Universität Rostock

Traditio et Innovatio

Model-based Symbolic Design Space Exploration at the Electronic System Level

– A Systematic Approach –

Dissertation zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

VORGELEGT VON:
M.Sc. Kai Neubauer,
geboren am 10.05.1991 in Demmin

EINGEREICHT AM:
17.08.2021

https://doi.org/10.18453/rosdok_id00003664

UNIVERSITÄT ROSTOCK | FAKULTÄT FÜR INFORMATIK UND ELEKTROTECHNIK
Institut für Angewandte Mikroelektronik und Datentechnik
Richard-Wagner-Str. 31, 18119 Rostock-Warnemünde | Tel: +49 381 498-7251

Gutachter

- Prof. Dr.-Ing. habil. Christian Haubelt
  Lehrstuhl “Eingebettete Systeme”
  Institut für Angewandte Mikroelektronik und Datentechnik
  Universität Rostock

- Prof. Dr. Torsten Schaub
  Wissensverarbeitung und Informationssysteme
  Institut für Informatik und Computational Science
  Universität Potsdam

- Prof. Dr.-Ing. Michael Glaß
  Institut für Eingebettete Systeme / Echtzeitsysteme
  Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
  Universität Ulm

Datum der Abgabe: 17.08.2021
Datum der Verteidigung: 11.03.2022

Acknowledgments

For the past six and a half years, many people have accompanied and supported me in the creation of the present work. I would like to take the opportunity here to express my sincere gratitude to all of them.

First and foremost, I would like to thank Prof. Dr. Christian Haubelt for providing me with the opportunity to work on this thesis topic. His continuous support and supervision – already beginning with my bachelor and master theses – was and is particularly appreciated. Without his valuable input, this work would not have been possible. Furthermore, my thanks go to Prof. Dr. Torsten Schaub at the University of Potsdam for the co-supervision, the helpful discussions regarding ASP, and accepting to review the thesis. I would also like to thank Prof. Dr. Michael Glaß at the University of Ulm, who accepted to review the thesis as well.

Many thanks go to my colleagues at the Institute of Applied Microelectronics and Computer Engineering for the consistently pleasant working atmosphere and the support through uncountable discussions and activities – both work-related and non-work-related. In particular, but also representatively for the whole team, I would like to thank Dr. Florian Grützmacher, Dr. Henning Puttnies, Dr. Michael Rethfeldt, Daniel Gis, Benjamin Beichler, and Luise Müller for their valuable input regarding the present thesis and all related topics. Also, I would like to thank Philipp Wanko at the University of Potsdam for the fruitful cooperation in the mutual project that led to this thesis.

Finally, my special thanks go to my parents, Holger and Ramona Neubauer, and my friends for motivating and supporting me in many ways. In anticipation of forgetting someone important, I refrain from mentioning you explicitly. You know who you are!

– Thank you!

Abstract

Today, computer systems are ubiquitously integrated in nearly all areas of everyday life. Well ahead of conventional general purpose computers, embedded systems are dominating the global market. They are present in telecommunication equipment, in current and future automotive and aviation technology, household appliances, and infrastructure. The design of those embedded computer systems is continuously becoming more complex as the requirements with respect to their functional and extra-functional properties grow steadily. Simultaneously, the adaption to technological progress demands for shorter development cycles. Hence, an efficient design process is imperative to fulfill these requirements. A bottom-up design strategy, where every detail of the system can be explored and evaluated, has not been viable for a long time now. Thus, the design process has been raised to higher abstraction levels, where large-scaled decisions can be explored more effectively. Decisions are first explored at the electronic system level (ESL), before the results are refined at lower abstractions. However, even at the ESL, the exploration of design alternatives is not a trivial task. It has to be decided, which resources are allocated, where the functionality is implemented, and how the communication infrastructure is realized. Depending on these decisions, the properties of the design have to be determined and evaluated. Typically, an embedded computer system must fulfill several constraints and shall be optimal with respect to multiple objectives. These objectives are often conflicting with each other such that no single optimal solution but a set of compromise, e.g., Pareto-optimal, solutions exists. Therefore, the design of embedded computer systems is considered a multi-objective optimization problem. Automatic approaches are mandatory to obtain Pareto-optimal designs, as the manual exploration of the whole design space is not feasible. To this end, the problem is typically encoded through mathematical models that allow for the evaluation of desired properties. For the exploration, many approaches rely on population-based heuristics that traverse the design space on the basis of random decisions. While this methodology has been shown to work well for many scenarios, it holds significant disadvantages. First, as the search is generally not executed systematically, the approaches cannot guarantee to explore the entire design space and, therefore, may miss optimal solutions. Furthermore, the approaches usually cannot identify already explored solutions. That is, the same design candidates are re-visited again even if they already have been evaluated and found to be non-optimal before. Second, as most population-based approaches are based on the re-combination of previously found solutions, they tend to run in saturation, exploring the same regions of design space over and over. Finally, the search is decoupled from the exploration. This prevents to exchange information between the individual steps that could allow to steer the search into more promising regions.

In this thesis, a novel, fully systematic approach is proposed that aims to address the outlined problems. The multi-objective optimization problem is encoded symbolically through a concise answer set programming (ASP) formulation. The programming paradigm ASP stems from the area of knowledge representation and reasoning and permits the systematic exploration of

the search space. While linear feasibility constraints can be directly verified in ASP, non-linear objectives are hard to formulate in standard ASP. Therefore, in the thesis at hand, several specialized solvers are tightly coupled as background theories with the foreground ASP solver under the ASP modulo theories (ASPmT) paradigm. The tight coupling of foreground and background theories allows the exchange of information between the previously isolated problems. That is, reasons for invalid design points found in one of the background theories are used to steer the search in the foreground theory and prune the search more effectively. Furthermore, the coupling of disparate solvers allows for the evaluation of partial assignments where only a subset of decisions has been made. This leads to an earlier detection of invalid regions in the search space. The conducted experiments show a significant advantage of using partial assignment checking when compared to an approach where only complete design can be evaluated. The ability to work on partial assignments not only benefits the evaluation of acquired solutions but also the Pareto-filtering as non-optimal solutions can be identified early.

However, the utilization of partial assignments imposes disadvantages as evaluations and Pareto checks have to be executed more regularly. As a remedy, the thesis at hand proposes specialized archive management and evaluation methodologies, that aim at reducing this overhead. It is shown, only when a complete solution is checked for optimality, the entire Pareto filtering process has to be executed. This leads to a significant reduction in required operations for partial solutions while still maintaining correct results. Similarly, a specialized approximation technique is proposed that reduces the number of unnecessary expensive evaluations. With this approach, up to 98 % of all evaluations can be skipped and the overall performance of the design space exploration can be increased by up to six times for the considered use cases.

To summarize, the thesis at hand provides a holistic framework for designing embedded computer systems at the ESL. By utilizing the ASPmT paradigm, the search is executed entirely systematically and the disparate synthesis steps can be coupled to explore the search space effectively.

Kurzfassung

Computersysteme sind heute allgegenwärtig und in fast allen Bereichen des täglichen Lebens integriert. Weit vor den herkömmlichen Allzweckcomputern dominieren eingebettete Systeme den Weltmarkt. Sie sind in Telekommunikationsgeräten, in der Automobil- und Luftfahrttechnik, in Haushaltsgeräten und in der Infrastruktur zu finden. Der Entwurf dieser eingebetteten Computersysteme wird immer komplexer, da die Anforderungen an ihre funktionalen und extrafunktionalen Eigenschaften stetig wachsen. Gleichzeitig verlangt die Anpassung an den technologischen Fortschritt nach kürzeren Entwicklungszyklen. Um diesen Anforderungen gerecht zu werden, ist ein effizienter Entwurfsprozess unabdingbar. Eine Bottom-up-Entwurfsstrategie, bei der jedes Detail des Systems exploriert und bewertet werden kann, ist schon seit langem nicht mehr praktikabel. Daher wurde der Entwurfsprozess auf höhere Abstraktionsebenen angehoben, auf denen Entscheidungen von großer Tragweite effektiver untersucht werden können. Entscheidungen werden zunächst auf der Systemebene untersucht, bevor die Ergebnisse auf niedrigeren Abstraktionsebenen verfeinert werden. Doch selbst auf der Systemebene ist die Exploration von Entwurfsalternativen keine triviale Aufgabe. Es muss entschieden werden, welche Ressourcen alloziert werden, wo die Funktionalität implementiert wird und wie die Kommunikationsinfrastruktur realisiert wird. Abhängig von diesen Entscheidungen müssen die Eigenschaften des Entwurfs bestimmt und bewertet werden. Typischerweise muss ein eingebettetes Computersystem mehrere Randbedingungen erfüllen und soll in Bezug auf mehrere Ziele optimal sein. Diese Ziele stehen oft in einem Konflikt zueinander, sodass keine einzelne optimale Lösung, sondern eine Menge von Kompromisslösungen, d.h. Pareto-optimale Lösungen, existieren. Daher wird der Entwurf von eingebetteten Computersystemen als ein mehrzieliges Optimierungsproblem betrachtet. Automatische Ansätze sind zwingend erforderlich, um alle Pareto-optimalen Entwurfspunkte zu erhalten, da die manuelle Exploration des gesamten Entwurfsraums nicht durchführbar ist. Zu diesem Zweck wird das Problem typischerweise durch mathematische Modelle modelliert, die die Auswertung der gewünschten Eigenschaften ermöglichen. Für die Exploration verlassen sich viele Ansätze auf populationsbasierte heuristische Verfahren, die den Entwurfsraum auf der Basis von Zufallsentscheidungen durchlaufen. Obwohl sich diese Methodik für viele Szenarien als gut geeignet erwiesen hat, birgt sie erhebliche Nachteile. Erstens können die Ansätze, da die Suche im Allgemeinen nicht systematisch ausgeführt wird, nicht garantieren, dass der gesamte Entwurfsraum durchsucht wird, und verpassen daher möglicherweise optimale Lösungen. Außerdem können die Ansätze oftmals keine bereits gefundenen Lösungen identifizieren. Das heißt, dieselben Entwurfskandidaten werden erneut untersucht, auch wenn sie bereits zuvor bewertet und als nicht optimal befunden wurden. Zweitens, da die meisten populationsbasierten Ansätze auf der Rekombination von zuvor gefundenen Lösungen basieren, neigen sie dazu, in die Sättigung zu laufen und dieselben Regionen des Suchraums immer wieder zu durchlaufen. Schließlich ist die Suche von der Exploration entkoppelt. Dies verhindert den Austausch von Informationen zwischen den einzelnen Schritten, die es erlauben könnten, die Suche in vielversprechendere Regionen zu lenken.

In dieser Arbeit wird ein neuartiger, vollständig systematischer Ansatz vorgeschlagen, der darauf abzielt, die skizzierten Probleme zu lösen. Das Mehrzieloptimierungsproblem wird symbolisch durch eine prägnante ASP-Formulierung kodiert. Das Programmierparadigma ASP stammt aus dem Bereich der Wissensrepräsentation und Logik und erlaubt die systematische Exploration des Suchraums. Während lineare Randbedingungen in ASP direkt verifiziert werden können, sind nicht-lineare Ziele in Standard-ASP schwer zu formulieren. Daher werden in der vorliegenden Arbeit mehrere spezialisierte Solver als Hintergrundtheorien eng mit dem Vordergrund-ASP-Solver unter dem ASPmT-Paradigma gekoppelt. Die enge Kopplung von Vordergrund- und Hintergrundtheorien ermöglicht den Austausch von Informationen zwischen den zuvor isolierten Problemen. Das heißt, Gründe für ungültige Entwurfspunkte, die in einer der Hintergrundtheorien gefunden werden, werden verwendet, um die Suche in der Vordergrundtheorie zu steuern und diese effektiver zu beschneiden. Darüber hinaus ermöglicht die Kopplung von disparaten Solvern die Auswertung von Teillösungen, bei denen nur eine Teilmenge von Entscheidungen getroffen wurde. Dies führt zu einer früheren Erkennung von ungültigen Regionen im Suchraum. Die durchgeführten Experimente zeigen einen signifikanten Vorteil der Prüfung von Teillösungen im Vergleich zu einem Ansatz, bei dem nur der komplette Entwurf ausgewertet werden kann. Die Möglichkeit, mit Teillösungen zu arbeiten, kommt nicht nur der Bewertung der gefundenen Lösungen zugute, sondern auch der Pareto-Filterung, da nicht-optimale Lösungen frühzeitig erkannt werden können.

Allerdings bringt die Verwendung von Teillösungen auch Nachteile mit sich, da Auswertungen und Optimalitätsprüfungen regelmäßiger durchgeführt werden müssen. Als Abhilfe schlägt die vorliegende Arbeit spezielle Archivverwaltungs- und Auswertungsmethoden vor, die darauf abzielen, diesen Overhead zu reduzieren. Es wird gezeigt, dass nur dann, wenn eine vollständige Lösung auf Optimalität geprüft wird, der gesamte Prozess der Pareto-Filterung ausgeführt werden muss. Dies führt zu einer signifikanten Reduktion der erforderlichen Operationen für Teillösungen bei gleichzeitiger Beibehaltung korrekter Ergebnisse. In ähnlicher Weise wird eine spezialisierte Approximationstechnik vorgeschlagen, die die Anzahl der unnötigen, teuren Auswertungen reduziert. Mit diesem Ansatz können bis zu 98 % aller Evaluationen übersprungen werden und die Gesamtleistung der Entwurfsraumexploration kann für die betrachteten Anwendungsfälle um das bis zu Sechsfache gesteigert werden.

