurlyeq 1$. This means that $\mathcal{I}_P(\varepsilon)(p) \preccurlyeq 1$ and $[\![\varphi_1]\!]^P_{Z,v\mapsto p} \preccurlyeq 1$. By VR1, there is some $\langle p, q\rangle \in \mathit{ref}$ such that $\mathcal{I}_Q(\varepsilon)(q) \preccurlyeq \mathcal{I}_P(\varepsilon)(p) \preccurlyeq 1$. By the induction hypothesis, $[\![\varphi_1]\!]^P_{Z,v\mapsto p} \succcurlyeq [\![\varphi_1]\!]^Q_{Y,v\mapsto q}$. Therefore, $\mathcal{I}_Q(\varepsilon)(q) \wedge^4 [\![\varphi_1]\!]^Q_{Y,v\mapsto q} \preccurlyeq 1$, which means that $[\![\exists v\colon \varphi_1]\!]^Q_Y \preccurlyeq 1$ as required.

If $[\![\exists v\colon \varphi_1]\!]^P_Z = 0$, then $\mathcal{I}_P(\varepsilon)(p') \wedge^4 [\![\varphi_1]\!]^P_{Z,v\mapsto p'} = 0$ for all $p' \in O_P$. Assume for the sake of contradiction that $[\![\exists v\colon \varphi_1]\!]^Q_Y = ½$ or 1. This means that there is some $q \in O_Q$ such that $\mathcal{I}_Q(\varepsilon)(q) \wedge^4 [\![\varphi_1]\!]^Q_{Y,v\mapsto q} \succcurlyeq 1$. Therefore, $\mathcal{I}_Q(\varepsilon)(q) \succcurlyeq 1$ and $[\![\varphi_1]\!]^Q_{Y,v\mapsto q} \succcurlyeq 1$. By VR2, there is some $\langle p, q\rangle \in \mathit{ref}$ such that $\mathcal{I}_P(\varepsilon)(p) \succcurlyeq 1$. Since $\mathcal{I}_P(\varepsilon)(p) \wedge^4 [\![\varphi_1]\!]^P_{Z,v\mapsto p} = 0$, we must have $\mathcal{I}_P(\varepsilon)(p) = ½$ and $[\![\varphi_1]\!]^P_{Z,v\mapsto p} = \lightning$. But, by the induction hypothesis, $\lightning = [\![\varphi_1]\!]^P_{Z,v\mapsto p} \succcurlyeq [\![\varphi_1]\!]^Q_{Y,v\mapsto y} \succcurlyeq 1$, which is a contradiction. Our assumption cannot hold. Thus, $0 = [\![\exists v\colon \varphi_1]\!]^P_Z \succcurlyeq [\![\exists v\colon \varphi_1]\!]^Q_Y$ as required.

If $[\![\exists v\colon \varphi_1]\!]^P_Z = \lightning$, then $\mathcal{I}_P(\varepsilon)(p') \wedge^4 [\![\varphi_1]\!]^P_{Z,v\mapsto p'} \preccurlyeq 0$ for all $p' \in O_P$ and there is some $p \in O_P$ such that $\mathcal{I}_P(\varepsilon)(p) \wedge^4 [\![\varphi_1]\!]^P_{Z,v\mapsto p} = \lightning$.

First we will show that $\mathcal{I}_Q(\varepsilon)(q') \wedge^4 [\![\varphi_1]\!]^Q_{Y,v\mapsto q'} \preccurlyeq 0$ for all $q' \in O_Q$. Assume for the sake of contradiction that there is some $q'' \in O_Q$ such that $\mathcal{I}_Q(\varepsilon)(q'') \wedge^4 [\![\varphi_1]\!]^Q_{Y,v\mapsto q''} \succcurlyeq 1$. Therefore, $\mathcal{I}_Q(\varepsilon)(q'') \succcurlyeq 1$ and $[\![\varphi_1]\!]^Q_{Y,v\mapsto q''} \succcurlyeq 1$. By VR2, there is some $\langle p'', q''\rangle \in \mathit{ref}$ such that $\mathcal{I}_P(\varepsilon)(p'') \succcurlyeq 1$. Since $\mathcal{I}_P(\varepsilon)(p'') \wedge^4 [\![\varphi_1]\!]^P_{Y,v\mapsto p''} \preccurlyeq 0$, we must also have $[\![\varphi_1]\!]^P_{Y,v\mapsto p''} \preccurlyeq 0$. But, by the induction hypothesis, $0 \succcurlyeq [\![\varphi_1]\!]^P_{Y,v\mapsto p''} \succcurlyeq [\![\varphi_1]\!]^Q_{Y,v\mapsto q''} \succcurlyeq 1$, which is a contradiction as required.

Now consider the individual $p \in O_P$ with $\mathcal{I}_P(\varepsilon)(p) \wedge^4 [\![\varphi_1]\!]^P_{Z,v\mapsto p} = \lightning$. We have $\mathcal{I}_P(\varepsilon)(p) \preccurlyeq 1$, $[\![\varphi_1]\!]^P_{Z,v\mapsto p} \preccurlyeq 1$, and either have $\mathcal{I}_P(\varepsilon)(p) = \lightning$ or $[\![\varphi_1]\!]^P_{Z,v\mapsto p} = \lightning$. If $\mathcal{I}_P(\varepsilon)(p) = \lightning$, then, by VR1, there exists some $\langle p, q\rangle \in \mathit{ref}$ such that $\lightning = \mathcal{I}_P(\varepsilon)(p) \succcurlyeq \mathcal{I}_Q(\varepsilon)(q) = \lightning$. By the induction hypothesis, $1 \succcurlyeq [\![\varphi_1]\!]^P_{Z,v\mapsto p} \succcurlyeq [\![\varphi_1]\!]^Q_{Y,v\mapsto q}$. Conversely, if $[\![\varphi_1]\!]^P_{Z,v\mapsto p} = \lightning$, then $\lightning = [\![\varphi_1]\!]^P_{Z,v\mapsto p} \succcurlyeq [\![\varphi_1]\!]^Q_{Y,v\mapsto q} = \lightning$ by the induction hypothesis and $1 \succcurlyeq \mathcal{I}_P(\varepsilon)(p) \succcurlyeq \mathcal{I}_Q(\varepsilon)(q)$. In both cases, we have $\mathcal{I}_Q(\varepsilon)(q) \wedge^4 [\![\varphi_1]\!]^Q_{Y,v\mapsto q} = \lightning$.

Putting the two statements above together, we have shown that $[\![\exists v\colon \varphi_1]\!]^Q_Y = \lightning$ as required.

$\underline{\forall v\colon \varphi_1}$: The formula $\exists v\colon \neg\varphi_1$ has one fewer existential quantifiers than $\forall v\colon \varphi_1$. Therefore, we may appeal to the induction hypothesis and the monotonicity of the $\neg^4$ connective w.r.t. the information ordering $\succcurlyeq$.

$\underline{\varsigma^+(v_1, v_2)}$: The formula

$$\varsigma(v_1, v_2) \vee \bigvee_{i=1}^{|O_P|} \exists u_1\colon \cdots \exists u_i\colon \varsigma(v_1, u_1) \wedge \bigwedge_{j=1}^{i-1} \varsigma(u_j, u_{j+1}) \wedge \varsigma(u_i, v_2)$$

has one fewer transitive closure operators than $\varsigma^+(v_1, v_2)$. Therefore, we may appeal to the induction hypothesis.

$\underline{\mu_1 \in iv}$: If $[\![\mu_1 \in iv]\!]^P_Z = ½$, then the lemma is trivially satisfied.

If $[\![\mu_1 \in iv]\!]^P_Z = 1$, then $(\!|\mu_1|\!)^P_Z \subseteq iv$. By the induction hypothesis, $iv \supseteq (\!|\mu_1|\!)^P_Z \supseteq (\!|\mu_1|\!)^Q_Y$. We either have $(\!|\mu_1|\!)^Q_Y = \emptyset$ and $1 = [\![\mu_1 \in iv]\!]^P_Z \succcurlyeq [\![\mu_1 \in iv]\!]^Q_Y = \lightning$, or $1 = [\![\mu_1 \in iv]\!]^P_Z \succcurlyeq [\![\mu_1 \in iv]\!]^Q_Y = 1$. In both cases, the lemma is satisfied.

If $[\![\mu_1 \in iv]\!]^P_Z = 0$, then $(\!|\mu_1|\!)^P_Z \cap iv = \emptyset$. By the induction hypothesis, $\emptyset = (\!|\mu_1|\!)^P_Z \cap iv \supseteq (\!|\mu_1|\!)^Q_Y \cap iv$. We either have $(\!|\mu_1|\!)^Q_Y = \emptyset$ and $0 = [\![\mu_1 \in iv]\!]^P_Z \succcurlyeq [\![\mu_1 \in iv]\!]^Q_Y = \lightning$, or $0 = [\![\mu_1 \in iv]\!]^P_Z \succcurlyeq [\![\mu_1 \in iv]\!]^Q_Y = 0$. In both cases, the lemma is satisfied.