Zusammenfassend lässt sich sagen, dass die vorliegende Arbeit einen ganzheitlichen Rahmen für den Entwurf von eingebetteten Computern auf der Systemebene bereitstellt. Durch die Verwendung des ASPmT-Paradigmas wird die Suche vollständig systematisch ausgeführt und die unterschiedlichen Syntheseschritte können gekoppelt werden, um den Suchraum effektiv zu durchsuchen.

Contents

List of Figures

List of Tables

Author's Publications

Authored

[1] Kai Neubauer, Christian Haubelt, and Michael Glaß. "Supporting Composition in Symbolic System Synthesis". In: *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. Samos, Greece, July 2016, pp. 132–139. DOI: 10.1109/SAMOS.2016.7818340.

Abstract: Typically, state-of-the-art approaches in system synthesis do not consider the trend in embedded systems design towards systems-of-systems where optimized subsystems exist from previous projects or as 3 rd party IP. In this paper, we propose a novel top-down system synthesis approach with additional support for the composition of subsystems that is based on the use of hierarchical mapping edges and a list-based scheduling algorithm using distributed priority queues. The proposed method not only enables composition of existing subsystems, but experimental results also show a significant reduction of the design space while maintaining a good quality of the implemented systems. Especially for large network-on-chip systems (NoC), our approach outperforms an existing top-down methodology in solving time by nearly 50% and in average quality by 11%.

[2] Kai Neubauer, Philipp Wanko, Torsten Schaub, and Christian Haubelt. "Enhancing Symbolic System Synthesis through ASPmT with Partial Assignment Evaluation". In: *Design, Automation and Test in Europe Conference (DATE)*. Lausanne, Switzerland, Mar. 2017, pp. 306–309. DOI: 10.23919/DATE.2017.7927005.

Abstract: The design of embedded systems is becoming continuously more complex such that efficient system-level design methods are becoming crucial. Recently, combined Answer Set Programming (ASP) and Quantifier Free Integer Difference Logic (QF-IDL) solving has been shown to be a promising approach in system synthesis. However, this approach still has several restrictions limiting its applicability. In the paper at hand, we propose a novel ASP modulo Theories (ASPmT) system synthesis approach, which (i) supports more sophisticated system models, (ii) tightly integrates the QF-IDL solving into the ASP solving, and (iii) makes use of partial assignment checking. As a result, more realistic systems are considered and an early exclusion of infeasible solutions improves the entire system synthesis.

[3] Kai Neubauer, Christian Haubelt, Philipp Wanko, and Torsten Schaub. "Utilizing Quad-Trees for Efficient Design Space Exploration with Partial Assignment Evaluation". In: *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jeju, Korea, Jan. 2018, pp. 434–439. DOI: 10.1109/ASPDAC.2018.8297362.

Abstract: Recently, it has been shown that constraint-based symbolic solving techniques offer an efficient way for deciding binding and routing options in order to obtain a feasible system level implementation. In combination with various background theories, a feasibility analysis of the resulting system may already be performed on partial solutions. That is, infeasible subsets of mapping and routing options can be pruned early in the decision process, which fastens the solving accordingly. Allowing a proper design space exploration including multi-objective optimization also requires an efficient structure for storing and managing non-dominated solutions. In this work, we propose and study the usage of the Quad-Tree

data structure in the context of partial assignment evaluation during system synthesis. Out experiments show that unnecessary dominance checks can be avoided, which indicates a preference of Quad-Trees over a commonly used list-based implementation for large combinatorial optimization problems.

[4] Kai Neubauer, Philipp Wanko, Torsten Schaub, and Christian Haubelt. "Exact Multi-Objective Design Space Exploration using ASPmT". In: *Design, Automation and Test in Europe Conference (DATE)*. Dresden, Germany, Mar. 2018, pp. 257–260. DOI: 10.23919/DATE.2018.8342014.

Abstract: An efficient Design Space Exploration (DSE) is imperative for the design of modern, highly complex embedded systems in order to steer the development towards optimal design points. The early evaluation of design decisions at system-level abstraction layer helps to find promising regions for subsequent development steps in lower abstraction levels by diminishing the complexity of the search problem. In recent works, symbolic techniques, especially Answer Set Programming (ASP) modulo Theories (ASPmT), have been shown to find feasible solutions of highly complex system-level synthesis problems with non-linear constraints very efficiently. In this paper, we present a novel approach to a holistic system-level DSE based on ASPmT. To this end, we include additional background theories that concurrently guarantee compliance with hard constraints and perform the simultaneous optimization of several design objectives. We implement and compare our approach with a state-of-the-art preference handling framework for ASP. Experimental results indicate that our proposed method produces better solutions with respect to both diversity and convergence to the true Pareto front.

[5] Kai Neubauer, Christian Haubelt, Philipp Wanko, and Torsten Schaub. "Systematic Test Case Instance Generation for the Assessment of System-level Design Space Exploration Approaches". In: *21. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*. Tübingen, Germany, Mar. 2018. DOI: 10.15496/publikation-25685.

Abstract: The design of embedded systems gets continually more arduous as the complexity of applications and hardware platforms advance to satisfy the increasing demands on functionality, performance, and power consumption. Mostly however, the concurrent fulfillment of those demands are impossible because quality parameters are usually conflicting with each other and cannot be guaranteed simultaneously. Thus, to find the best compromises of all possible solutions, an efficient Design Space Exploration (DSE) becomes imperative. While, in recent time, many DSE techniques to the system-level synthesis problem of embedded systems design have been proposed, a systematic approach on how to produce a viable set of variant test cases with definite similar properties is not available. In this work, we therefore propose a methodology for the test case generation for DSE techniques and present a versatile and easily expendable benchmark generator based on Answer Set Programming (ASP) that is able to produce hard synthesis problem instances.

[6] Kai Neubauer, Christian Haubelt, Philipp Wanko, and Torsten Schaub. "Work-in-Progress: On Leveraging Approximations for Exact System-level Design Space Exploration". In: *International Conference on Hardware Software Codesign and System Synthesis (CODES/ISSS)*. Sept. 2018, pp. 1–2. DOI: 10.1109/CODESISSS.2018.8525974.

Abstract: In order to find good design points for embedded systems, an efficient exploration of the design space is imperative. The ever-increasing complexity of embedded systems, however, results in a deterioration of the overall exploration performance. The DSE essentially consists of two parts: (1) the search for feasible solutions and (2) the evaluation of found feasible solutions. While the search has been massively improved by ASPmT-based strategies, the evaluation emerges as the main bottleneck. Tragically, evaluating bad solutions takes as much time as evaluating good ones. Hence, in this paper we

study the utilization of approximations in the evaluation process integrated in an ASPmT-based DSE to identify bad solutions more quickly while still retaining the exact Pareto-front.

[7] Kai Neubauer, Benjamin Beichler, and Christian Haubelt. "Exact Design Space Exploration Based on Consistent Approximations". In: *Electronics* 9.7 (June 2020), p. 1057. ISSN: 2079-9292. DOI: 10.3390/electronics9071057.

Abstract: The aim of design space exploration (DSE) is to identify implementations with optimal quality characteristics which simultaneously satisfy all imposed design constraints. Hence, besides searching for new solutions, a quality evaluation has to be performed for each design point. This process is typically very expensive and takes a majority of the exploration time. As nearly all the explored design points are sub-optimal, most of them get discarded after evaluation. However, evaluating a solution takes virtually the same amount of time for both good and bad ones. That way, a huge amount of computing power is literally wasted. In this paper, we propose a solution to the aforementioned problem by integrating efficient approximations in the background of a DSE engine in order to allow an initial evaluation of each solution. Only if the approximated quality indicates a promising candidate, the time-consuming exact evaluation is executed. The novelty of our approach is that (1) although the evaluation process is accelerated by using approximations, we do not forfeit the quality of the acquired solutions and (2) the integration in a background theory allows sophisticated reasoning techniques to prune the search space with the help of the approximation results. We have conducted an experimental evaluation of our approach by investigating the dependency of the accuracy of used approximations on the performance gain. Based on 120 electronic system level problem instances, we show that our approach is able to increase the overall exploration coverage by up to six times compared to a conservative DSE whenever accurate approximation functions are available.

Co-Authored

[8] Christian Haubelt, Kai Neubauer, Torsten Schaub, and Philipp Wanko. "Design Space Exploration with Answer Set Programming". In: *KI - Künstliche Intelligenz*. Vol. 32. 2-3. Berlin Heidelberg: Springer Nature, May 2018, pp. 205–206. DOI: 10.1007/s13218-018-0530-3.

Abstract: The aim of our project design space exploration with answer set programming is to develop a general framework based on Answer Set Programming (ASP) that finds valid solutions to the system design problem and simultaneously performs Design Space Exploration (DSE) to find the most favorable alternatives. We leverage recent developments in ASP solving that allow for tight integration of background theories to create a holistic framework for effective DSE.

[9] Joachim Falk, Kai Neubauer, Christian Haubelt, Christian Zebelein, and Jürgen Teich. "Integrated Modeling Using Finite State Machines and Dataflow Graphs". In: *Handbook of Signal Processing Systems*. Ed. by S.S. Bhattacharyya, E.F. Deprettere, R. Leupers, and J. Takala. Third Ed. Springer International Publishing, Oct. 2019, pp. 825–864. DOI: 10.1007/978-3-319-91734-4_23.

Abstract: In this chapter, different application modeling approaches based on the integration of finite state machines with dataflow models are reviewed. Many well-known Models of Computation (MoC) that are used in design methodologies to generate optimized hardware/software implementations from a model-based specification turn out to be special cases thereof. A particular focus is put on the analyzability of these models with respect to schedulability and the generation of efficient schedule imple-

mentations. Here, newest results on clustering methods for model refinement and schedule optimization by means of quasi-static scheduling are presented.

[10] Luise Müller, Kai Neubauer, and Christian Haubelt. "Exploiting Similarity in Evolutionary Product Design for Improved Design Space Exploration". In: *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. July 2021.

Abstract: The design of new products is often an evolutionary process, where product versions are build on one another. This form of product generation engineering (PGE) reuses some parts of previously developed systems, while others have to be designed from scratch. In consideration of subsequent design steps, i.e., verification, testing, and production, PGE may lead to significant reduction of the time-to-market as these steps can be skipped for reused parts. Thus, deciding which components have to be replaced or added to meet the updated requirements while preserving as many legacy components as possible is one of the key problems in PGE. A further aspect of PGE is the potentially more efficient search for valid design candidates. An already optimized base system can be systematically extended by new functionality without the necessity to search the entire design space. To this end, in this work, we propose a systematic approach, based on Answer Set Programming, to exploit the ideas of PGE in electronic system-level design space ex- ploration. The idea is to gather information of the implementation of a previous design, analyze the changes to a new version, and utilize the information to steer the search towards potentially good regions in the design space. Extensive experiments show that the presented approach is capable of finding near-optimal design points up to 1,000 times faster than a conventional approach from scratch.

1 Introduction

Throughout the last decades, embedded systems have penetrated the whole spectrum of products in the market. They are nowadays ubiquitous and form with about 98 % [11, 12] the largest share of all microprocessor systems. Beside typical areas such as telecommunication, embedded systems are also utilized in traditionally mechanical sectors. Prominent examples include automobiles, airplanes, factory automation, clinical devices, and domestic appliances.

Embedded systems are characterized by a tight integration into a larger system where they fulfill a specific task and interact with the environment through sensors and actuators. They have to adhere to a number of constraints which can be roughly categorized into behavioral, functional, and extra-functional requirements. Behavioral constraints demand that the system behaves under all circumstances the same as specified for the desired task. On the example of a network router, the forwarding of a message to the correct destination is considered a behavioral constraint. Functional and extra-functional requirements, on the other hand, both refer to the properties of the system. While the former ones are imperative for error-free operation, the latter are considered quality constraints that are additionally imposed by the specification. Liveliness, e.g., an arrived message will eventually be forwarded by the aforementioned router, is a typical functional requirement while performance, area costs and power dissipation are typically considered to be extra-functional. However, the distinction between functional and extra-functional requirements is not always a clear cut but rather dependent on the specific case. In airbag controllers, for instance, the latency from sensing a collision to triggering the airbag must not exceed a few milliseconds to guarantee the desired functionality, i.e., save lives.

Additional to the compliance to the various constraints given by the specification, the time-to-market is an important factor in current and future designs of embedded systems. Due to worldwide competition, companies are forced to enhance design and decision processes in order to beat their competitors to new or optimized features. Simultaneously, more and more features with challenging requirements, both functional and extra-functional, increase the complexity of embedded systems constantly. Combined with the pressure on a short time-to-market, manual design is not viable anymore as the number of design options is too large.