If $[\![\mu_1 \in iv]\!]^P_Z = \lightning$, then $(\!|\mu_1|\!)^P_Z = \emptyset$. By the induction hypothesis, we have $\emptyset = (\!|\mu_1|\!)^P_Z \supseteq (\!|\mu_1|\!)^Q_Y = \emptyset$. Therefore, as $[\![\mu_1 \in iv]\!]^Q_Y = \lightning$ as required.

literal: Observe that $(\!|literal|\!)^P_Z = (\!|literal|\!)^Q_Y = \{literal\}$, because the semantics of a numerical literal expression is independent of the partial model and the variable binding. Therefore, the lemma is trivially satisfied in this base case.

v: By VR4, we have $\mathcal{V}_P(Z(v)) \supseteq \mathcal{V}_Q(Y(v))$. Therefore, $(\!|v|\!)^P_Z \supseteq (\!|v|\!)^Q_Y$ as required.

$\mu_1 \langle op \rangle \mu_2$: We appeal to the induction hypothesis and the monotonicity of the interval algebra operators $+^\sharp$, $-^\sharp$, $\cdot^\sharp$, $/^\sharp$, and $\uparrow^\sharp$. □

A.2 Scoped partial models

Proposition 2.26 Refinement of regular scoped partial models is transitive, i.e., if P, Q, R are regular partial models, $P \succcurlyeq_{ref_1} Q$, and $Q \succcurlyeq_{ref_2} R$, then $P \succcurlyeq_{ref_3} R$ for some refinement ref_3.

Proof. We will set $ref_3 = ref_2 \circ ref_1 = \{\langle p, r\rangle \mid \langle p, q\rangle \in ref_1, \langle q, r\rangle \in ref_2\}$ and show that it satisfies the properties of a refinement relation. We may prove the conditions SR1–SR3 analogously to the conditions VR1–VR3 in Proposition 2.9 by noting that the information ordering of 3-valued logic values coincides with that of 4-valued logic values. Thus, only SR4 remains to be handled here. In the proof for the transitivity of SR4, we will make use of the regularity of the partial models involved, but the transitivity of SR1–SR3 holds even for scoped partial models that are not regular.

SR4: We will show that $\mathcal{S}_P[ref_1][ref_2] = \mathcal{S}_P[ref_3]$. Consider a variable $\mathfrak{x} = \hat{\gamma}(p_1, \ldots, p_{\alpha(\gamma)})$ in $\mathcal{X}_P$. In the $[ref_1]$ substitution of $\mathcal{S}_P$, $\mathfrak{x}$ will get replaced by the sum $\sum_{\mathfrak{y} \in \mathcal{Y}} \mathfrak{y}$ of the variables in

$$\mathcal{Y} = \{\hat{\gamma}(q_1, \ldots, q_{\alpha(\gamma)}) \mid \langle p_1, q_1, \rangle, \ldots, \langle p_{\alpha(\gamma)}, q_{\alpha(\gamma)}\rangle \in ref_1\}.$$

Assume for the sake of contradiction that $\mathfrak{y} = \hat{\gamma}(q_1, \ldots, q_{\alpha(\gamma)})$ and $\mathfrak{y}' = \hat{\gamma}(q'_1, \ldots, q'_{\alpha(\gamma)})$ are two distinct variables in $\mathcal{Y}$ such that the sets

$$\mathcal{Z}_\mathfrak{y} = \{\hat{\gamma}(r_1, \ldots, r_{\alpha(\gamma)}) \mid \langle q_1, r_1, \rangle, \ldots, \langle q_{\alpha(\gamma)}, r_{\alpha(\gamma)}\rangle \in ref_2\},$$
$$\mathcal{Z}_{\mathfrak{y}'} = \{\hat{\gamma}(r_1, \ldots, r_{\alpha(\gamma)}) \mid \langle q'_1, r_1, \rangle, \ldots, \langle q'_{\alpha(\gamma)}, r_{\alpha(\gamma)}\rangle \in ref_2\}$$

have at least one element $\mathfrak{z} = \hat{\gamma}(r_1, \ldots, r_{\alpha(\gamma)})$ in common. Since $\mathfrak{y} \neq \mathfrak{y}'$ there must be some $1 \leq i \leq \alpha(\gamma)$ such that $q_i \neq q'_i$. However, $\mathfrak{z} \in \mathcal{Z}_\mathfrak{y}$ and $\mathfrak{z} \in \mathcal{Z}_{\mathfrak{y}'}$ imply that both $\langle q_i, r_i\rangle, \langle q'_i, r_i\rangle \in ref_2$. Since R is regular, $\mathcal{I}_R(\sim)(r_i, r_i) = 1$. By SR3, we also have $\mathcal{I}_Q(\sim)(q_i, q'_i) \neq 0$. This would mean that Q is not regular, which is a contradiction. The sets $\mathcal{Z}_\mathfrak{y}$ and $\mathcal{Z}_{\mathfrak{y}'}$ must be disjoint for any distinct $\mathfrak{y}, \mathfrak{y}' \in \mathcal{Y}$. Therefore, the substitution $[ref_2]$ of $\mathcal{S}_P[ref_1]$ replaces each term $\mathfrak{y}$ of the sums $\sum_{\mathfrak{y} \in \mathcal{Y}} \mathfrak{y}$ obtained from $\mathfrak{x}$ by the first substitution $[ref_1]$ with non-overlapping sums $\sum_{\mathfrak{z} \in \mathcal{Z}_\mathfrak{y}} \mathfrak{z}$. Taken together, the substitutions $[ref_1][ref_2]$ have the effect of replacing $\mathfrak{x}$ with the sum $\sum_{\mathfrak{z} \in \mathcal{Z}} \mathfrak{z}$, where

$$\mathcal{Z} = \bigcup_{\mathfrak{y} \in \mathcal{Y}} \mathcal{Z}_\mathfrak{y} = \{\hat{\gamma}(r_1, \ldots, r_{\alpha(\gamma)}) \mid \langle p_1, r_1\rangle, \ldots, \langle p_{\alpha(\gamma)}, r_{\alpha(\gamma)}\rangle \in ref_2 \circ ref_1\},$$

which is precisely the substitution $[ref_3]$. By SR4, we have $\mathcal{S}_Q \vDash \mathcal{S}_P[ref_1]$ and $\mathcal{S}_R \vDash \mathcal{S}_Q[ref_2]$ and thus $\mathcal{S}_R \vDash \mathcal{S}_Q[ref_2] \vDash \mathcal{S}_P[ref_1][ref_2] = \mathcal{S}_P[ref_3]$ required. □

Lemma 2.28 If $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ and $Q = \langle \mathcal{O}_Q, \mathcal{I}_Q, \mathcal{S}_Q \rangle$ are regular scoped partial models over the signature $\langle \Sigma, \Gamma, \alpha \rangle$, $P \succcurlyeq_{ref} Q$ for some refinement ref, $V = \{v_1, \ldots, v_m\}$ is a set of variables, $Z \colon V \to \mathcal{O}_P$ and $Y \colon V \to \mathcal{O}_Q$ are variable bindings, $\langle Z(v), Y(v) \rangle \in ref$ for all $v \in V$, and $\gamma, \delta \in \Gamma$, then

a. if $\mathcal{S}_P \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^P_Z \leq U$, then $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^Q_Y \leq U$.
b. if $\mathcal{S}_P \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^P_Z \geq L$, and $\langle \bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)} \rangle$ is Z-focused, then $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^Q_Y \geq L$; and
c. if $\mathcal{S}_P \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^P_Z \leq \#[\delta(\bar{v}'_1, \ldots, \bar{v}'_{\alpha(\delta)})]^P_Z$, and $\langle \bar{v}'_1, \ldots, \bar{v}'_{\alpha(\delta)} \rangle$ is Z-focused, then $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^Q_Y \leq \#[\delta(\bar{v}'_1, \ldots, \bar{v}'_{\alpha(\delta)})]^Q_Y$.

Proof. Case a: Assume for the sake of contradiction that $\mathcal{S}_Q \nvDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^Q_Y \leq U$, i.e., there is some binding $k \colon \mathcal{X}_Q \to \mathbb{N}$ such that $\sum_{\langle q_1, \ldots, q_{\alpha(\gamma)} \rangle \in resolve^Q_Y(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})} k(\hat{\gamma}(q_1, \ldots, q_{\alpha(\gamma)})) \geq U + 1$. Consider the binding $k' \colon \mathcal{X}_P \to \mathbb{N}$, where, for all $\delta \in \Gamma$ and $q'_1, \ldots, q'_{\alpha(\delta)} \in \mathcal{O}_Q$,

$$k'(\hat{\delta}(q'_1, \ldots, q'_{\alpha(\delta)})) = \sum_{\langle p'_1, q'_1 \rangle, \ldots, \langle p'_{\alpha(\delta)}, q'_{\alpha(\delta)} \rangle \in ref} k(\hat{\delta}(p'_1, \ldots, p'_{\alpha(\delta)})).$$

By SR4, we have $k \vDash \mathcal{S}_P[ref]$, which implies $k' \vDash \mathcal{S}_P$. We will show that

$$\begin{aligned} U + 1 &\leq \sum_{\langle q_1, \ldots, q_{\alpha(\gamma)} \rangle \in resolve^Q_Y(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})} k(\hat{\gamma}(q_1, \ldots, q_{\alpha(\gamma)})) && \text{(A.1)} \\ &\leq \sum_{\langle p_1, \ldots, p_{\alpha(\gamma)} \rangle \in resolve^P_Z(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})} \; \sum_{\langle p_1, q_1 \rangle, \ldots, \langle p_{\alpha(\gamma)}, q_{\alpha(\gamma)} \rangle \in ref} k(\hat{\gamma}(q_1, \ldots, q_{\alpha(\gamma)})) && \text{(A.2)} \\ &= \sum_{\langle p_1, \ldots, p_{\alpha(\gamma)} \rangle \in resolve^P_Z(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})} k'(\hat{\gamma}(p_1, \ldots, p_{\alpha(\gamma)})) \end{aligned}$$

by showing that any term appearing in the summation (A.1) also appears in (A.2), while the rest of the terms in (A.2) are nonnegative.

Consider the case when some term $k(\hat{\gamma}(q_1, \ldots, q_{\alpha(\gamma)}))$ of (A.1) does not appear in (A.2). This means that there is some index $1 \leq i \leq \alpha(\gamma)$ such that there is no $\langle p_1, \ldots, p_{\alpha(\gamma)} \rangle \in resolve^P_Z(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})$ with $\langle p_1, q_i \rangle \in ref$. If $\bar{v}_i \in V$, this is impossible, because then we have $p_i = Z(\bar{v}_i)$, $q_i = Y(\bar{v}_i)$, and $\langle Z(\bar{v}_i), Y(\bar{v}_i) \rangle \in ref$ by assumption. If $\bar{v}_i = *$, this is also impossible: Since Q is regular, we have $\mathcal{I}_Q(\varepsilon)(q_i, q_i) \neq 0$. Thus, there must exist some $p_i \in \mathcal{O}_P$ with $\langle p_i, q_i \rangle \in ref$ due to SR2. As a result, all terms of (A.1) must also appear in (A.2).

Therefore, $k' \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^P_Z \geq U + 1$. But we also have $k' \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^P_Z \leq U$, which is a contradiction. We must have had $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^Q_Y \leq U$ originally as required by the statement of the lemma.

Case b: Assume for the sake of contradiction that $\mathcal{S}_Q \nvDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^Q_Y \geq L$, i.e., there is some binding $k \colon \mathcal{X}_Q \to \mathbb{N}$ such that $\sum_{\langle q_1, \ldots, q_{\alpha(\gamma)} \rangle \in resolve^Q_Y(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})} k(\hat{\gamma}(q_1, \ldots, q_{\alpha(\gamma)})) \leq L + 1$. Like in case a, consider the binding $k' \colon \mathcal{X}_P \to \mathbb{N}$, where, for all $\delta \in \Gamma$ and $q'_1, \ldots, q'_{\alpha(\delta)} \in \mathcal{O}_Q$,

$$k'(\hat{\delta}(q'_1, \ldots, q'_{\alpha(\delta)})) = \sum_{\langle p'_1, q'_1 \rangle, \ldots, \langle p'_{\alpha(\delta)}, q'_{\alpha(\delta)} \rangle \in ref} k(\hat{\delta}(p'_1, \ldots, p'_{\alpha(\delta)})),$$

which satisfies $k' \vDash \mathcal{S}_P$ by SR4. We may assume w.l.o.g. that $\bar{v}_1 = v_1, \ldots, \bar{v}_m = v_m, \bar{v}_{m+1} = *, \ldots, \bar{v}_m = *$, i.e., the first m variables in the mask $\langle \bar{v}_1, \ldots, \bar{v}_{\alpha(\varsigma)} \rangle$ are bound while the rest are

wildcards. We will show that

$$
\begin{aligned}
L+1 \geq & \sum_{\langle q_1,\ldots,q_{\alpha(\gamma)}\rangle \in resolve_Y^Q(v_1,\ldots,v_m,*,\ldots,*)} k(\hat{\delta}(q_1,\ldots,q_{\alpha(\gamma)})) \\
= & \sum_{q_{m+1},\ldots,q_{\alpha(\gamma)} \in O_Q} k(\hat{\delta}(Y(v_1),\ldots,Y(v_m),q_{m+1},\ldots,q_{\alpha(\gamma)})) \qquad \text{(A.3)} \\
= & \sum_{p_{m+1},\ldots,p_{\alpha(\gamma)} \in O_P} \sum_{\langle Z(v_1),q_1\rangle,\ldots,\langle Z(v_m),q_m\rangle,\langle p_{m+1},q_{m+1}\rangle,\ldots,\langle p_{\alpha(\gamma)},q_{\alpha(\gamma)}\rangle \in ref} k(\hat{\delta}(q_1,\ldots,q_{\alpha(\gamma)})) \qquad \text{(A.4)} \\
= & \sum_{p_{m+1},\ldots,p_{\alpha(\gamma)} \in O_P} k'(\hat{\delta}(Z(v_1),\ldots,Z(v_m),p_{m+1},\ldots,p_{\alpha(\gamma)})) \\
= & \sum_{\langle p_1,\ldots,p_{\alpha(\gamma)}\rangle \in resolve_Z^P(v_1,\ldots,v_m,*,\ldots,*)} k'(\hat{\delta}(p_1,\ldots,p_{\alpha(\gamma)}))
\end{aligned}
$$

by showing that each term in (A.3) appears exactly once in (A.4) and vice verse.

Firstly, consider the case when a term $k(\hat{\gamma}(q_1,\ldots,q_{\alpha(\gamma)}))$ appears in (A.4), but does not appear in (A.3). Then we have some $1 \leq m$ such that $q_i \neq Y(v_i)$. This means that we have $\langle Z(v_1), Y(v_i)\rangle, \langle Z(v_i), q_i\rangle \in ref$. Since our mask is Z-focused, $\mathcal{I}_P(\sim)(Z(v_i), Z(v_i)) = 1$. By SR3, $\mathcal{I}_Q(\sim)(Y(v_i), q_i) = 1$. But this would violate the regularity of Q, so all terms of (A.4) must also appear in (A.3).

We handle the situation when a term of (A.3) is missing from (A.4) analogously to case a.

Consider the case when some term $k(\hat{\gamma}(Y(v_1),\ldots,Y(v_m),q_{m+1},\ldots,q_{\alpha(\gamma)}))$ in (A.3) appears twice in (A.4). This would mean that there is some $1 \leq j \leq \alpha(\gamma)$ and two distinct $p_j, p'_j \in O_P$ such that $\langle p_j, q_j\rangle, \langle p'_j, q_j\rangle \in ref$. Since Q is regular, $\mathcal{I}_Q(\sim)(q_j, q_j) \neq 0$. By SR3, $\mathcal{I}_Q(\sim)(p_j, p'_j) = 0$. But this would violate the regularity of P due to $p_j \neq p'_j$, so no term of (A.3) can appear more than once in (A.4).

Therefore, $k' \vDash \#[\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})]_Z^P \leq L-1$. But we also have $k' \vDash \#[\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})]_Z^P \geq L$, which is a contradiction. We must have had $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})]_Y^Q \geq L$ originally as required by the statement of the lemma.