Thus, automated approaches that are able to explore the vast search space and evaluate potential design candidates according to specified design constraints are needed for an efficient design. An important prerequisite for automated approaches is a formal description of the system that abstracts the properties and interaction of components of the system mathematically. This model-based design can be employed at various abstraction levels, i.e., depending on the required level of detail, properties of components are more or less precisely modelled. To date, most approaches explore the design space at high abstraction levels and only further refine potential candidates in lower abstraction levels. In the monograph at hand, automated approaches are investigated at the system-level. At the system-level, architectural decisions are

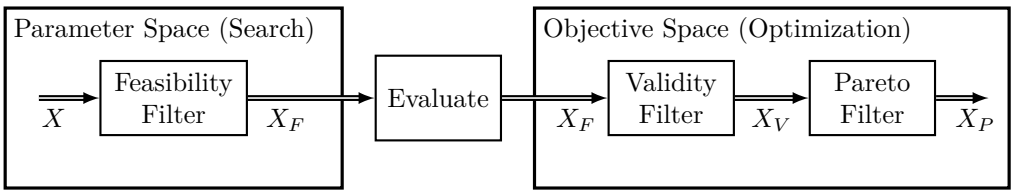


Figure 1.1: Design Space Exploration as a filtering process (inspired by [13])

made on the basis of task-level algorithmic descriptions of the system. Architectural decisions include the allocation of necessary hardware resources such as microprocessors, FPGAs, memories, and communication devices. Tasks are partitioned into hardware and software by binding them to the corresponding resources. The scheduling step finally determines the execution order of the tasks.

The results of the synthesis, i.e., all allocation, binding, and scheduling decisions that have been made, are considered as a design candidate that, subsequently, has to be evaluated. The entire process can be abstracted as a filtering process as depicted in Figure 1.1. The set of all potential design candidates X is first filtered by the feasibility filter. It guarantees that no designs with infeasible decision combinations are selected. This excludes, for example, solutions with unmapped or multiple mapped tasks, a broken communication infrastructure, and erroneous message routing. The set of feasible designs X_F is then evaluated with respect to specific objective functions. That is, the evaluation transforms the design from the parameter space into the objective space. In the objective space, the designs can subsequently be filtered by the validity filter resulting in the set X_V. A design candidate is considered valid if all specified constraints in the objective space are fulfilled. Beside the adherence to specified constraints, the desired result of a design space exploration (DSE) is often the optimization of its properties. In a typical embedded system, the optimization objectives are manifold and normally conflict with each other. As a result, there exists no single optimal design that dominates (i.e., evaluates better for all objectives) every other design. Instead, a set of Pareto optimal, mutually non-dominated design points (Pareto front X_P) is obtained that renders the best compromise solutions to a given problem.

In summary, the challenges for an efficient design space exploration can be split into the modeling of the system, the evaluation of one design point, the coverage of the design space, and the optimization with respect to specified objectives:

Modeling: A formal model of the system is imperative for an automated design space exploration. It must be able to include all properties and constraints that are imposed by the specification. In this work, the first-order language of ASP is utilized to model both the specification and the rules that are implied by constraints and objectives. A synthesis problem is therefore split into two parts. First, the specific problem instance defines the properties of the individual parts of the system such as the existing task dependencies, hardware resources, and mapping options. Second, a general encoding defines rules for synthesizing the specification into an implementation. This leads to the advantage that the encoding can be utilized for every possible problem instance and does not have to be generated anew.

Evaluation of Design Points: The computational complexity of the evaluation step highly depends, among other factors, on the desired accuracy and the specific objective that has to be evaluated. At system-level, the primary goal is a fast exploration of the design space resulting

in a set of potential designs that are investigated further at lower abstraction levels (e.g., gate and transistor-level on the hardware side). While this indicates that the accuracy is of lower concern, excessively increasing the evaluation performance at the expense of accuracy may introduce non-optimal designs. Thus, a good evaluation constitutes a compromise to allow for good accuracy while maintaining high performance.

Covering the Design Space: Although the decreased level of detail at the system-level narrows the number of design decision significantly, the complexity of medium to large systems restricts the exhaustive exploration of the entire design space even at high abstraction levels. Hence, finding promising regions of the design space that contain feasible, valid, and ultimately optimal design candidates proves to be hard. Traditionally, population-based heuristics were employed to search for these regions. They start with an initial randomized population of individual design decisions whose parameters are subsequently combined and altered in order to create new design candidates. In general, this produces diverse and good solutions. There are, however, two main disadvantages with these strategies. First, as the initial population is created by a randomized process, finding regions with valid design points becomes a problem for stringently constrained environments. Second, because the search is generally not executed systematically but based on combining previously found solutions, heuristics tend to run into saturation and stop finding novel solutions after an arbitrary number of iterations. Thus, giving a guarantee that the whole design space will be explored is nearly impossible even when they are run indefinitely. Therefore, a formal approach is examined in this thesis that explores the design space systematically. With solving techniques derived from Boolean satisfiability problem (SAT) and satisfiability modulo theories (SMT) solvers, finding regions with potentially good design points is accelerated and the exploration coverage is, at least in theory, guaranteed to be complete. Especially if the constraints are too stringent, the problem can be proven to be unsatisfiable.

Optimization: Obtaining the set of Pareto-optimal designs points implies comparing each valid design candidate with every other design candidate. This can be done by first saving all valid designs with subsequent filtering of non-dominated ones or by only saving non-dominated solutions in the first place. For medium to large systems, the former method is not viable, as there might be billions of valid designs that only differ in marginal details. The latter, on the other hand, demands continuous comparisons of new design points with previously found ones which could influence the exploration speed negatively if the underlying archive is not managed efficiently. Independent of the specific method used, the optimization approach has to guarantee for a correct and complete result even if not the entire search space could have been explored. That is, with respect to all explored designs, the Pareto-set must not contain any dominated solutions (correctness) and has to include all non-dominated ones (completeness).

1.1 Contributions and Limitations

In this section, the main contributions and limitations of the thesis at hand within the individual areas described above will be collated. In summary, the result is a novel methodology for designing embedded systems at the system-level that parts from traditional heuristics by proposing an automated DSE that is based on ASP.

Optimization The main contribution of the thesis at hand is the successful merger of the two, previously isolated problems searching for Pareto-optimal design points and defining a feasibility preserving encoding. This has been accomplished by the tight integration of background theories into the solving process of the foreground logic solver under the programming paradigm of ASPmT. This way, the Boolean ASP solver automatically ensures feasibility constraints while non-linear objectives and the optimization are executed by specialized propagators. The main benefit of the proposed approach is the shared utilization of decision variables by both foreground and background solvers. That is, compared to separated or only loosely coupled solvers, conflicting decisions as well as dominated design candidates detected by the background propagators can be used to directly steer the search and prune invalid and dominated designs from the search space. Furthermore, the logical connection of both steps permits a succinct encoding. Instead of separated input languages, the thesis at hand proposes a uniform formulation of the entire DSE problem. Hence, feasibility and validity constraints can be encoded side by side and, more importantly, share information on decisions imposed by other constraints. Finally, as variables are shared in the foreground and background solvers, the repeated exploration of the same design points is actively prevented. Due to the strict systematic search, decisions that are shown to be infeasible or invalid by the ASP engine and background propagators, respectively, are definitively removed from search. While this can also be achieved with disjoint solvers, the shared variables provide the opportunity for the specialized propagators to obtain minimal reasons. Hence, larger regions of the design space can be pruned compared to individual solutions that are evaluated externally without shared variables.

The main limitation of the proposed optimization approach stems from the complexity of the considered systems. As the system synthesis is proven to be NP-complete, the time required to solve an instance grows exponentially with its size. Although the approach guarantees finding each solution theoretically, in reality, a complete exploration of the search space is generally not viable in a reasonable amount of time.

Evaluation In order to obtain valid solutions to a specific synthesis problem, an evaluation has to be performed for every explored design candidate. Typically, a design candidate can only be evaluated for validity if a complete assignment is available. In this thesis, a novel approach allowing the evaluation of partial assignments is proposed. Enabled by the tight integration of the search and optimization through the ASPmT paradigm, as described above, design candidates can be checked early for constraint violations. That is, as the variables are shared between foreground and background theory, whenever a Boolean decision is taken, the specialized propagators can check whether the solution can still be satisfied. This allows for the detection of invalid regions early during the search and, thus, the pruning of entire invalid and dominated regions of the design space. Experimental results (q.v., Section 3.4.2) show the superiority of the proposed approach. While with an evaluation of full assignments only, less than 25 % of all problem instances have been solved, the proposed approach utilizing partial assignment checking has yielded a success rate of more than 90 %.

As partial assignment checking imposes a larger number of evaluations to be carried out, the evaluation is identified as a bottleneck of the DSE. Therefore, the thesis at hand proposes the utilization of approximations for the majority of evaluations before exact evaluations are only executed for promising candidates. The proposed approach builds upon the use of safe approximation. As such, compared to previous work, the correctness and completeness of the

obtained non-dominated solutions can be guaranteed although only a fraction of all designs is evaluated exactly. The approach is experimentally evaluated and is shown to accelerate the DSE by up to one magnitude for the considered instances and objectives.

Similar to the evaluation, the number of necessary dominance checks also increases with the partial assignment checking. To tackle this problem, the thesis proposes an archive management technique that can be used to relay expensive operations to when the solution is completely decided. The remaining checks conducted for the partial assignments require significantly reduced dominance checks while retaining completeness and correctness. Compared to a traditional management structure, the proposed approach shows an improvement of both filtering time and the number of necessary comparisons by about one and two orders of magnitude, respectively.

The evaluation is executed on the basis of abstract properties that are annotated within the specification model. In reality, these properties have to be obtained through thorough investigation. The execution time of a task on different resources, for instance, can be obtained by compiling the task for different architectures. While realistic assumptions are aspired, the data source of these properties is not the primary scope in this thesis. It is assumed that the properties are known when starting the design space exploration. A second limitation is the restriction of partial assignment checking for assignment monotonic objective functions. That is, with every additional decision, the evaluation result must constantly move into the same direction for one objective. All objective functions used here are assignment monotonic. The utilization of safe approximations is, in addition to the requirement of assignment monotony, restricted to objectives that can be consistently approximated. If no such approximation can be determined, the DSE is equivalent to a traditional approach that has to exactly evaluate all design candidates.

Guided Decoding The design space of realistically sized problem instances is essentially constituted of vast numbers of mapping, routing, and scheduling options. In general, feasible design candidates are located within narrow and separated regions of the design space. That is, many combinations of the individual decisions may either lead to infeasible or similar design candidates. The main goal of the DSE is therefore twofold. First, pruning infeasible regions from the search and second, finding disconnected regions to obtain diversified designs. In contrast to previous approaches, in the present work, the DSE is carried out strictly systematically. Hence, advancing from one solution to another is steered by the previously found solution, the encountered conflicts, and dominance checks. This is different to the mainly heuristic approaches advancing entirely or at least partially randomly through the design space. The proposed approach has the advantages that no design candidate is explored twice and that every design point is eventually found in finite time. Furthermore, invalid regions, if detected, are not considered for further exploration as they are automatically pruned from the search.

At the downside, the strictly systematic search may prevent finding diverse solutions fast. The strict systematic Boolean search tends to vary subsequent solutions only marginally as small changes in the individual decisions, equally yield small changes in the resulting solutions. Especially, complex problem instances contain large amounts of designs that are technically different but are characterized by similar properties. Thus, the escape from local optima during the search is generally less effective when compared to approaches where optimization and encoding is logically separated and applies random decision into the search.

Modeling The central contribution of the thesis at hand is the overall methodology. However, to evaluate the proposed methods regarding the DSE at the ESL, an appropriate specification model is imperative. This monograph does not develop a completely novel model but instead builds upon well-established, graph-based models at the ESL. Therefore, a model is chosen that permits the definition of task-level applications, a hardware platform template implementing a network on chip (NoC) and mapping options connecting the previous two. In the context of evaluation, a methodology is proposed to systematically generate problem instances to test the proposed approaches with defined problem classes.

The selection of the specification model implies some limitations on its own. First, the applications are statically defined. That is, no dynamic information such as data-dependent behavior, interrupts or sporadicity is supported. That is, all information that is specified in the model has to be known at design time. This includes the availability of hardware resources, task dependencies, and mapping options. Furthermore, no automatic dynamic reconfiguration is possible if unforeseen events such as resource failure happen at runtime of the system. In this case, a new synthesis had to be carried out.

1.2 Thesis Overview

Most of the approaches proposed in this monograph have previously been published by the author at peer-reviewed conferences, workshops, and journals. This monograph aggregates the results of the respective works and puts them into the context of a holistic view on the DSE at the ESL. To this end, the formulation is unified, refined, and extended by more extensive experimental evaluations. In the following, the structure of the remainder of this monograph is outlined.

Prior to the main chapters, the fundamental concepts of model-based design are presented. Chapter 2 deals with all topics that are necessary to fully comprehend the subsequent chapters. The introduction of the general process of system design is followed by the necessary definitions regarding model-based constraint checking with logic solvers and the basics of multi-objective optimizations. The chapter concludes with an elaborate discussion of related work, where existing approaches are compared to the proposed solutions in the thesis at hand.

Chapter 3 addresses the synthesis at the ESL. It is based on the work, that was previously published in [2] and [5]. The system model is defined and the integration of background theories under the ASPmT paradigm is discussed. Orthogonal to the proposed ASPmT-based synthesis framework, a methodology for a modular test case generation is proposed. This is subsequently used to evaluate the developed framework.