Case c: Assume that $\mathcal{S}_Q \nvDash \#[\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})]_Y^Q \leq \#[\delta(\bar{v}'_1,\ldots,\bar{v}'_{\alpha(\gamma)})]_Y^Q$, i.e., there is some binding $k\colon \mathcal{X}_Q \to \mathbb{N}$ such that

$$
\sum_{\langle q_1,\ldots,q_{\alpha(\gamma)}\rangle \in resolve_Y^Q(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})} k(\hat{\gamma}(q_1,\ldots,q_{\alpha(\gamma)})) \geq \sum_{\langle q'_1,\ldots,q'_{\alpha(\delta)}\rangle \in resolve_Y^Q(\bar{v}'_1,\ldots,\bar{v}'_{\alpha(\delta)})} k(\hat{\gamma}(q'_1,\ldots,q'_{\alpha(\delta)})) + 1.
$$

Let us construct the binding $k'\colon \mathcal{X}_P \to \mathbb{N}$ as in cases a and b. We determine that

$$
\sum_{\langle p_1,\ldots,p_{\alpha(\gamma)}\rangle \in resolve_Z^P(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})} k'(\hat{\gamma}(p_1,\ldots,p_{\alpha(\gamma)})) \geq \sum_{\langle p'_1,\ldots,p'_{\alpha(\delta)}\rangle \in resolve_Z^P(\bar{v}'_1,\ldots,\bar{v}'_{\alpha(\delta)})} k'(\hat{\gamma}(p'_1,\ldots,p'_{\alpha(\delta)})) + 1
$$

by applying the reasoning from case a to the left side of the inequality and the reasoning from case b to the right side of the inequality. This is a contradiction. Therefore, we must have had $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1,\ldots,\bar{v}_{\alpha(\gamma)})]_Y^Q \leq \#[\delta(\bar{v}'_1,\ldots,\bar{v}'_{\alpha(\delta)})]_Y^Q$ instead. □

Lemma 2.30 The semantics of expressions obeys partial model refinement, i.e., if $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P\rangle$ and $Q = \langle O_Q, \mathcal{I}_Q, \mathcal{S}_Q\rangle$ are regular scoped partial models over the scoped signature $\langle \Sigma, \Gamma, \alpha\rangle$, $P \succcurlyeq_{ref} Q$ for some refinement ref, φ is a logic formula with free variables $V = \{v_1,\ldots,v_k\}$, $Z\colon V \to O_P$ and $Y\colon V \to O_Q$ are variable bindings, and $\langle Z(v), Y(v)\rangle \in ref$ for all $v \in V$, then $[\![\varphi]\!]_Z^P \succcurlyeq [\![\varphi]\!]_Y^Q$.

Proof. We proceed by case analysis and structural induction over the sub-formulas of φ as in the proof of Lemma 2.7.

$\underline{\varsigma(v_1, \ldots, v_{\alpha(\varsigma)})}$, $\underline{v_1 \sim v_2}$, $\underline{\neg\varphi_1}$, $\underline{\varphi_1 \vee \varphi_2}$, $\underline{\varphi_1 \wedge \varphi_2}$, $\underline{\exists v\colon \varphi_1}$, $\underline{\forall v\colon \varphi_1}$, $\underline{\varsigma^+(v_1, v_2)}$: These cases can be handled in the same way as in Lemma 2.7, but we can disregard the situations where $[\![\varphi]\!]^P_Z = \lightning$.

$\underline{\mathsf{count}\, \gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)}) \leq U}$: The case $[\![\mathsf{count}\, \gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)}) \geq L]\!]^P_Z = \frac{1}{2}$ is trivial.

If $[\![\mathsf{count}\, \gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)}) \leq U]\!]^P_Z = 1$, then $\mathcal{S}_P \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^P_Z \leq U$. By Lemma 2.28a, we also have $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^Q_Y \leq U$. Therefore, $[\![\mathsf{count}\, \gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)}) \leq U]\!]^Q_Y = 1$.

If $[\![\mathsf{count}\, \gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)}) \leq U]\!]^P_Z = 0$, then $\mathcal{S}_P \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^P_Z \geq U{+}1$. By Lemma 2.28b, we also have $\mathcal{S}_Q \vDash \#[\gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)})]^Q_Y \geq U + 1$. Therefore, $[\![\mathsf{count}\, \gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)}) \leq U]\!]^Q_Y = 0$.

$\underline{\mathsf{count}\, \gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)}) \geq L}$: Analogous to the case of $\mathsf{count}\, \gamma(\bar{v}_1, \ldots, \bar{v}_{\alpha(\gamma)}) \leq U$, but we apply Lemma 2.28b in the 1 subcase and Lemma 2.28a in the 0 subcase. □

Appendix **B**

Proofs of propositions from Chapter 4

Theorem 4.1 (Forward refinement of predicates) Let φ be a logic expression without free variables and let P and Q be regular scoped partial models, where $P \succcurlyeq Q$.

- If $[\![\varphi]\!]^P = 1$, then $[\![\varphi]\!]^Q = 1$.
- If $[\![\varphi]\!]^P = 0$, then $[\![\varphi]\!]^Q = 0$.

Proof. By Lemma 2.30, we have $[\![\varphi]\!]^P \succcurlyeq [\![\varphi]\!]^Q$. Therefore, if $[\![\varphi]\!]^P = 1$, we must also have $1 \succcurlyeq [\![\varphi]\!]^Q = 1$, because in 3-valued logic, there is no $\lightning$ logic value. Likewise, if $[\![\varphi]\!]^P = 0$, we must also have $0 \succcurlyeq [\![\varphi]\!]^Q = 0$. □

Theorem 4.2 (Backward refinement of predicates) Let φ be a logic expression without free variables and let P and Q be regular scoped partial models, where $P \succcurlyeq Q$.

- If $[\![\varphi]\!]^Q = 1$, then $[\![\varphi]\!]^P \succcurlyeq 1$.
- If $[\![\varphi]\!]^Q = 0$, then $[\![\varphi]\!]^P \succcurlyeq 0$.

Proof. The statements follow immediately from $[\![\varphi]\!]^P \succcurlyeq [\![\varphi]\!]^Q$ by Lemma 2.30. □

Theorem 4.5 (Forward refinement of scopes) Let φ be a logic expression with free variables V, P and Q regular scoped partial models with $P \succcurlyeq_{ref} Q$, $Z \colon V \to \mathcal{O}_P$ and $Y \colon V \to \mathcal{O}_Q$ variable binding with $\langle Z(v), Y(v) \rangle \in ref$ for all $v \in V$, and $L, U \in \mathbb{Z}$. Then the following implications hold:

$$\mathcal{S}_P \vDash \#_v^{\frac{1}{2}}[\![\varphi]\!]_Z^P < L \implies \mathcal{S}_Q \vDash \#_v^{\frac{1}{2}}[\![\varphi]\!]_Y^Q < L, \qquad \text{(i)}$$
$$\mathcal{S}_P \vDash \#_v^{1}[\![\varphi]\!]_Z^P > U \implies \mathcal{S}_Q \vDash \#_v^{1}[\![\varphi]\!]_Y^Q > U, \qquad \text{(ii)}$$

i.e., (i) when objects that *may* satisfy φ violate a lower bound L in P, they also violate it in any refined partial model Q, and (ii) objects that *must* satisfy φ similarly carry forward the violation of the upper bound U.

Proof. (i) Let us consider the signature $\langle \Sigma, \{\varepsilon\}, \alpha \rangle$ over which the regular scoped partial models $P = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_P \rangle$ and $Q = \langle \mathcal{O}_Q, \mathcal{I}_Q, \mathcal{S}_Q \rangle$ are defined. Moreover, let $P' = \langle \mathcal{O}_P, \mathcal{I}_P, \mathcal{S}_{P'} \rangle$ and $Q' = \langle \mathcal{O}_Q, \mathcal{I}_Q, \mathcal{S}_{Q'} \rangle$ be defined over the signature $\langle \Sigma, \Gamma', \alpha' \rangle$, where $\Gamma' = \{\varepsilon, \mathsf{X}\}$, $\mathsf{X} \notin \Sigma$ is a new

numerically tracked symbol (i.e., $\mathsf{X} \in \Gamma'_{\mathsf{X}}$), $\alpha'(\mathsf{X}) = 2$ and $\alpha'(\varsigma) = \alpha(\varsigma)$ for all $\varsigma \in \Sigma$, and

$$\mathcal{S}_{P'} = \mathcal{S}_P \cup \{\hat{\mathsf{X}}(p) = 0 \mid [\![\varphi]\!]^P_{Z,v\mapsto p} = 0\} \cup \{\hat{\mathsf{X}}(p) \le \hat{p} \mid [\![\varphi]\!]^P_{Z,v\mapsto p} \neq 0\},$$
$$\mathcal{S}_{Q'} = \mathcal{S}_Q \cup \{\hat{\mathsf{X}}(q) = 0 \mid [\![\varphi]\!]^Q_{Y,v\mapsto q} = 0\} \cup \{\hat{\mathsf{X}}(q) = \hat{q} \mid [\![\varphi]\!]^Q_{Y,v\mapsto q} \neq 0\}.$$

We may see that $P' \succcurlyeq_{ref} Q'$. By $P \succcurlyeq_{ref} Q$, we already know that Conditions SR1–SR3 hold. Moreover, by Condition SR4, we already have $\mathcal{S}_Q \vDash \mathcal{S}_P[ref]$, so we only need to show that SR4 also holds for the newly introduced linear inequalities.

If $[\![\varphi]\!]^P_{Z,v\mapsto p} = 0$, then $\hat{\mathsf{X}}(p) = 0 \in \mathcal{S}_{P'}$ and $\sum_{\langle p,q\rangle\in ref} \hat{\mathsf{X}}(q) = 0 \in \mathcal{S}_{P'}[ref]$. By Lemma 2.30, we have $0 = [\![\varphi]\!]^P_{Z,v\mapsto p} \succcurlyeq [\![\varphi]\!]^P_{Z,v\mapsto p} = 0$ for each $\langle p, q\rangle \in ref$, which means that $\mathcal{S}_{Q'} \vDash \hat{\mathsf{X}}(q) = 0$. Therefore, $\mathcal{S}_{Q'} \vDash \sum_{\langle p,q\rangle\in ref} \hat{\mathsf{X}}(q) = \sum_{\langle p,q\rangle\in ref} 0 = 0$.

If $[\![\varphi]\!]^P_{Z,v\mapsto p} \neq 0$, then $\hat{\mathsf{X}}(p) \le \hat{p} \in \mathcal{S}_{P'}$ and $\sum_{\langle p,q\rangle\in ref} \hat{\mathsf{X}}(q) \le \sum_{\langle p,q\rangle\in ref} \hat{q} \in \mathcal{S}_{P'}[ref]$. We have $\mathcal{S}_{Q'} \vDash \hat{\mathsf{X}}(q) \le \hat{q}$ for each $q \in O_Q$, because either $\mathcal{S}_{Q'} \vDash \hat{\mathsf{X}}(q) = 0$ or $\mathcal{S}_{Q'} \vDash \hat{\mathsf{X}}(q) = \hat{q}$ hold depending on the value of $[\![\varphi]\!]^Q_{Y,v\mapsto q}$. Therefore, $\mathcal{S}_{Q'} \vDash \sum_{\langle p,q\rangle\in ref} \hat{\mathsf{X}}(q) \le \sum_{\langle p,q\rangle\in ref} \hat{q}$, which completes the proof that $\mathcal{S}_{Q'} \vDash \mathcal{S}_P[ref]$ and $P' \succcurlyeq_{ref} Q'$.

Every linear inequality variable binding $k\colon \mathcal{X}_P \to \mathbb{N}$ with $k \vDash \mathcal{S}_P$ extends to a binding $k'\colon \mathcal{X}_{P'} \to \mathbb{N}$ with $k' \vDash \mathcal{S}_P$ by setting

$$k'(\mathfrak{x}) = \begin{cases} k(\mathfrak{x}) & \text{if } \mathfrak{x} \in \mathcal{X}_P, \\ 0 & \text{if } \mathfrak{x} = \hat{\mathsf{X}}(p) \text{ and } [\![\varphi]\!]^P_{Z,v\mapsto p} = 0, \\ k(\hat{p}) & \text{if } \mathfrak{x} = \hat{\mathsf{X}}(p) \text{ and } [\![\varphi]\!]^P_{Z,v\mapsto p} \neq 0, \end{cases}$$

and we also have $\mathcal{S}_{P'} \vDash \mathcal{S}_P$. Therefore, $\mathcal{S}_P \vDash \#^{½}_v [\![\varphi]\!]^P_Z < L$ if and only if $\mathcal{S}_{P'} \vDash \#^{½}_v [\![\varphi]\!]^{P'}_Z < L$. After applying the same argument as above to $\mathcal{S}_Q$ and $\mathcal{S}_{Q'}$, we may see that it suffices to show that $\mathcal{S}_{P'} \vDash \#^{½}_v [\![\varphi]\!]^{P'}_Z < L \implies \mathcal{S}_{Q'} \vDash \#^{½}_v [\![\varphi]\!]^{Q'}_Y < L$.

We may see that $\mathcal{S}_{P'} \vDash \#[\mathsf{X}(*)]^{P'} \le \#^{½}_v [\![\varphi]\!]^{P'}_Z < L$ and $\mathcal{S}_{Q'} \vDash \#[\mathsf{X}(*)]^{Q'} = \#^{½}_v [\![\varphi]\!]^{Q'}_Y$. By Case (a) of Lemma 2.28, $\mathcal{S}_{P'} \vDash \#[\mathsf{X}(*)]^{P'} < L$ implies $\mathcal{S}_{Q'} \vDash \#[\mathsf{X}(*)]^{Q'} < L$. Therefore, we have $\mathcal{S}_{Q'} \vDash \#^{½}_v [\![\varphi]\!]^{Q'}_Y < L$ as required.

(ii) Similarly to (i), we extend P and Q to obtain P' and Q' by

$$\mathcal{S}_{P'} = \mathcal{S}_P \cup \{\hat{\mathsf{X}}(p) = \hat{p} \mid [\![\varphi]\!]^P_{Z,v\mapsto p} = 1\} \cup \{\hat{\mathsf{X}}(p) \le \hat{p} \mid [\![\varphi]\!]^P_{Z,v\mapsto p} \neq 1\},$$
$$\mathcal{S}_{Q'} = \mathcal{S}_Q \cup \{\hat{\mathsf{X}}(q) = \hat{q} \mid [\![\varphi]\!]^Q_{Y,v\mapsto q} = 1\} \cup \{\hat{\mathsf{X}}(q) = 0 \mid [\![\varphi]\!]^Q_{Y,v\mapsto q} \neq 1\}.$$

We have $P' \succcurlyeq Q'$ again, and it will suffice to show $\mathcal{S}_{P'} \vDash \#^1_v [\![\varphi]\!]^{P'}_Z > U \implies \mathcal{S}_{Q'} \vDash \#^1_v [\![\varphi]\!]^{Q'}_Y > U$.

We may see that $\mathcal{S}_{P'} \vDash \#[\mathsf{X}(*)]^{P'} \ge \#^1_v [\![\varphi]\!]^{P'}_Z > L$ and $\mathcal{S}_{Q'} \vDash \#[\mathsf{X}(*)]^{Q'} = \#^1_v [\![\varphi]\!]^{Q'}_Y$. By Case (b) of Lemma 2.28, $\mathcal{S}_{P'} \vDash \#[\mathsf{X}(*)]^{P'} > U$ implies $\mathcal{S}_{Q'} \vDash \#[\mathsf{X}(*)]^{Q'} > U$. Therefore, we have $\mathcal{S}_{Q'} \vDash \#^1_v [\![\varphi]\!]^{Q'}_Y > U$ as required. □

Lemma B.1 Let φ be a logic predicate and P a regular scoped partial model. Then $\mathcal{S}_P \vDash \#^1_v [\![\varphi]\!]^P_Z \le \#^{½}_v [\![\varphi]\!]^P_Z$.

Proof. Consider the sets

$$A = \{x \in O_Q \mid [\![\varphi]\!]^Q_{Z,v\mapsto x} \Longleftarrow ½\},$$
$$B = \{x \in O_Q \mid [\![\varphi]\!]^Q_{Z,v\mapsto x} = 1\},$$
$$C = \{x \in O_Q \mid [\![\varphi]\!]^Q_{Z,v\mapsto x} = ½\},$$

where $A = B \cup C$. Then $\mathcal{S}_P \vDash \sum\{\widehat{x} \mid x \in C\} \geq 0$ and

$$\mathcal{S}_P \vDash \#_v^1[\![\varphi]\!]_Z^P = \sum\{\widehat{x} \mid x \in B\} \leq \sum\{\widehat{x} \mid x \in B\} + \sum\{\widehat{x} \mid x \in C\} = \sum\{\widehat{x} \mid x \in A\} = \#_v^{½}[\![\varphi]\!]_Z^P.$$

□

Theorem 4.6 (Backward refinement of scopes) Let φ be a logic expression with free variables V, P and Q regular scoped partial models with $P \succcurlyeq_{ref} Q$, $Z : V \to \mathcal{O}_P$ and $Y : V \to \mathcal{O}_Q$ variable binding with $\langle Z(v), Y(v)\rangle \in ref$ for all $v \in V$, and $L, U \in \mathbb{Z}$. Then the following implications hold:

$$\mathcal{S}_Q \vDash \#_v^1[\![\varphi]\!]_Y^Q \geq L \implies \mathcal{S}_P \nvDash \#_v^{½}[\![\varphi]\!]_Z^P < L,$$
$$\mathcal{S}_Q \vDash \#_v^{½}[\![\varphi]\!]_Y^Q \leq U \implies \mathcal{S}_P \nvDash \#_v^1[\![\varphi]\!]_Z^P > U.$$

Proof. We prove the proposition

$$\mathcal{S}_Q \vDash \#_v^1[\![\varphi]\!]_Y^Q \geq L \implies \mathcal{S}_P \nvDash \#_v^{½}[\![\varphi]\!]_Z^P < L$$

by contradiction. Let us assume that $\mathcal{S}_P \vDash \#_v^{½}[\![\varphi]\!]_Z^P < L$. By Theorem 4.5, we also have $\mathcal{S}_Q \vDash \#_v^{½}[\![\varphi]\!]_Y^Q < L$. Hence, $\mathcal{S}_Q \vDash \#_v^1[\![\varphi]\!]_Y^Q \leq \#_v^{½}[\![\varphi]\!]_Y^Q < L$ by Lemma B.1. We simultaneously have $\mathcal{S}_Q \vDash L \leq \#_v^1[\![\varphi]\!]_Y^Q < L$, which is a contradiction. $\mathcal{S}_Q$ cannot be satisfiable.