The holistic DSE is elaborated in Chapter 4. It extends the proposed synthesis framework with multi-objective optimization and is primarily based on the work in [4]. After the evaluation of different approaches to the DSE problem regarding the obtained quality of the non-dominated front, improvements to the substeps dominance checking and evaluation are proposed. The former is based on the methods proposed in [3] and aims in reducing the complexity of the Pareto filter by utilizing specialized data structures. The latter has been previously published in [6] and [7]. Here, the use of safe approximations is proposed that aim at accelerating the evaluation of a design candidate while retaining the correctness of the obtained front.

The thesis concludes with Chapter 5, where the contributions and limitations of the proposed approaches are summarized and an outlook of future work is given.

1.3 Funding and Cooperation

The monograph at hand is part of the result of the cooperation between the chair of *Embedded Systems* at the University of Rostock and the chair of *Knowledge Processing and Information Systems* at the University of Potsdam. It was funded by the *German Science Foundation* (DFG) under the grants HA 4463/4-1 (Rostock) and SCHA 550/11-1 (Potsdam) with the title *Scalable Design Space Exploration with Answer Set Programming* (original title: *Skalierbare Entwurfsraumexploration mit Antwortmengenprogrammierung*) [14].

In this cooperation, the work was essentially split into the enhancement of the utilized ASP solver clingo and the application to the problem at hand contributed by the University of Potsdam and the design space exploration methodology contributed by the University of Rostock. The spent effort of the University of Rostock was mainly accomplished by the author of this monograph. The work on the solver includes the development and integration of the ASPmT paradigm and the implementation of a dedicated integer difference logic solver. Therefore, the internal realization and theory of the ASP solving mechanisms have not been contributed by the author of this monograph but instead by the cooperating partners at the University of Potsdam. It shall be noted, however, that the basics of the implementation are also discussed in the monograph at hand (see Sections 3.2 and 3.3) as the systems will be used as a tool in further considerations.

While the ASPmT technology developed in context with this work has a more general application, the main focus of the work conducted by the author has been related to domain specific problems. This includes both the definition and encoding of necessary parameters of the specification as well as the necessary information that shall be considered for optimal design points (i.e., objective functions). In addition, non-ASP related optimizations to the design space exploration have been studied by the author of this monograph. In particular, this concerns the archive management technique based on Quad–Trees (see Section 4.2) and the proposal for the use of safe approximations (see Section 4.3) to accelerate the Pareto filtering and evaluation substeps, respectively. Furthermore, the conceptual design of the conducted experiments within the individual sections and chapters as well as the generation of test cases (see Section 3.4) have been realized by the author.

The synthesis and optimization architectures, proposed in Sections 3.3 and 4.1, respectively, are the result of the combined efforts of both project partners. They are build on both the developed internal solver mechanisms and the design space exploration methodology. Hence, these architectures provide the common result of the cooperation between the University of Potsdam and the University of Rostock in this project.

2
Model-based Design

The work at hand cannot be considered in isolation but is instead embedded in a much wider context in the area of design space exploration at the electronic system level (ESL). It is based on well established concepts that are partly combined and extended particularly towards exact design exploration methodologies. To motivate these general concepts, the goal of the present chapter is the introduction of model-based design techniques and theory solvers.

Therefore, first, the typical design flow of digital systems is described, detailing design steps at different levels of abstraction from system level to logic and instruction levels. Here, it will be shown that commencing at system level benefits the entire design process regarding exploration speed and coverage. In the second part of the chapter, well established concepts regarding multi-objective optimization that are utilized throughout this work are outlined. Within this section, exact solving approaches are motivated and expounded why they are especially well suited for highly constraint design problems. Finally, an extensive investigation of related work is presented at the end of this chapter where main contributions of the work at hand are elaborated and set into context.

2.1 Hardware/Software Co-design

The design of application-specific embedded computer systems is an interwoven process of simultaneous hardware and software development. The development of such systems has a tradition of over 70 years now. Starting with the first universally programmable computers Z1 (1938) to Z4 (1945) [15], through computer-aided space exploration to current systems in everyday life, the complexity has been risen constantly. Today, billions of nanoscaled interconnected devices are integrated in only a few square centimeters that run software that has to fulfill strict performance and power constraints. Such complexity necessitates highly automated approaches.

However, even with modern high-performance computers and algorithms, the vast design space is often too large to optimize every least detail of the system. Thus, a system is typically grouped into larger building blocks that abstract various details and are used to minimize the overall design complexity. For example, individual transistors are grouped into logic gates, logic gates into functional units, and finally functional units into processing and communication resources. At software side, individual instructions are clustered into basic blocks, functions, and finally abstract tasks. As a result, the design space of a digital system defined with the largest building blocks is smaller and can be explored faster. The downside is that many details cannot be expressed at the highest abstraction levels. Hence, current design methodologies utilize different abstraction levels and consequently refine the resulting system gradually throughout the design process.

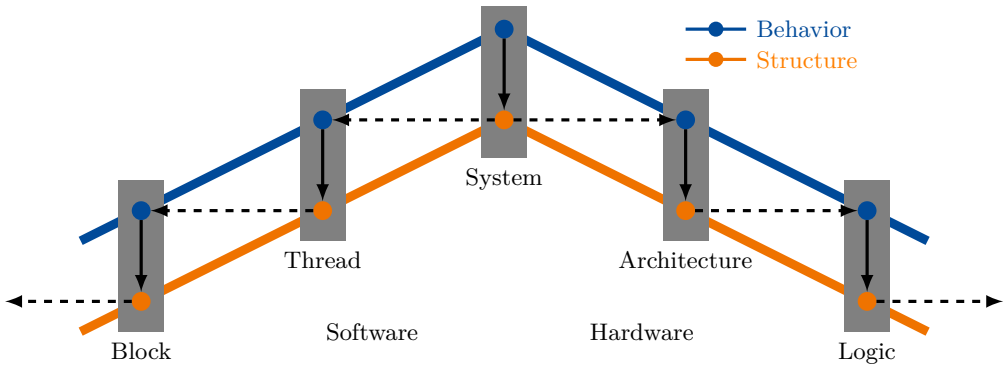


Figure 2.1: Double roof model of systems design (adapted from [16]).

2.1.1 Design Process

The employment of different abstraction levels in the design process is typically a top-down process. The double roof model of design [16], as depicted in Figure 2.1, is a concrete methodology implementing such a process. It is divided into a hardware and a software side with both meeting at the center, where the system level is located. Hardware and software are in turn divided into further abstraction levels, that involve the respecting building blocks as described above. Within each abstraction level, the goal is the transformation from a behavioral specification into a structural implementation which involves decision-making and optimization. These steps are indicated by the vertical arrows in Figure 2.1. The horizontal arrows, on the other hand, represent the refinement steps from one abstraction level to the next. Here, the structural implementation as the result of a higher abstraction level is (partially) transformed into a behavioral description and used as input at a lower abstraction level. This way, the design process gets gradually more detailed until the system description is detailed enough to manufacture the hardware and execute the software as machine code on the corresponding computational resources.

At system level, the fundamental design decisions are made. The decision-making process includes the allocation of available computational and communication resources, the partitioning of functionality into hardware and software, and the task scheduling, i.e., the order of task executions and identification of serial and parallel structures. The structual implementation reflects the taken design decisions and can further be represented as transaction-level system simulation models or as as graph-based models. Subsequently, the result of the design is then split into hardware and software, and relayed to the subsequent thread and architecture abstraction levels, respectively. At thread level on the software side, communicating processes/tasks have to be bound to available processors and scheduled according to the underlying operating systems. The software synthesis at the thread level typically results in source code, structured into classes, functions, and basic blocks. It is subsequently used for compilation in the final step at the software side. At block level, the previously generated source code is compiled and linked and results in machine-readable code for the respective processors. On the hardware side, first the computation and communication is implemented at the architecture level. That is, tasks and desired capabilities of the system that have been decided to be implemented as hardware accelerators are elaborated. Mainly, the selection and interconnection of various memories, arithmetic logical units (ALUs), and communication infrastructures is subject at

this level of abstraction. This synthesis step can also be automated, which is commonly known as high level synthesis. The resulting register transfer level (RTL) description is often realzied in form of hardware description languages such as VHDL [17] or Verilog [18]. At the logic level, the lowest abstraction on the hardware side, the register transfer level (RTL) description is transformed into logic gates like flip-flops, inverters, and Boolean gates. Admittedly, the hardware synthesis continues even further at transistor level (not shown in Figure 2.1). Here, the Boolean logic is compiled into individual transistors, wires, and their spacial placement necessary for the final physical implementation on the die.

Beginning the design process at the highest level of abstraction, the system level, is largely motivated by two considerations. First, the high abstraction at system level conceals details of specific parts of the digital system. Hence, the focus is shifted towards more fundamental decisions which can be explored and evaluated more quickly which, in turn, increases the exploration coverage. That is, decisions made at system level have potentially the largest impact on the final design as they define the golden reference of all subsequent steps. Second, while in traditional approaches, the firmware and software development is dependent on the hardware design process, both design steps can be started concurrently on the basis of the system-level implementation (e.g., represented by a virtual prototype). Although the overhead of obtaining this implementation may be significant, it can reduce the overall development time. Moreover, as first evaluations of the system properties are already applied here, potential flaws in the specification resulting in over- or underdesigned systems can be detected. This helps to prevent late errors that are, in general, harder to fix and result in longer turn around times.

Note that a strict top-down approach is not always possible nor desirable in real world designs of digital systems. A company may use subsystems available from previous projects or may choose to obtain parts of the system as 3rd-party intellectual property (IP). These parts do not have to be designed again and can directly be used in new systems resulting in a hybrid approach. For example, a processor is typically not designed from scratch, but instead bought as an off-the-shelf subsystem with predefined properties and interfaces. On the other hand, design-specific functionalities such as sensor subsystems might not be available through 3rd-party IP and have to be designed in a top-down process. This hybrid strategy is, however, not in conflict with the general approach. Moreover, the known properties of existing parts can be utilized to simultaneously increase the evaluation accuracy in high abstraction levels and decrease the time-to-market of the digital system.

2.1.2 Modeling Approaches

In order to allow a holistic specification and an automated design of digital systems, a sophisticated design methodology is needed. In principle, there are two different ways to specify digital systems. First, language-based approaches and, second, model-based approaches [16]. Language-based approaches are, generally speaking, dedicated to specific abstraction layers of the design process. The hardware design is mostly exclusively executed with the hardware description languages (HDL) Verilog and VHDL as they provide solutions from the architectural view to the logic level of the system. Typical HDL often do not support the same tool-supported development processes as modern high-level languages in software development. As a result, development of complex software in VHDL or Verilog is often error-prone and time-consuming. Therefore, they are less suitable for the software design process. Here, high-level languages such as C, C++, and Java (among others) are more prominently used. Yet, without modifications,

they lack the concept of concurrency and do not cover timing behavior and are therefore not adequate for the description of hardware. System-level design languages (SLDLs) (e.g., SpecC and SystemC) aim at a combined specification of both hard- and software. SystemC [19], for example, is an extension library to C++ and contains structures for specifying concurrency and timing constraints while maintaining compatibility to standard C++. An event-based simulator can be used to evaluate the design for correctness regarding behavior and timing constraints. Furthermore, there exist high-level compilers that allow to translate a subset of SystemC into RTL descriptions and synthesize it into logic. While the language-based design is highly expressive, i.e., various behaviors can be specified with a limited set of basic elements, it generally lacks of clearly defined formal semantics. Both, the high expressiveness and the absence of formal semantics often deteriorate the analyzability of extra-functional properties such as timing, cost, or energy consumption of the specified system. In turn, this disqualifies language-based approaches for automated exploration, especially at high levels of abstraction.

In contrast, model-based approaches offer strict formal semantics. They are abstracted through mathematical models that reduce the expressiveness and, thus, allow for an improved analyzability. Typical approaches for modeling systems include finite state machines (FSMs), Petri nets, timed automata, and dataflow models. In the area of transformative embedded system design, dataflow models are often used as parallelism of the underlying application can be easily identified [20]. Dataflow models consist of individual actors that communicate over channels. The actors model the behavior of processes and consume and produce data (i.e., tokens) from and to connected channels, respectively. This way, especially streaming applications can be naturally modeled. Besides the modeling of the functional behavior of an application, the formal semantics of synchronous dataflow models[2-1] also allow for the evaluation of extra-functional properties. Especially, the buffer size (for FIFO channels) and the maximum throughput analysis are well studied throughout literature (e.g., [21–23]). Although hierarchical dataflow models can be used to represent mapping and scheduling decisions (e.g., [20]), dataflow typically only models the behavior of an application.

Again, the hardware is typically abstracted and is reduced to the properties that are needed to evaluate the desired properties such as individual costs, power consumption, and processing abilities. While both hard- and software models can be extended with respect to their expressiveness, especially at ESL, this hinders automated design space exploration (DSE) as evaluation performance and analyzability will deteriorate with increasing model complexity.