The case of

$$\mathcal{S}_Q \vDash \#_v^{½}[\![\varphi]\!]_Y^Q \leq U \implies \mathcal{S}_P \nvDash \#_v^1[\![\varphi]\!]_Z^P > U$$

is analogous. After assuming $\mathcal{S}_P \vDash \#_v^1[\![\varphi]\!]_Z^P > U$, we get the contradiction

$$\mathcal{S}_Q \vDash U \geq \#_v^{½}[\![\varphi]\!]_Y^Q \geq \#_v^1[\![\varphi]\!]_Y^Q > U$$

by applying Theorem 4.5 and Lemma B.1. □

Appendix C

Proofs of propositions from Chapter 6

Proposition 6.5 Let τ be the execution time of the query q on the concrete model M,

$$CL = \max \sum_{\mathfrak{x}_i \in \mathcal{X}} c_i \cdot \mathfrak{x}_i \text{ subject to } \mathcal{S}_{\text{IPET}}, \quad DS_M = \max \sum_{\mathfrak{x}_i \in \mathcal{X}} c_i \cdot \mathfrak{x}_i \text{ subject to } \mathcal{S}_{\text{IPET}} \cup \mathcal{S}_{\text{flow}},$$

where CL is the classical IPET estimate obtained from q, and DS_M is the domain-specific estimate with flow facts derived from M using Algorithm 4. Then $\tau \leq DS_M \leq CL$.

Proof. $\tau \leq DS_M$ (safety): Consider any execution path π of q. Because CL is safe, there is a solution $k_\pi : \mathcal{X}_{\text{IPET}} \to \mathbb{Z}$ such that $k_\pi(\mathfrak{f}(e)) = \pi \# e$ for all edges $e \in E$ of the CFG of q and $k_\pi \vDash \mathcal{S}_{\text{IPET}}$.

Consider the linear equations in $\mathcal{S}_{\text{flow}}$. We have a single linear equation for each basic block $bb \in BB$. If bb is a loop header, then every execution of bb is represented by either a match of ψ_{bb} of ψ'_{bb}. Thus, $\sum_{e=\langle n_1,n_2\rangle \in E, tr(n_1)=bb} \pi \# e = \sum_{e=\langle n_1,n_2\rangle \in E, tr(n_1)=bb} k_\pi(\mathfrak{f}(e)) = M\#\psi_{bb} + M\#\psi'_{bb}$, which means the corresponding linear equation in $\mathcal{S}_{\text{flow}}$ holds. Otherwise, every execution of bb is represented by a match of ψ_{bb}. Thus, $\sum_{e=\langle n_1,n_2\rangle \in E, tr(n_1)=bb} \pi \# e = \sum_{e=\langle n_1,n_2\rangle \in E, tr(n_1)=bb} k_\pi(\mathfrak{f}(e)) = M\#\psi_{bb}$, which means the corresponding linear equation in $\mathcal{S}_{\text{flow}}$ holds.

Therefore, we have $k_\pi \vDash \mathcal{S}_{\text{IPET}} \cup \mathcal{S}_{\text{flow}}$. We also have $\tau \leq CL \leq DS_M$ using the safety of CL.

$DS_M \leq CL$ (tightness): Assume that $DS_M > CL$. Then there is some $k \vDash \mathcal{S}_{\text{IPET}} \cup \mathcal{S}_{\text{flow}}$ such that $g(k) = \sum_{\mathfrak{x}_i} k(\mathfrak{x}_i) > CL = g(k^*)$, where $k^* \vDash \mathcal{S}_{\text{IPET}}$ is the optimal solution of the classical IPET integer program with $g(k^*) \geq g(k')$ for all $k' \vDash \mathcal{S}_{\text{IPET}}$.

However, $k \vDash \mathcal{S}_{\text{IPET}}$ (because $\mathcal{S}_{\text{IPET}} \cup \mathcal{S}_{\text{flow}} \vDash \mathcal{S}_{\text{IPET}}$), which means k^* cannot be optimal. Thus the assumption cannot hold. □

In the following, we will use the notation $\lceil P \rceil$ to refer to the extended partial model output by Algorithm 5 for a scoped partial model P. Recall the notation $\lfloor P' \rfloor$ from Definition 6.7 that refers to the projection of an extended partial model to the original signature $\langle \Sigma, \Gamma, \alpha \rangle$, i.e., removes the interpretations of the symbols and linear inequalities added by Algorithm 5. In particular, we have $\lfloor \lceil P \rceil \rfloor = P$.

Lemma C.1 Let P', Q' be a pair of regular extended partial models out put Algorithm 5, and let $P' \succcurlyeq_{ref} Q'$ for some refinement ref. Then $\lfloor P' \rfloor \succcurlyeq_{ref} \lfloor Q' \rfloor$.

Proof. Let $P' = \langle \mathcal{O}_{P'}, \mathcal{I}_{P'}, \mathcal{S}_{P'} \rangle$, $Q' = \langle \mathcal{O}_{Q'}, \mathcal{I}_{Q'}, \mathcal{S}_{Q'} \rangle$, $P = \lfloor P' \rfloor = \langle \mathcal{O}_{P'}, \mathcal{I}_P, \mathcal{S}_P \rangle$, and $Q = \lfloor Q' \rfloor = \langle \mathcal{O}_{Q'}, \mathcal{I}_Q, \mathcal{S}_Q \rangle$. Since the object sets $\mathcal{O}_{P'}$ and $\mathcal{O}_{Q'}$ are unchanged by projection, P and Q satisfy both SR1 and SR2. Moreover, after removing the symbols in $\Sigma' \setminus \Sigma$ from $\mathcal{I}_{P'}$ and $\mathcal{I}_{Q'}$, $\mathcal{I}_{P'}$ and $\mathcal{I}_{Q'}$ satisfying SR3 implies that $\mathcal{I}_P$ and $\mathcal{I}_Q$ also satisfy SR3.

By SR3, $k \vDash \mathcal{S}_{Q'}$ implies that $k \vDash \mathcal{S}_{P'}[ref]$. Due to the regularity of P', the substitutions $\sum_{\langle p_1,q_1\rangle\ldots,\langle p_{\alpha(\varsigma)},q_{\alpha(\varsigma)}\rangle\in ref} \hat{\varsigma}(q_1,\ldots,q_{\alpha(\varsigma)})$ under $[ref]$ are disjoint for distinct $\hat{\varsigma}(p_1,\ldots,p_{\alpha(\varsigma)})$ by Lemma 2.28. Therefore, applying Fourier–Motzkin elimination to $\mathcal{S}_{P'}[ref]$ is equivalent to first applying to $\mathcal{S}_{P'}$ to obtain $\mathcal{S}_P$, and then performing the substitution $\mathcal{S}_P[ref]$.

By the properties of Fourier–Motzkin elimination [Sch98, pp. 155-157], the systems of linear inequalities $\mathcal{S}_{Q'}$ and $\mathcal{S}_Q$, as well as $\mathcal{S}_{P'}[ref]$ and $\mathcal{S}_P[ref]$ are equi-satisfiable. Firstly, for any $k' \vDash \mathcal{S}_{Q'}$ (resp. $k' \vDash \mathcal{S}_{P'}[ref]$), we also have $k'\restriction_{\mathcal{X}_Q} \vDash \mathcal{S}_Q$ (resp. $k'\restriction_{\mathcal{X}_Q} \vDash \mathcal{S}_P[ref]$), where $k'\restriction_{\mathcal{X}_Q}$ is the projection of the variable binding k' to the domain $\mathcal{X}_Q$. Moreover, for any $k \vDash \mathcal{S}_Q$ (resp. $k \vDash \mathcal{S}_P[ref]$), there is some $k' \vDash \mathcal{S}_{Q'}$ (resp. $k' \vDash \mathcal{S}_{P'}[ref]$) such that $k'\restriction_{\mathcal{X}_Q} = k$.

Therefore, for any variable binding k, we have

$$k \vDash \mathcal{S}_Q \overset{\text{by F–M elimination}}{\Longrightarrow} k' \vDash \mathcal{S}_{Q'} \overset{\text{by SR3}}{\Longrightarrow} k' \vDash \mathcal{S}_{P'}[ref] \overset{\text{by F–M elimination}}{\Longrightarrow} k'\restriction_{\mathcal{X}_Q} = k \vDash \mathcal{S}_P[ref]$$

as required. □

Lemma C.2 Let P' and $\mathcal{T}'$ be a regular extended partial model and theory, respectively, output by Algorithm 5 for the regular scoped partial model P and theory $\mathcal{T}$, and $M' \in \mathit{solutions}(P', \mathcal{T}')$. Then

a. $M = \lfloor M' \rfloor \in \mathit{solutions}(P, \mathcal{T})$;
b. $M'' = \lceil M \rceil = \lceil \lfloor M' \rfloor \rceil \geqslant_{id} M'$, where id is the *identity relation* on $O_{M'} = O_{M''}$, i.e., $id = \{\langle o, o\rangle \mid o \in O_{M'}\}$; and
c. $\max \hat{\mathrm{X}}^*$ *subject to* $\mathcal{S}_{M'} \leq \max \hat{\mathrm{X}}^*$ *subject to* $\mathcal{S}_{M''}$.