2.1.3 Synthesis

Independent of the specific abstraction level, a specification is transformed into an implementation. The overall goal is not only to identify valid designs that comply to given requirements and behavioral properties but also to search for optimal ones among them. In the end, the transformation results in a structural description and specific quality characteristics of one or potentially a set of (Pareto) optimal design points. This step is known as synthesis and is detailed in the following. Note that the synthesis is treated here in the context of the system level abstraction. The process, however, is similar for lower levels of abstraction although the degree of detail and the focus on specific steps might differ, depending on the abstraction level.

[2-1] Note that the analyzability deteriorates with increasing expressiveness of the dataflow model. For example, Boolean and other dynamic dataflow models are already Touring-complete and only allow limited evaluation of extra-functional properties.

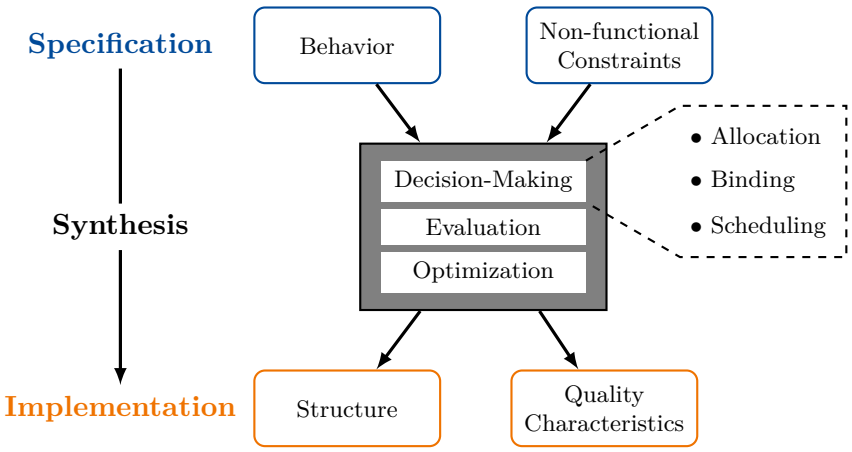


Figure 2.2: X-Chart model of synthesis based on [24]

As depicted by the X-Chart diagram [24] in Figure 2.2, three intermediate steps are necessary to transform a specification into an implementation. That is, decision-making, evaluation, and optimization. These steps generally cannot be considered isolated from each other, but instead the results build up mutual dependencies. For instance, the evaluation of an invalid design candidate (i.e., the requirements are not met for the made decisions) can be analyzed and return the reason for the invalidity to the decision-making in order to steer it.

When specifying a digital system, it is normally unclear which components are used to compose the final design, where the functionality is implemented, and in which order tasks are processed. These steps are subject to the decision-making process which is further structured into allocation, binding, and scheduling. During allocation, resources are selected that are used as a hardware platform to execute the functionality of the digital system. At system level, such hardware platforms consist of computing resources (CPUs, DSPs, and GPUs), communication components (busses and/or networks on chip (NoCs)), memories, and other hardware IP blocks that add specialized functionality. In the binding step, the specified tasks are mapped onto the previously allocated computational resources. Data and messages that have to be exchanged between tasks are routed over the communication infrastructure and (intermediate) results are assigned to memories. Allocation and binding are tightly coupled. Allocated computing resources have to allow the realization of the desired tasks and the allocated communication infrastructure has to allow for the necessary links between dependent tasks mapped to different resources. Finally, scheduling determines the execution order of tasks and communication steps. Therefore, parallel and serial structures are identified and resource sharing of tasks and messages is organized. Note that the determination of a static schedule is not always possible. If, for example, the task execution is conditional on dynamic input data not known at design time, the exact schedule must be determined at runtime of the system. In these cases, scheduling policies for every resource involved must be defined which can include priority assignments, partial execution orders of tasks, and deadlock avoidance strategies. The treatment of dynamic schedules, however, is out of scope of the present monograph. It is assumed that a static schedule can be obtained for the considered systems. This is enabled through the assignment of worst case execution times for tasks that encompass all possible execution traces for the corresponding tasks.

The evaluation of a design candidate has the principal goal of determining the quality of the design. Based on the decisions made during allocation, binding, and scheduling, the evaluation step assesses the properties of the system with respect to the requirements defined in the specification. The evaluation can be conducted based on simulations of the system, by means of mathematical models, or a combination of both. Simulations are typically more accurate than mathematical models, especially when the digital system has a high amount of dynamic behavior. On the other hand, mathematical models are typical much faster than simulations which, in turn, favors the coverage of the search space. Apart from that, mathematical models can be tuned towards the assessment of partial designs (i.e., only a subset of decisions has been made) helping to indicate infeasible designs earlier. This is discussed in more detail in Section 3.2.2. Independent of the specific evaluation approach, it provides the quality characteristics of the design candidate which are utilized by the final optimization step.

Apart from a feasible design, it is often desired to design the digital system as good as possible. In order to outperform the competitors, it might not be sufficient to comply to a given power budget but instead to deliver a system that consumes less power while still adhering to the remaining requirements. This can, for example, be achieved by selecting fewer or slower resources and instead put the emphasis on a better utilization through resource sharing. Thus, the goal of the optimization is to analyze the design decisions and to steer the search towards potentially better regions in the design space. This illustrates again the importance to commence the design process at system level. Design decisions at lower abstraction levels have less impact on the quality characteristics of the entire system. For instance, removing a whole CPU from the hardware platform at system level has a higher impact on the power consumption than a waiver of a few gates at logic level.

2.2 Constraint Modeling and Checking

To allow for an efficient synthesis of electronic systems, both functional and non-functional constraints must be automatically provable for each design candidate. Therefore, the constraints are typically encoded by mathematical formulations. In this section, two encoding techniques are detailed that allow a combined and automated decision-making and constraint checking. First, the Boolean satisfiability problem (SAT) is based on propositional formulas and is used in numerous applications in the area of electronic design automation (EDA), verification, and testing. Second, answer set programming (ASP) stems from the area of knowledge representation and reasoning and is based on a first-order language. The latter is especially well suited for encoding reachability constraints that are required for the routing substep. As a specific encoding technique is in general only tailored towards one class of problems, the section closes with the concept of background theory solving. This methodology allows combining multiple specialized techniques under one general approach.

2.2.1 Boolean Satisfiability

Propositional logic is a common approach to encode decision problems and their corresponding constraints. It consists of binary variables (i.e., atoms) that are connected through logical connectives such as negations, conjunctions, and disjunctions into propositional formulas. The Boolean satisfiability problem (SAT) addresses whether a given propositional formula is satisfiable, i.e., if there exists a variable assignment such that the evaluation of the whole formula

```
1: function DPLL(M = {C_1, ..., C_n})
2:     if M = ∅ then return TRUE                          ▷ Set of clauses is satisfiable
3:     else if ∅ ∈ M then return FALSE                    ▷ Set of clauses is unsatisfiable
4:     else if {L} ∈ M then return DPLL(M | L)            ▷ Propagate unit clause
5:     else
6:         choose X : X ∈ ⋃C_i | i ∈ [1, n]
7:         if DPLL(M | X) == TRUE then return TRUE        ▷ Branching
8:         else return DPLL(M | ¬X)                       ▷ Backtracking
```

Figure 2.3: The basic DPLL algorithm without the deduction from pure literals [26, 27].

results to `TRUE`. In 1971 the authors of [25] proved SAT to be the first NP-complete problem, i.e., it is both in NP and NP-hard. Hence, on a non-determinstic machine, SAT is solvable in polynomial time (NP) and every other problem in NP can be reduced to SAT in polynomial time (NP-hard). So far, it is unknown whether there exists a general algorithm of polynomial complexity to solve SAT (and other NP-complete problems) on a deterministic machine. However, based on the Davis-Putnam-Logemann-Loveland algorithm (DPLL) [26, 27], today, various systematic approaches exist that can identify the satisfiability of many propositional formulas faster than the complexity class suggests.

The basic idea of DPLL is that the search space is pruned from infeasible regions and structural properties of the formula are used to propagate inevitable decisions. The original algorithm is given in Figure 2.3. It necessitates the propositional formula (viz. M in Figure 2.3) to be in conjunctive normal form (CNF)[2-2], i.e., a conjunction of clauses. A clause is a disjunction of literals and a literal is either an atom or its negation. Basically, the algorithm consists of three phases. During the deduction phase, the clauses are scanned for unit clauses and pure literals[2-3] to propagate inevitable decisions. A clause is considered to be a unit clause if it consists of exactly one literal. If a unit clause is identified, the decision variable can be assigned to the truth value according to its parity. A variable that appears only in one parity throughout the entire formula is called a pure literal. Again, the corresponding variable can be assigned the truth value of its parity. For example, consider the formula $g = (\neg p) \wedge (q \vee r) \wedge (q \vee \neg r)$. The unit clause $\neg p$ implies that p must not be set to `TRUE` as this would render the entire formula `FALSE`. Furthermore, the variable q appears only without a negation and can be safely removed from the formula, i.e., set to `TRUE`. Line 4 in Figure 2.3 executes the deduction from unit clauses. Whenever a unit clause of the form $\{L\}$ is detected, the algorithm is called recursively with the literal L removed from M, i.e., $DPLL(M \mid L)$.

In the second phase of the algorithm, the truth value of an unassigned variable is decided (lines 5–8). This step signifies a choice and creates two branches — one for the `TRUE` and one for the `FALSE` valuation (line 7 and 8, respectively). Whenever a `TRUE` valuation is assigned to a variable φ, all literals $\neg\varphi$ and all clauses containing φ are removed from the formula. Analogously, literals φ and clauses containing $\neg\varphi$ are removed if `FALSE` is assigned. If the following deduction phase leads to a conflict, characterized by an empty clause, the third phase is triggered. Here, the decisions and deductions are backtracked and reversed until an alternate choice in a previous branch can be made. The formula is satisfiable when no clauses are left in the formula (line 2). In contrast, it is proven to be unsatisfiable if no alternate decisions can

[2-2] There exist polynomial algorithms that allow the transformation of any propositional formula into a CNF (e.g., [28]) such that this requirement does not increase the complexity.

[2-3] In later approaches, pure literals are not utilized anymore as their identification is too complex and benefits are too low [27].

be identified in the backtracking phase. State-of-the-art SAT solvers (e.g., Chaff [29], MiniSAT [30]) still use the basic idea of DPLL. However, the backtracking step has been replaced by conflict-driven clause learning (CDCL) and backjumping [31]. Here, whenever a conflict arises, the reason for it is analyzed and a corresponding conflict clause is added to the original problem. Decisions are then revoked to the level when the first variable of the conflict clause has been decided. This prevents the solver to revisit the decision combination and, thus, prunes the search space more effectively.

Although DPLL can solve many SAT problems effectively in practice, in general, the algorithm is exponential in time with respect to the number of decision variables. The performance is especially dependent on the specific problem structure and the methodology used for assigning the next undecided variable. The latter can be influenced by various branching heuristics [32]. Greedy heuristics such as maximum occurrences of minimum sized clauses and Bohm's heuristic try to estimate the impact of the unassigned variable on the Boolean formula. They select the variable that satisfies most clauses or generates the most unit clauses. As the estimation is based on statistics, greedy heuristics work well for random SAT instances but cannot capture structural properties of the formulas [32]. Another approach consists of literal count heuristics. Before branching, for each variable, the amount of literals is counted in all currently unsatisfied clauses and the most frequent variable is selected for branching. The problem with this approach is the computational overhead introduced before each branching step. The authors of [29] have introduced the variable state independent decaying sum (VSIDS) heuristic. At the beginning, it assigns a score to each literal according to its number of occurrences in the formula. In each iteration, the highest valued literal is selected for branching. In regular intervals, all scores are adapted by dividing them by a constant number. However, whenever a conflict clause is added to the problem, the scores of containing literals are increased. This way, more recent decisions have a higher impact on the branching strategy than earlier ones. The computational overhead of VSIDS is much lower than those of literal count heuristics. Similar to VSIDS, the heuristic used in the BerkMin SAT sovler [33] is also based on decaying scores. However, in BerkMin, not only the literals of the conflict clause are considered for increasing their scores but each literal that leads to a conflict. While this makes BerkMin more robust, the overall performances of VSIDS and BerkMin are comparable.

Independent of the specific approach, SAT solving has become an important framework in many areas of EDA. In particular, the logic synthesis of integrated circuits and model checking techniques [31] in verification rely on SAT solvers. The application of SAT at the ESL has been introduced in [34]. The work presented a Boolean formula to encode feasibility constraints of mapping decisions throughout the synthesis. While decision constraints such as the mapping problem are well expressible in CNF, reachability (as required by the routing) is hard to encode effectively. Hence, the automatic checking of routing constraints is cumbersome with SAT and requires a large overhead. Therefore, another encoding approach is needed.

2.2.2 Answer Set Programming

Answer set programming (ASP) is a declarative programming paradigm that, in contrast to imperative programming languages, focuses on describing what the problem is instead of how to solve it. It is tailored towards NP-hard combinatorial search problems and is based on the stable model semantics, introduced by Gelfond et al. [35] in the late 1980s [36]. The term ASP has been first used one decade later in the late 1990s by [37]. In ASP, search problems

are reduced to computing stable models (i.e., answer sets). A stable model is a set of truth assignments that satisfy the problem and can be derived from given information. Although most ASP solvers utilize variants of DPLL, ASP and SAT mainly differ in two aspects. In contrast to SAT, reasoning in ASP is done under the closed-world assumption. That is, an atom is considered to be assigned `FALSE` whenever there is no evidence that it is `TRUE`. This makes ASP especially powerful for problems, where only a small subset of decision variables has to be selected whereas the remaining ones are not relevant. In particular, the closed-world assumption allows a natural encoding of reachability, and hence, routing constraints.

The second aspect regards the problem encoding. In contrast to SAT, ASP utilizes a first-order input language. The language consists of Prolog-style [38] rules that include predicate symbols, constants, variables[2-4] and logical connectives but must not contain quantifiers [39]. Knowledge is encoded as n-ary predicates (i.e., atoms) that consist of a predicate name and n arguments. While a constant represents itself, variables act as placeholders for each variable-free term in the logic program. Similar to SAT, a literal is an atom or its (default) negation. Each rule is composed of a head and a body in the form

$$A_0 :\text{-}\, L_1, \ldots, L_n.$$

and states that the atom A_0 in the head is inferred if all literals $L_1, \ldots, L_n$ in the body hold. Facts and integrity constraints are specialized rules that have empty bodies and heads, respectively. Facts of the form

$$A_0.$$

encode the knowledge of the system indicating that the atom A_0 is unconditionally `TRUE`. Integrity constraints are rules that filter solution candidates as they require that the literals $L_1, \ldots, L_n$ in the body must not be concurrently satisfied:

$$:\text{-}\, L_1, \ldots, L_n.$$

Aggregate atoms are common extensions to the stable model semantics in logic programs. They allow for expressing constraints over groups of literals. In detail, a function is applied to a set of literals and its result is compared with given values which then leverages a truth value.

In ASP, it is common to separate a problem into a specific instance and a uniform encoding that is applicable to every instance. The instance is composed of facts that represent the initial knowledge. In the area of ESL design, this conforms to the specification including the available computation and communication resources, tasks and messages, mapping options, and properties of the aforementioned elements. On the other hand, the encoding is composed of rules that are utilized to infer knowledge and generate answer sets for the given instance. The encoding therefore maps to the non-functional constraints enforced by the specification. Furthermore, the encoding of a search problem in ASP is typically characterized by a programming methodology that divides the logic program into three parts in which the different rule types are used [40]. In the "generate" part, potential solution candidates are typically defined using choice rules and aggregates. The "define" part contains definitions of auxiliary predicates. Finally, during "test", solution candidates that cannot be considered as models are removed from the search by integrity constraints that use predicates generated in the define part.

[2-4] Variables (i.e., atoms) in SAT do not map to variables but rather to predicates and their arguments in ASP.

Table 2.1: Encoding of the partitioning problem in SAT and ASP

SAT [34]	ASP

$m_{i,j} = \begin{cases} 1 & \text{if } t_i \text{ is bound to } p_j \\ 0 & \text{else} \end{cases}$
$b(m) = (m_{1,1} \vee m_{1,2}) \wedge (\neg m_{1,1} \vee \neg m_{1,2}) \wedge$
$(m_{2,1} \vee m_{2,2}) \wedge (\neg m_{2,1} \vee \neg m_{2,2}) \wedge$
$(m_{3,1} \vee m_{3,2}) \wedge (\neg m_{3,1} \vee \neg m_{3,2})$
	`task(t1). task(t2). task(t3).`
	`processor(p1). processor(p2).`
	`map(t1,p1). map(t1,p2). map(t2,p1).`
	`map(t2,p2). map(t3,p1). map(t3,p2).`
	`% GENERATE`
	`{bind(T,P)} :- map(T,P).`
	`% DEFINE`
	`bound(T) :- bind(T,P).`
	`% TEST`
	`:- bind(T,P1), bind(T,P2), P1 != P2.`
	`:- task(T), not bound(T).`

Together with the first-order language, this leads to a more readable representation when compared to SAT. Both functional and non-functional constraints can be specified more readable and the encoding can be reused and does not have to be generated for each instance of the same problem. As an example, assume the following partitioning problem. Three tasks t_1, t_2, t_3 shall be bound onto two processors p_1, p_2. Each task can be mapped onto both processors and must be bound exactly once. The different encodings for this problem are given in Table 2.1. In the SAT encoding, the problem is encoded with six atoms $m_{1,1}, \ldots, m_{3,2}$ representing the six possible mapping options. An atom $m_{i,j}$ is assigned true if the task t_i is mapped onto the processor p_j. The property that each task must be bound exactly once is encoded according to [34] in the CNF $b(m)$. Each line correlates with one task. The first disjunction demands that at least one mapping option is chosen and the latter prevents both from being chosen concurrently. Analogously, the ASP encoding is given in right column of Table 2.1. Note that in this and following examples, the syntax is aligned with the ASP solver clingo [41, 42]. It is separated into two parts as described above. The upper part consists of the specific problem instance and includes the knowledge about the three tasks, two processors and the six possible mapping options. The arguments of the predicates `task/1` (shorthand for: "a unary predicate with the name `task`"), `processor/1`, and `map/2` are constant symbols that represent the corresponding elements of the specification. In contrast to the problem instance, the encoding in the lower part utilizes variables as arguments. The variables are placeholders for the constants and are replaced during solving. The first line of the encoding consists of a choice rule rendering the "generate" part. Here, all potential solution candidates are defined as it states that a mapping option may or may not lead to a specific binding. The next rule depicts the "define" part. It defines a task T to be bound if any binding holds that includes that task T. Finally, the last two lines represent the "test" through integrity constraints. While the former prevents a task to be bound onto multiple processors, the latter eliminates answer sets where a task T is not bound to any processor. Althoguh the encoding nicely shows the methodology of "generate", "define", and "test", it can be significantly reduced by using aggregate atoms:

```
1 {bind(T,P) : map(T,P)} 1 :- task(T).
```

Thus, a single line is able to encode the uniqueness property specified in the problem. In short, the rule states, that for each task T, at least one and at most one (i.e., exactly one)

mapping must be chosen. Therefore, the head atom consists of an aggregate atom that generates `bind/2` predicates from the set of all `map/2` atoms containing the variable T. In turn, it evaluates to `TRUE` if exactly one `bind/2` atom holds.

The advantage of the ASP encoding is twofold. The specification is represented intuitively as elements can be expressed in a near-natural language. The encoding itself is more concise and can be reused for other instances of the same problem. While adding tasks or processors to the problem or changing the mapping options requires the CNF of the SAT encoding to be regenerated entirely, the ASP rule can remain unchanged.

Processing answer set programs with an ASP solver is executed in two phases. In the first phase, the logic program is translated into a semantically equivalent variable-free representation. This is called grounding. Principally, a naive approach where the variables are replaced with all possible constant combinations occuring in the logic program, can be employed. However, it would increase the number of rules exponentially. Assume the generate rule of the example above. Replacing the variables P and T with all combinations of the five available constants would result in 5^2 variable-free rules. Hence, modern grounding techniques employ more intelligent techniques to generate a grounded program. This includes techniques such as partial evaluation, rewriting, and the elimination of equivalences like tautology rules [39]. In the case above, for example, many combinations can be removed as only the constants of tasks and processors are eligible for T and P, respectively.

After a variable-free representation is generated, the actual ASP solver is started. All modern solvers are based on CDCL techniques as developed for SAT. However, they additionally extend the search with a foundedness constraint. That is, the atoms in an answer set must be derivable by a rule in the logic program. Similar to SAT solvers, they can be improved by heuristics that influence the assignment order and preferred phases.

2.2.3 Background Theory Solving

Pure combinational theories such as ASP and SAT are predestined for solving Boolean decision problems. That is, a subset of decision variables is selected according to linear constraints and rules. Often however, real constraints do not have linear dependencies. One example at hand is finding and optimizing a valid schedule for a selected binding of tasks. As resources are shared and tasks executions are dependent on each other, selecting a binding for a task might influence (i.e., increase) further task execution times or does not affect the overall timing at all (i.e., the task execution is fitted into a free time slot). In fact, the schedule is not only dependent on the binding of the tasks but also the order of executions and the communication routing over the network. In order to account for all these constraints, a Boolean encoding would need decision variables that represent every possible time slot for every task. As a consequnece, the number of these decision variables would increase exponentially. Hence, handling such constraints is not feasible for realistic problem sizes.

Satisfiability modulo Theories One possibility to handle the aforementioned problem is the application of background theories. The idea is that a part of the encoding is split from the original problem and solved using a specialized technique. The result is then fed back to the foreground theory (e.g., ASP, SAT) with the help of indicator variables that are known by both fore- and background theories. The methodology is originally known as satisfiability modulo theories (SMT) and was first applied to SAT. Here, in addition to propositional terms, SMT

formulas consist of multiple different (combined) theory terms. In general, there exist two possibilities to handle these combined formulas. The first approach is based on the translation of the background theory into SAT. Such eager approaches have been developed as efficient SAT solvers became available. The main advantages of an eager approach is that a state-of-the-art SAT solver can be used to check the resulting formula for satisfiability. Although there exist algorithms for translating various theories (e.g., the theory of equality of uninterpreted functions (EUF) or difference logic) into propositional logic, they are often not as efficient as specialized solvers [43]. Lazy SMT techniques are the second category and are used in many modern SMT solvers such as Z3 [44]. Here, each atom in the original SMT formula is considered a propositional symbol, i.e., not as a special theory term. The resulting propositional formula is then checked for satisfiability by a SAT solver. If the propositional formula is already unsatisfiable, the whole SMT formula is too. Otherwise, the propositional model is checked by a specialized theory solver. In turn, the theory solver returns a conflict clause to the SAT solver if the model does not satisfy the theory. This is repeated until a model for the theory is found or the SAT solver identifies unsatisfiability. In most solvers today, this idea is directly integrated into the solving algorithm of DPLL. For SMT solvers, the technique is called DPLL(T). The following example shall illustrate the methodology of DPLL(T). Consider the SMT formula F consisting of a combination of propositional and EUF terms.

$$F = (p \vee q) \wedge \left(\underbrace{h(a) = b}_{x_1} \right) \wedge \left(\underbrace{g(b) \neq h(g(b))}_{\neg x_2} \vee \underbrace{(a \neq b)}_{\neg x_3} \right) \wedge \left(\underbrace{g(b) = b}_{x_4} \vee \underbrace{h(b) = b}_{x_5} \right)$$

With each EUF term assumed to be a propositional symbol $x_1, \ldots, x_5$, the SAT solver considers the standard CNF $F_{CNF} = (p \vee q) \wedge (x_1) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_4 \vee x_5)$. A possible satisfying assignment is identified by the SAT solver as $\{p, x_1, \neg x_2, x_3, x_4, \neg x_5\}$. In the next step, that particular assignment is checked by an EUF solver. As x_3 implies that $a = b$, it results in a conflict in x_2: $g(b) \neq h(g(b)) \overset{x_4}{\rightarrow} b \neq h(b) \overset{x_3}{\rightarrow} b \neq h(a) \overset{x_1}{\rightarrow} b \neq b$. Hence, the assignment does not satisfy the original SMT formula. According to DPLL, the conflict clause $(\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4 \vee x_5)$ is added to the problem and the search is backtracked. The next assignment to be checked is $\{p, x_1, x_2, \neg x_3, x_4, \neg x_5\}$. Again, the assignment does not satisfy the theory as $g(b) = h(g(b)) \overset{x_4}{\rightarrow} b = h(b)$ conflicts with $h(b) \neq b$ implied by $\neg x_5$. Eventually, an assignment satisfying the theory such as $\{p, x_1, x_2, \neg x_3, x_4, x_5\}$ is found.

The above methodology can be improved by incremental theory solving, theory propagation and theory lemmas that steer backjumping of the underlying SAT solver [43]. With incremental theory solving, whenever a decision is made in the SAT solver, a partial assignment is checked whether it is still satisfiable by the theory. Theory propagation allows the theory solver to propagate decisions based on already made partial assignments. Theory lemmas improve the conflict clauses returned by the theory solver and allow the underlying DPLL algorithm to perform focused backjumps. In the formula F above, theory propagation can lead to faster satisfiability solving. For example, whenever x_1 and x_3 concurrently hold, x_5 has to hold too, i.e., $h(a) = b \overset{x_4}{\rightarrow} h(b) = b$.

Answer Set Programming modulo Theories More recently, this concept has been extended to ASP [45, 46]. In general, ASP modulo theories (ASPmT) relates to ASP similarly as SMT relates to SAT. Hence, theory atoms in stable models of a grounded logic program have to be derivable by the given rules and need to be checked through specialized solvers. In contrast to

SMT, special care has to be taken during the grounding phase. That is, variable replacement is limited in theory atoms as shown for example in [42]. A more detailed discussion of ASPmT and its application in system design is conducted in Chapter 3 of this monograph.