Proof. Case a: Notice that $\mathcal{T}'$ was formed by adding further predicates to $\mathcal{T}$ but removing no predicates or error patterns. If M' is concrete and compatible with $\mathcal{T}'$, then so is M concrete and compatible with $\mathcal{T}$, since the removed interpretations and inequalities do not affect concreteness or the compatibility with the predicates and error patterns already existing in $\mathcal{T}$. By $P' \geqslant M'$ and Lemma C.1, we may also conclude that $P \geqslant M$ as required.

Case b: Let $M' = \langle O_{M'}, \mathcal{I}_{M'}, \mathcal{S}_{M'}\rangle$, $M = \langle O_{M'}, \mathcal{I}_{M'}, \mathcal{S}_M\rangle$,and $M'' = \langle O_{M'}, \mathcal{I}_{M'}, \mathcal{S}_{M''}\rangle$ and observe that id trivially satisfies SR1–SR3. To check SR4, notice that $\mathcal{S}_{M'}[id] = \mathcal{S}_{M'}$. Each linear inequality in $\mathcal{S}_{M''}$ is already present in $\mathcal{S}_{M'}$: either they belong to $\mathcal{S}_M$, or were added to $\mathcal{S}_{M''}$ (and thus to $\mathcal{S}_{M'}$, since M' itself is also an extended partial model) by Algorithm 5. Thus, $\mathcal{S}_{M''} \vDash \mathcal{S}_{M'} = \mathcal{S}_{M'}[id]$ as required.

Case c: Assume that $\max \hat{\mathrm{X}}^*$ *subject to* $\mathcal{S}_{M'} > \max \hat{\mathrm{X}}^*$ *subject to* $\mathcal{S}_{M''}$, i.e., there is some $k' \vDash \mathcal{S}_{M'}$ such that $k'(\hat{\mathrm{X}}^*) > k''(\hat{\mathrm{X}}^*)$ for all $k'' \vDash \mathcal{S}_{M''}$. We have $\mathcal{S}_{M'} \vDash \mathcal{S}_{M''}$ by Case b, which means that $k' \vDash \mathcal{S}_{M'}$ implies $k' \vDash \mathcal{S}_{M''}$. This is a contradiction: by picking $k'' = k'$, we would have $k'(\hat{\mathrm{X}}^*) > k'(\hat{\mathrm{X}}^*)$. Therefore, we must have had $\max \hat{\mathrm{X}}^*$ *subject to* $\mathcal{S}_{M'} \leq \max \hat{\mathrm{X}}^*$ *subject to* $\mathcal{S}_{M''}$ originally as required. □

Lemma C.3 Let P, Q be a pair of regular scoped partial models, and let $P \geqslant_{ref} Q$ for some refinement ref. Then $\lceil P \rceil \geqslant_{ref} \lceil Q \rceil$.

Proof. Let $P = \langle O_P, \mathcal{I}_P, \mathcal{S}_P\rangle$, $Q = \langle O_Q, \mathcal{I}_Q, \mathcal{S}_Q\rangle$, $P' = \lceil P \rceil = \langle O_P, \mathcal{I}_{P'}, \mathcal{S}_{P'}\rangle$, and $Q' = \lceil Q \rceil = \langle O_Q, \mathcal{I}_{Q'}, \mathcal{S}_{Q'}\rangle$. Since the object sets O_P and O_Q are unchanged by Algorithm 5, P' and Q' satisfy both SR1 and SR2. Moreover, since we set the interpretations of the newly added $\Sigma' \setminus \Sigma$ symbols in $\mathcal{I}_{P'}$ and $\mathcal{I}_{Q'}$ to $\frac{1}{2}$, $\mathcal{I}_P$ and $\mathcal{I}_Q$ satisfying SR3 implies that $\mathcal{I}_{P'}$ and $\mathcal{I}_{Q'}$ also satisfy SR3.

To see that $\mathcal{S}_{Q'} \vDash \mathcal{S}_{P'}[ref]$, i.e., $\mathcal{S}_{P'}$ and $\mathcal{S}_{Q'}$ satisfy SR4, we proceed by checking each linear inequality $\left[\sum_{\mathfrak{x}_i\in\mathcal{X}_{P'}} a_i \cdot \mathfrak{x}_i \leq b\right] \in \mathcal{S}_{P'}$ individually. For any linear inequality also in $\mathcal{S}_P$, we may appeal to the fact that $\mathcal{S}_P$ and $\mathcal{S}_Q$ satisfies SR4. For the inequalities of the forms $\sum_{\mathfrak{x}_i\in\mathcal{X}_{\mathrm{IPET}}} a_i \cdot \hat{\mathrm{X}}_{\mathfrak{x}_i} \leq b$ and $hat\mathrm{X}^* = \sum_{\mathfrak{x}_i\in\mathcal{X}_{\mathrm{IPET}}} c_i \cdot \hat{\mathrm{X}}_{\mathfrak{x}_i}$ added to $\mathcal{S}_{P'}$ in lines 6 and 7 of Algorithm 5,

respectively, we may notice that all the symbols they refer to are of arity 0. Therefore, they are unchanged by the refinement $P \succcurlyeq Q$ or the substitution $[ref]$.

For the linear equalities of the forms

$$\sum_{e=\langle n_1,n_2\rangle\in E, tr(n_1)=bb} \hat{\mathsf{X}}_{\mathsf{f}(e)} = \#[\mathsf{Y}_{bb}(*,\dots,*)]^{P'} + \#[\mathsf{Y}'_{bb}(*,\dots,*)]^{P'},$$
$$\sum_{e=\langle n_1,n_2\rangle\in E, tr(n_1)=bb} \hat{\mathsf{X}}_{\mathsf{f}(e)} = \#[\mathsf{Y}_{bb}(*,\dots,*)]^{P'}$$

added in lines 11 and 14, respectively, we only have to discuss the right sides of the equalities, as the left sides contain only 0-ary symbols. For brevity, we only discuss the latter case (with a single count aggregations), as the former case (with two count aggregations) is analogous. By Lemma 2.28, the substitution $[ref]$ changes $\#[\mathsf{Y}_{bb}(*,\dots,*)]^{P'}$ into $\#[\mathsf{Y}_{bb}(*,\dots,*)]^{P'}$, because the mask $*,\dots,*$ is focused at the empty variable binding $\emptyset$ (omitted from the count aggregation notation). Therefore,

$$\left[\sum_{e=\langle n_1,n_2\rangle\in E, tr(n_1)=bb} \hat{\mathsf{X}}_{\mathsf{f}(e)} = \#[\mathsf{Y}_{bb}(*,\dots,*)]^{P'}\right] \in \mathcal{S}_{P'}$$

implies

$$\left[\sum_{e=\langle n_1,n_2\rangle\in E, tr(n_1)=bb} \hat{\mathsf{X}}_{\mathsf{f}(e)} = \#[\mathsf{Y}_{bb}(*,\dots,*)]^{Q'}\right] \in \mathcal{S}_{P'}[ref].$$

Since we also have

$$\left[\sum_{e=\langle n_1,n_2\rangle\in E, tr(n_1)=bb} \hat{\mathsf{X}}_{\mathsf{f}(e)} = \#[\mathsf{Y}_{bb}(*,\dots,*)]^{Q'}\right] \in \mathcal{S}_{Q'},$$

we may see that $\mathcal{S}_{Q'} \vDash \mathcal{S}_{P'}[ref]$ as required. □

Lemma C.4 Let P' and $\mathcal{T}'$ be an extended partial model and theory, respectively, output by Algorithm 5 for the scoped partial model P and theory $\mathcal{T}$, and $M \in solutions(P,\mathcal{T})$. Then

a. $M' = \lceil M \rceil \in solutions(P',\mathcal{T}')$; and
b. $\max \hat{\mathsf{X}}^*$ *subject to* $\mathcal{S}_{M'} = DS_M(M)$.

Proof. Case a: Notice that M' is concrete and compatible with $\mathcal{T}'$, since M is also concrete and compatible with $\mathcal{T}$. By Lemma C.3 and $P \succcurlyeq M$ from Case a, we have $P' \succcurlyeq M'$. Therefore, $M' \in solutions(P',\mathcal{T}')$ as required.