To allow the definition of difference (and other theory) terms within the ASP program, the input language of the solver has to be extended. To this end, the ASP solver *clingo 5* is utilized that allows for the definition of arbitrary theory languages [42]. Hence, a short review of the language definition as described in [42] is given. Here, a theory language definition is introduced by the `#theory` keyword followed by a unique name T and a set of theory term and theory atom definitions D_i with $i \in [1, n]$:

$$\texttt{\#theory T \{D}_1\texttt{; ...; D}_n\texttt{\}.}$$

A theory atom is characterized by a leading ampersand `&`, a predicate name `p`, and its arity `k` followed by its syntax definition. Theory atoms may occur in two forms distinguished by its definition. It either consists of a single theory term `t` or it is a concatenation of two theory terms `t, t'` connected by a set of theory operators $\diamond_i$:

$$\texttt{\&p/k : t, o} \quad \text{or} \quad \texttt{\&p/k : t, \{}\diamond_1\texttt{,...,}\diamond_m\texttt{\}, t', o.}$$

The final parameter `o` $\in$ `{head, body, any, directive}` determines the possible position in an ASP rule. Theory terms with name `t` are integral parts of a theory atom and contain a set of theory operator definitions `D`$_i$:

$$\texttt{t : \{D}_1\texttt{;...;D}_m\texttt{\}.}$$

In the logic program, a theory term `t` can have one of the following forms:

a constant term:	c	a function theory term:	$f(t_1, \ldots, t_k)$
a variable term:	v	a tuple theory term:	$(t_1, \ldots, t_l,)$
a binary theory term:	$t_1 \diamond t_2$	a set theory term:	$\{t_1, \ldots, t_l\}$
a unary theory term:	$\diamond t_2$	a list theory term:	$[t_1, \ldots, t_l]$

where each t_i is a theory term, $\diamond$ is a theory operator defined by some D_i, c and f are symbolic constants, v is a first-order variable, $k \geq 1$, and $l \geq 0$. Finally, a theory operator $\diamond$ characterizes the relations between theory terms. It may be a unary or a binary operator, whereas for binary theory operators, their associativity `a` can be defined as left or right. Thus, it has the form

$$\diamond\texttt{ : p, unary} \quad \text{or} \quad \diamond\texttt{ : p, binary, a}$$

where `p` ≥ 0 defines the precedence over other theory operators.

2.3 Multi-objective Optimization

The optimization is an integral part of the synthesis step. When a designer is only interested in one objective, e.g., power consumption, the various design candidates are totally ordered. A design candidate A that consumes 10 mW is clearly superior to a design candidate B that consumes 20 mW for the same functionality. In many real-life scenarios, however, the design is subject to multiple objectives concurrently.

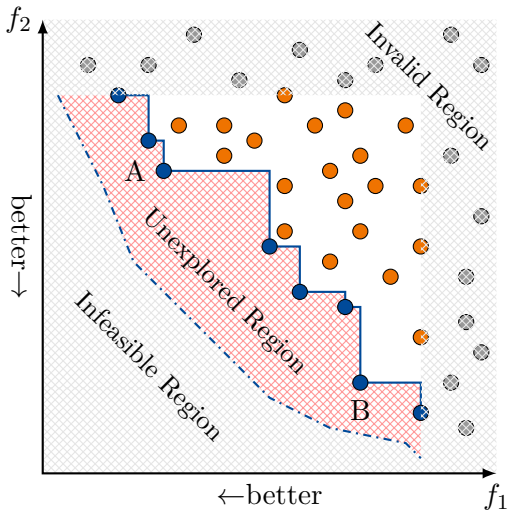


Figure 2.4: Regions in the objective space of a 2-dimensional minimization problem

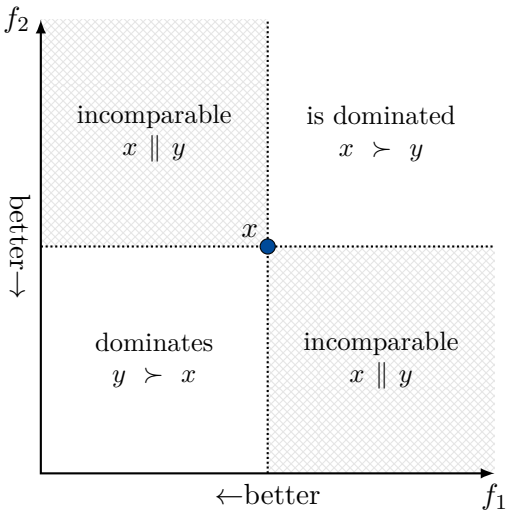


Figure 2.5: Dominance relation of objective vectors in a 2-dimensional minimization problem

If a solution candidate x conforms to the feasibility and validity constraints, it belongs to the sets X_F and X_V, respectively, with $X_V \subseteq X_F$. An exemplary result of an (incomplete) optimization run is given in Figure 2.4. The figure depicts the regions of the problem in the objective space. The gray lower left part does not contain any solutions as potential solutions do not conform to given feasibility constraints $h_i(x)$. Similar, solution candidates in the top and right gray regions do not conform to validity constraints $g_i(x)$. Only solutions that are located in the valid region, indicated by the white and red regions are potentially good ones, i.e., the orange and blue solutions. Note that Figure 2.4 is an illustrative example. Especially the shapes of the invalid and infeasible regions may be irregular and do not have to be connected. Optimizing the individual objectives simultaneously is generally not possible as they are often conflicting with each other, i.e., improving one objective implies the deterioration of another. For instance, optimizing for low power consumption and high performance is generally mutually exclusive as a low-power design is not able to deliver high performance and vice versa. Assume again the designs A and B from above. In addition to the power consumption of 10 mW and 20 mW, design A is evaluated to attain a latency of 100 µs while design B achieves an end-to-end latency of just 50 µs. While A is better with respect to power consumption, B achieves a higher performance. Neither A nor B can be considered superior to the other.

Definition 2.3.1 (Multi-objective Optimization Problem): Given a set of objective functions $f(x) = \{f_1(x), \ldots, f_n(x)\} \in O$, an n-dimensional multi-objective optimization problem (MOOP) is formulated as follows:

$$
\begin{aligned}
&\textit{optimize } f(x) \\
&\textit{subject to}: \\
&\quad h_i(x) = \top, \\
&\quad g_i(x) < 0,
\end{aligned}
$$

where P is the parameter space, $x \in P$ is the set of parameters, O is the objective space, $h_i(x)$ is the set of feasibility constraints, and $g_i(x)$ is the set of validity constraints.

Table 2.2: Classification of different dominance relations of objective vectors and non-dominated fronts assuming a minimization problem [47]

objective vectors ($i, j \in \{0, \ldots, n\}$)		relation	solution fronts ($x \in X, y \in Y$)	
$\forall i : f_i(x) < f_i(y)$	$x \succ\succ y$	strictly dominates	$X \succ\succ Y$	$\forall y \exists x : x \succ\succ y$
$x \succeq y \wedge \exists j : f_j(x) < f_j(y)$	$x \succ y$	dominates	$X \succ Y$	$\forall y \exists x : x \succ y$
–	–	better	$X \rhd Y$	$X \succeq Y \wedge X \neq Y$
$\forall i : f_i(x) \leq f_i(y)$	$x \succeq y$	weakly dominates	$X \succeq Y$	$\forall y \exists x : x \succeq y$
$x \not\succeq y \wedge y \not\succeq x$	$x \parallel y$	incomparable	$X \parallel Y$	$X \not\succeq Y \wedge Y \not\succeq X$

In contrast to single-objective optimization problems (SOOPs), the design candidates of MOOPs are not totally but rather partially ordered through the notion of dominance. The dominance relation $\succ$ is formally defined for n-dimensional objective vectors of two distinct solutions. A candidate solution x is said to dominate another solution y ($x \succ y$) if x is evaluated at least as good in every objective and better in at least one objective when compared to y.

Definition 2.3.2 (Dominance Relation): Without loss of generality, for a minimization problem with n objectives evaluated through the functions $f_1, \ldots, f_n$, the dominance relation $\succ$ between two solutions x and y is defined as follows:

$$x \succ y \leftrightarrow \forall i \in \{1, \ldots, n\} : f_i(x) \leq f_i(y) \ \wedge \exists j \in \{1, \ldots, n\} : f_j(x) < f_j(y). \tag{2.1}$$

The dominance relation can be further classified into strict and weak dominance as given in the left part of Table 2.2. If a solution vector x is evaluated better than another solution y in each objective, x is said to strictly dominate y ($x \succ\succ y$). The weak dominance relation $x \succeq y$ is similar to the standard dominance relation ($x \succ y$) but it explicitly allows for the equality of two solution vectors. A visualization is given in Figure 2.5. The design candidate x is given as a reference in a 2-dimensional minimization problem. Another design candidate y that is evaluated and located in the upper right quadrant of the coordinate system is dominated by x as both objectives f_1 and f_2 evaluate worse than for x. Contrary, if y was located in the lower left quadrant, x was dominated by y. Finally, if y was located in the crosshatched quadrants, neither of the two design candidates would be dominated by the other. In the upper left quadrant, y would be evaluated better by f_1 and worse by f_2. Analogue, in the lower right quadrant, the evaluation of f_2 would be better and f_1 would be worse. In these cases, x and y are said to be *incomparable* ($x \parallel y$) or mutually non-dominated.

On the basis of the dominance relation, *Pareto optimality* is defined.

Definition 2.3.3 (Pareto optimality): A design candidate x is said to be Pareto optimal if no valid dominating solution y exists: $\nexists y \in X_V : y \succ x$.

The set of all Pareto optimal solutions forms the Pareto front. Thus, by definition, solutions that are contained in the Pareto front X_P are therefore mutually non-dominated to each other. The Pareto front represents the best compromise solutions that can be found for a given MOOP. According to the extended dominance relation towards sets of solutions, the Pareto front weakly dominates every other possible front, that can be formed by the solution candidates. This is shown in the right part of Table 2.2.

Obtaining the *true* Pareto front is, however, hard in general. Especially in the domain of

system design, it has been shown to be NP-complete [48]. Hence, an exhaustive search for all Pareto optimal solutions is often not feasible due to the vast search space. The result of an optimization run is therefore only an approximate Pareto front while a fraction of the design space remains unexplored. Although some solutions might not be found, the approximated Pareto front still only contains mutually non-dominated solutions, i.e., it is domination-free. Note that Pareto front approximations obtained by individual optimization runs generally contain varying solution candidates as different regions of the search space have been explored. For the sake of clarity, the term *non-dominated front* will be used in the remainder of this monograph to refer to an approximate Pareto front, i.e., whenever not the entire search space has been explored.

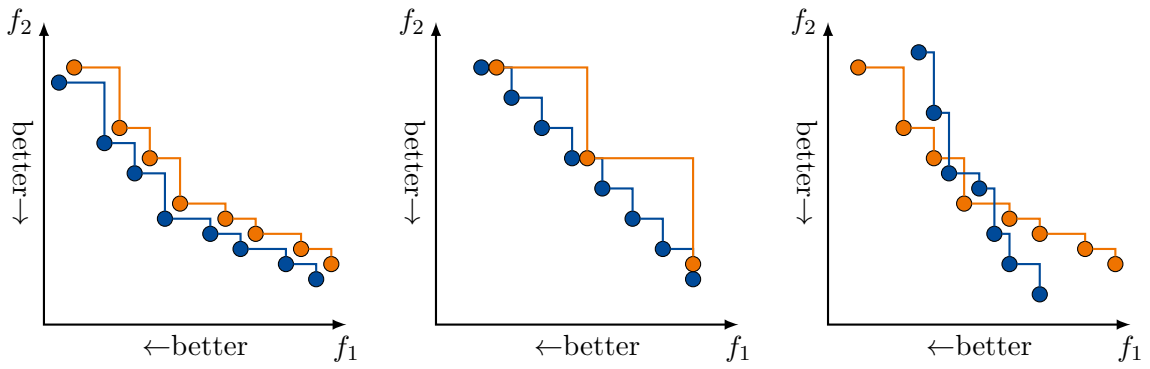
Consider again the (incomplete) optimization run in Figure 2.4. While the gray solutions are filtered out as they are not located in the valid region, each orange solution is valid but dominated by at least one other solution. The non-dominated front is represented by the blue solutions as none of them is dominated by another solution. However, as the optimization run is incomplete (e.g., due to a vast search space), the non-dominated front is not guaranteed to be Pareto optimal. Further solutions might be located in the unexplored region (red) of the design space and would be contained in the *true* Pareto front, indicated by the dashed curve.

2.3.1 Quality Indicators

Considering that only a fraction of the search space is explored, it becomes imperative to evaluate the performance of different approaches. Therefore, generally two characteristics are of interest: the convergence and the diversity[2-5] of the obtained non-dominated front. While the convergence assesses the proximity of the non-dominated front to the true Pareto front, the diversity evaluates the distribution of the individual solutions within the non-dominated front. Hence, a front that is located close to the true Pareto front has a good convergence, cf. Figure 2.6(a). Evenly distributed solutions constitute a good diversity, cf. Figure 2.6(b). While the blue fronts have a clearly better quality than the orange ones in the first two cases, Figure 2.6(c) shows an example where the assessment is not trivial. On the one hand, the orange front contains better solutions with respect to the first objective f_1. On the other hand, the blue front contains better solutions regarding the second objective f_2. As a remedy, numerous formal quality indicators have been presented in the literature that aim at assessing the quality of non-dominated fronts.