Case b: We will show that any variable binding $k_1 \colon \mathcal{X}_{\text{IPET}} \to \mathbb{N}$ with $k_1 \vDash \mathcal{S}_{\text{IPET}}$ in the optimization problem $DS_M(M) = \max \sum_{\mathfrak{x}_i} c_i \cdot \mathfrak{x}_i$ *subject to* $\mathcal{S}_{\text{IPET}}$ corresponds to a variable binding $k_2 \colon \mathcal{X}_{M'} \to \mathbb{N}$ with $k_2 \vDash \mathcal{S}_{M'}$, such that $\sum_{\mathfrak{x}_i \in \mathcal{X}_{\text{IPET}}} c_i \cdot k_1(\mathfrak{x}_i) = k_2(\hat{\mathsf{X}}^*)$, and vice versa. Therefore, the optima of the two linear programs must be the same.

$k_1 \mapsto k_2$: Let us pick an arbitrary $k_3 \colon \mathcal{X}_M \to \mathbb{N}$ such that $k_3 \vDash \mathcal{S}_M$ (since M is concrete, at least one such binding must exist). Let us define

$$k_2(\mathfrak{x}) = \begin{cases} k_3(\mathfrak{x}) & \text{if } \mathfrak{x} \in \mathcal{X}_M, \\ 1 & \text{if } \mathfrak{x} = \hat{\mathsf{Y}}_{bb}(o_1,\dots,o_{\alpha'(\mathsf{Y}_{bb})}) \text{ and } \mathcal{I}_{M''}(\mathsf{Y}_{bb})(_1,\dots,o_{\alpha'(\mathsf{Y}_{bb})}) = 1 \\ & \quad \text{or } \mathfrak{x} = \hat{\mathsf{Y}}'_{bb}(o_1,\dots,o_{\alpha'(\mathsf{Y}'_{bb})}) \text{ and } \mathcal{I}_{M''}(\mathsf{Y}_{bb})(_1,\dots,o_{\alpha'(\mathsf{Y}_{bb})}) = 1, \\ 0 & \text{if } \mathfrak{x} = \hat{\mathsf{Y}}_{bb}(o_1,\dots,o_{\alpha'(\mathsf{Y}_{bb})}) \text{ and } \mathcal{I}_{M''}(\mathsf{Y}_{bb})(_1,\dots,o_{\alpha'(\mathsf{Y}_{bb})}) = 0 \\ & \quad \text{or } \mathfrak{x} = \hat{\mathsf{Y}}'_{bb}(o_1,\dots,o_{\alpha'(\mathsf{Y}'_{bb})}) \text{ and } \mathcal{I}_{M''}(\mathsf{Y}_{bb})(_1,\dots,o_{\alpha'(\mathsf{Y}_{bb})}) = 0, \\ k_1(\mathfrak{z}) & \text{if } \mathfrak{x} = \hat{\mathsf{X}}_{\mathfrak{z}} \text{ for some } \mathfrak{z} \in \mathcal{X}_{\text{IPET}}. \end{cases}$$

Notice that we have $k_2 \vDash \mathcal{S}_{M''}$: Firstly, k_2 satisfies any linear inequalities from $\mathcal{S}_M$, because we have $k_3 \vDash \mathcal{S}_M$. Moreover, k_2 satisfies any inequalities added to $\mathcal{S}_{M'}$ by Algorithm 5, since these

inequalities either correspond to some inequality from $\mathcal{S}_{\text{IPET}}$ and we have $k_1 \vDash \mathcal{S}_{\text{IPET}}$, or we take advantage of the fact that $k_2 \vDash \#[\mathsf{Y}_{bb}(*,\ldots,*)]^{M'} = M'\#\psi_{bb}$ and $k_2 \vDash \#[\mathsf{Y}'_{bb}(*,\ldots,*)]^{M'} = M'\#\psi'_{bb}$ for any basic block or loop header bb.

$k_2 \mapsto k_1$: Let us define $k_1(\mathfrak{z}) = k_2(\hat{\mathsf{X}}_{\mathfrak{z}})$ and notice that we have $k_1 \vDash \mathcal{S}_{\text{IPET}}$ by an argument similar to the one above. □

Proposition 6.8 (Witness model) Let $DS_M(M)$ be the domain-specific WCET estimate of a query program q obtained by Algorithm 4 for a concrete model M, DS_P be the domain-specific WCET estimate of q for a regular scoped partial model P and theory $\mathcal{T}$ by Algorithm 5, and M^* be the witness model for the WCET of q, i.e., the optimal solution of DS_P. Then $\lfloor M^* \rfloor \in \mathit{solutions}(P,\mathcal{T})$ and $DS_M(M) \leq DS_M(\lfloor M^* \rfloor) = DS_P$ for all $M \in \mathit{solutions}(P,\mathcal{T})$.

Proof. $\underline{\lfloor M^* \rfloor \in \mathit{solutions}(P,\mathcal{T})}$: Follows immediately from Lemma C.2a.

$\underline{DS_M(M) \leq DS_P}$: Since M^* is an optimal solution of DS_P, we have $DS_P = \max_{k \vDash \mathcal{S}_{M^*}} k(\hat{\mathsf{X}}^*)$ and $\max_{k \vDash \mathcal{S}_{M'}} k(\hat{\mathsf{X}}^*) \leq DS_P$ for all extended partial models $M' \in \mathit{solutions}(P',\mathcal{T}')$. In particular, we may pick $M' = \lceil M \rceil$ by Lemma C.4a. By Lemma C.4b, $DS_M(M) = \max_{k \vDash \mathcal{S}_{\lceil M \rceil}} k(\hat{\mathsf{X}}^*) \leq DS_P$.

$\underline{DS_M(M^*) = DS_P}$: Assume the contrary, i.e., $DS_M(\lfloor M^* \rfloor) < DS_P$. By Lemma C.4b and Lemma C.2c, we have

$$DS_P = \max \hat{\mathsf{X}}^* \textit{ subject to } \mathcal{S}_{M^*} \leq \max \hat{\mathsf{X}}^* \textit{ subject to } \mathcal{S}_{\lceil \lfloor M^* \rfloor \rceil} = DS_M(\lfloor M^* \rfloor) < DS_P.$$

This is a contradiction, we must have had $DS_M(M^*) = DS_P$ originally as required. □

Proposition 6.6 (Safety and tightness) Let $\tau(M)$ be the execution time of a query program q on a concrete model M, P be regular scoped partial model, $\mathcal{T}$ be a theory, and

$$CL = \max \sum_{\mathfrak{x}_i \in \mathcal{X}} c_i \cdot \mathfrak{x}_i \textit{ subject to } \mathcal{S}_{\text{IPET}}, \qquad DS_P = \max \mathsf{X}^* \textit{ subject to } \langle P', \mathcal{T}' \rangle,$$

where CL is the classical IPET estimated obtained from q, and DS_P is the domain-specific estimate based on the extended graph generation problem form Algorithm 5. Then $\tau(M) \leq DS_P \leq CL$ for all $M \in \mathit{solutions}(P,\mathcal{T})$.

Proof. By Proposition 6.8, we have $DS_M(M) \leq DS_P = DS_M(M^*)$ for all $m \in \mathit{solutions}(p, \sqcup)$. By applying Proposition 6.5 to any $M \in \mathit{solutions}(P,\mathcal{T})$, we have $\tau(M) \leq DS_M(M)$, and by applying it to M^* in particular, we have $DS_M(M^*) \leq CL$. This completes the proof $\tau(M) \leq DS_M(M) \leq DS_P = DS_M(M^*) \leq CL$. □

Proposition 6.9 (Tightening by refinement) Let $DS_P(P,\mathcal{T})$ denote the domain-specific WCET estimate of a query program q for a regular scoped partial model P and theory $\mathcal{T}$ obtained by Algorithm 5 and $P \succcurlyeq Q$ for some regular scoped partial model Q. Then $DS_P(Q,\mathcal{T}) \leq DS_P(P,\mathcal{T})$. In particular, if $P = P_{init}$ is the initial partial model for a metamodel $\langle \Sigma, \Gamma, \alpha \rangle$ from Section 4.2.2, then we may see that the WCET estimate for any partial model conforming to the metamodel is at least as tight as the DS_Σ estimate for the metamodel.

Proof. Assume that $DS_P(Q,\mathcal{T}) > DS_P(P,\mathcal{T})$. Let M^* and N^* be the witness models corresponding to $DS_P(Q,\mathcal{T})$ and $DS_P(P,\mathcal{T})$, respectively, and notice that we have $P \succcurlyeq Q \succcurlyeq \lfloor N^* \rfloor$. Therefore, by Proposition 6.8, we have

$$DS_M(\lfloor N^* \rfloor) \leq DS_M(\lfloor M^* \rfloor) = DS_P(P,\mathcal{T}) < DS_P(Q,\mathcal{T}) = DS_M(\lfloor N^* \rfloor),$$

which is a contradiction. We must have had $DS_P(Q,\mathcal{T}) \leq DS_P(P,\mathcal{T})$ originally as required. □