Dominance Ranking Considering a collection of approximate Pareto fronts C, dominance ranking [50] calculates a rank for each individual front $C_i \in C$ within C based on the dominance relation. The rank assigned to each front C_i corresponds to the number of other fronts C_j that are better than C_i: $rank(C_i) = 1 + |\{C_j \in C : C_j \triangleright C_i\}|$. Thus, a high rank signifies a worse front than a low rank. As a result, the dominance ranking yields a relative statement about the convergence of the non-dominated front with respect to the entire collection C. It cannot, however, provide information on how much better or worse a front is compared to the rest and if two fronts C_i and C_j are incomparable to each other, the dominance ranking will result in the same ranks, cf. Figure 2.6(c).

[2-5] In some works (e.g., [49]), the diversity of the non-dominated front is further classified into uniformity and spread which is not covered in the present monograph.



(a) Convergence differences **(b)** Diversity differences **(c)** Unclear quality

Figure 2.6: Comparison of two non-dominated fronts obtained by different optimization runs. (a) Both runs have a similar diversity, but blue has a better convergence. (b) Both runs result in a similar convergence, but blue has a higher diversity. (c) It is unclear which front has a better convergence/diversity.

Generational Distance The generational distance (GD) [51] is another quality indicator to assess a non-dominated front with respect to the convergence. In contrast to dominance ranking, the GD needs a reference front to be calculated but quantifies the quality of the non-dominated front. Given a non-dominated front X and a reference front A, the generational distance $GD(X)$ is based on the minimal Euclidean distance d_i of every solution candidate $i \in X$ to the reference front A:

$$GD(X) = \frac{1}{n} \cdot \sqrt{\sum_{i=1}^{n} d_i^2}, \tag{2.2}$$

where n is the number of solutions in X.

The main problem of the GD is its averaging character. With an increasing number of solutions in the non-dominated front, the quality measure decreases and the front is evaluated better. A remedy has been proposed as the GD_p indicator in [52]:

$$GD_p(X) = \frac{1}{\sqrt{n}} \cdot \sqrt{\sum_{i=1}^{n} d_i^2}. \tag{2.3}$$

Using the power mean prevents the preference towards large non-dominated fronts and makes the comparison of fronts of different size more fair.

Convergence and Diversity Measure In [53], two quality indicators have been proposed. The convergence measure is calculated similar to the GD but without applying the square root to the accumulated distances in the numerator. Thus, the problem comparing differently sized fronts does not arise.

Regarding measuring the diversity of a non-dominated front, an entropy-based approached is used. To calculate the diversity measure, the n-dimensional objective space is therefore first projected onto a $(n-1)$-dimensional hyperplane. The hyperplane is then divided into N small grids resulting in $(n-1)$-dimensional boxes. Two binary variables H_i and h_i are calculated

for each grid $i \in N$ that define whether a point of a reference front A and a point of the non-dominated front X lie within each box, respectively. Depending on the values of H_i and h_i, and their direct neighbors, a distribution function m is defined. Finally, all results of $m(H_i)$ and $m(h_i)$ are accumulated and set into relation with each other:

$$D(X) = \frac{\sum_{\substack{i \in N \\ h_i \neq 0}} m(h_i)}{\sum_{\substack{i \in N \\ H_i \neq 0}} m(H_i)} \tag{2.4}$$

The grid size and the exact definition of the distribution function m are set by the user. While small, and thus many grids increase the resolution of the indicator, it simultaneously increases the complexity of the indicator. As the grid is also dependent on the number of objectives (i.e., the dimensionality of the problem), the complexity increases exponentially with the number of objectives. Furthermore, a proposal for m is given by the authors only for 2-dimensional problems. A definition for more than two objectives remains unclear.

Entropy Indicator Instead of the distribution function m, in [54] a Gaussian influence function is used to determine the entropy of the non-dominated set. The advantage of this approach is the general applicability for more than two objectives as it is based on the Euclidean distance. To apply the entropy indicator to an n-dimensional MOOP, the front has to be projected onto an $(n-1)$-dimensional hyperplane orthogonal to the line between normalized ideal and nadir points[2-6]. The hyperplane is divided into a grid of $a_1 \times \ldots \times a_{n-1}$ cells, where a_1 to a_{n-1} (i.e., the grid size) are problem specific and user defined. Each solution x in the projected non-dominated front X influences the density of a grid cell i by a certain degree according to a user-defined influence function $\Omega(r_{x \to i})$, where $r_{x \to i}$ is the Euclidean distance from x to grid cell i:

$$D(i) = \sum_{x \in X} \Omega(r_{x \to i}). \tag{2.5}$$

The original work [54] proposes to utilize the Gaussian influence function

$$\Omega(r) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \exp\left(\frac{-r^2}{\sigma^2}\right) \tag{2.6}$$

with $\sigma = 1/6$. In order to calculate the entropy for the entire front, the density has to be normalized to the accumulated density of all grid cells:

$$\rho_i = \frac{D_i}{\sum_{k=1}^{a_1 \cdot \ldots \cdot a_{n-1}} D_k}. \tag{2.7}$$

Finally, the entropy is defined as:

$$H = -\sum_{k=1}^{a_1 \cdot \ldots \cdot a_{n-1}} \rho_k \cdot \ln(\rho_k). \tag{2.8}$$

The entropy indicator evaluates the flatness of the distribution with a higher value indicating a better diversity of the solutions. Although the entropy measure is more generally applicable than the diversity measure above, it is still exponential in the number objectives.

[2-6] In MOOPs, the ideal and nadir points represent the best and worst of each objective value in the non-dominated set, respectively.

Diversity Comparison Indicator The diversity comparison indicator (DCI) [49] implements another approach based on the entropy of the non-dominated front. Similar to the diversity measure, DCI also spans a grid to assess the distribution of solutions within the objective space. However, it only considers boxes that contain solutions and therefore achieves a quadratic complexity independent of the grid size and the number of objectives. The grid is constructed manually and should be located near the true Pareto front. Hence, the authors propose to use the *satisfied* region, i.e., a user defined region that is expected to be located near the Pareto front. If it is not possible to determine this region, the (approximated) ideal and nadir points of the problem can be used instead.

Applying DCI to a single non-dominated front is not feasible, i.e., the indicator does not produce an absolute quality value. Rather, it is applied to a set of non-dominated fronts to assess the diversity of each front with respect to the entire set. Therefore, a combined front is created to determine the grid boxes that are used (i.e., active) for the calculation. Grids that contain now-dominated solutions are not considered in the assessment. For each active grid box h, the *contribution degree* $CD(P,h)$ of each individual non-dominated front P is calculated:

$$CD(P,h) = \begin{cases} 1 - D(P,h)^2/(n+1) & \text{if } D(P,h) < \sqrt{m+1} \\ 0 & \text{else} \end{cases}, \tag{2.9}$$

where $D(P,h)$ equals the minimum Euclidean distance of P to grid box h and n is the number of objectives. The DCI of each front P is finally calculated by averaging the contribution degree to each active box h. While the complexity of the algorithm is not influenced by the number of grid boxes, it is sensitive towards the size of the grid. If the grid size is chosen too small, the contribution of solutions may be insignificant towards the active boxes. Contrary, if the grid size is too large, too many solutions will be present in each grid box and the DCI will fail to distinguish differences in the diversity of individual non-dominated fronts.

Hypervolume The hypervolume quality indicator[55] calculates the area dominated by the solutions present in the non-dominated front. Considering an n-dimensional MOOP, for each solution candidate x_i, the point in objective space $P_{x_i} = (f_1(x_i), \ldots, f_n(x_i))$ and a reference point P_r span a hyperbox H_i. The intersections of all hyperboxes yield the hypervolume H for the obtained non-dominated front. A larger hypervolume w.r.t. the same reference point signifies a better quality of the front. Choosing the reference point is important for the relevance of the approach. A good location for the reference point is, for example, at the crossing point of invalid region of the individual objective functions. There, it is guaranteed to be dominated by each solution, and it is identical for every non-dominated front. In the 2-dimensional case, each solution candidate spans a rectangular hyperbox. The white region in Figure 2.4, for example, represents the area H that is calculated by the hypervolume measure when the reference is located at the intersection of the invalid regions. As a unary quality indicator, the hypervolume can be calculated for each front individually, i.e., a reference front is not needed. However, it prefers convex over concave regions and therefore might overrate certain solutions [55]. The complexity to calculate the hypervolume is $O(n^{d-2} \log n)$ as analyzed in [56].

Epsilon Dominance The Epsilon Dominance (ϵ-Dominance) is an indicator to measure the convergence of non-dominated fronts. It is introduced in [57] and quantifies the quality differences of two solutions x and y by a factor ϵ. The solution x is said to ϵ-dominate y ($x \succ_\epsilon y$) for

some $\epsilon > 0$ if $f(x)$ evaluates better for each objective $1, \ldots, n$ than $f(y)$ by a factor of $(1+\epsilon)$, i.e., $\forall i \in \{1, \ldots, n\} : (1+\epsilon) \cdot f_i(x) \geq f_i(y)$. Inspired by this concept, the binary ϵ-indicator $I_\epsilon(X, Y)$ is proposed in [47] which extends the concept of the epsilon dominance towards two non-dominated fronts X and Y. The indicator $I_\epsilon(X, Y)$ is defined as the greatest lower bound (infimum) of ϵ such that each solution vector $y \in Y$ is ϵ-dominated by at least one solution $x \in X$, i.e., $I_\epsilon(X, Y) = \inf_{\epsilon \in \mathbb{R}}\{\forall y \in Y : \exists x \in X : x \succeq_\epsilon y\}$. Hence, it represents a factor by which a non-dominated front is worse than another with respect to all objectives. Calculating the binary ϵ-dominance in practice has a time complexity of $\mathcal{O}(n \cdot |X| \cdot |Y|)$:

$$I_\epsilon(X, Y) = \max_{y \in Y} \min_{x \in X} \max_{i=1}^{n} \frac{x_i}{y_i}. \tag{2.10}$$

The binary ϵ-indicator can be used to identify incomparable fronts as its application yields conflicting results. For instance, considering the problem in Figure 2.6(c), applying the binary ϵ-indicator results in $I_\epsilon(BLUE, ORANGE) > 1$ and simultaneously $I_\epsilon(ORANGE, BLUE) > 1$. Depending on the order, one front or the other is considered to be worse. Hence, for the identification, it is imperative to calculate the binary ϵ-indicator in both directions.

Based on the binary indicator, a unary ϵ-indicator $I_\epsilon(X)$ can be used instead. Given a reference P, an absolute quality value can be calculated for each front X, i.e., $I_\epsilon(X) = I_\epsilon(X, P)$. Ideally, the true Pareto front is taken as a reference. However, as discussed above, the true Pareto front is usually not known. A combined front of all non-dominated solutions can be used instead. By implication, the combined front weakly dominates every other non-dominated front such that the quality of each front can be directly compared.

2.3.2 Optimization Approaches

The aim of solving a multi-objective optimization problem is to provide Pareto optimal or at least non-dominated solution candidates to a decision maker (DM). The DM is expected to have domain knowledge and can select the most appropriate solution for implementation. Depending on whether the preferences of the DM are known before the optimization run or not, different approaches for solving a MOOP must be followed. A priori approaches are available whenever preferences of the DM are known before an optimization run. A posteriori methods construct the Pareto front first and the DM selects a preferred solution afterwards.

If the preference towards a specific objective (e.g., performance) is known before the problem is being tackled, the generally formulated MOOP can be transformed into a SOOP. Therefore, only one objective is being optimized while the remaining objectives are converted into constraints with an upper bound ϵ [58]. Note that the result of this SOOP is not guaranteed to be Pareto optimal w.r.t. the original MOOP as only one minimization is executed. A lexicographic approach (e.g. [59]) can be used if, instead of one specific objective, the order of objectives is defined by the DM, i.e., objective f_1 is more important than f_2 and so on. This is enabled by a series of SOOPs that have to be executed. First, only the most important objective is optimized resulting in the best possible objective value. The next iteration optimizes the second to most important objective while adding a constraint to prevent the previous objective from deteriorating. This procedure is repeated for the remaining objectives and the final optimization yields one Pareto optimal solution of the corresponding MOOP.

A solution to the MOOP can also be obtained by scalarization where the problem is again transformed to a single objective. An example is the weighting method [60]. In the weighting method, a single objective is constructed as the sum of all n objective functions f_i multiplied

with individual weights w_i, i.e., $\sum_{i=1}^{n} w_i \cdot f_i(x)$. The function is then minimized. If a solution exists with $w_i > 0$ for all i, the solution is Pareto optimal for the corresponding MOOP[2-7].

In this monograph, a more thorough investigation of a priori approaches is out of scope. The interested reader is referenced to [61] and the references therein for further studies. In the following, preferences by the DM are instead assumed to be unavailable before the optimization is executed. That is, all Pareto optimal solutions are considered equally good during the optimization run and the emphasis is put on a posteriori approaches. In general, it is possible to adapt most a priori methods such that they deliver a complete Pareto front. Assume, for instance, the application of the lexicographic approach for a 2-dimensional minimization problem. At first, the first Pareto optimal solution $x_{P1} = (f_1(x_1), f_2(x_1))$ is obtained as described above. As it is guaranteed that there is no better objective value for f_1 than $f_1(x_1)$, any further Pareto optimal solutions achieve a worse quality in f_1 but a better quality in f_2. Therefore, two constraints of the form $f_1(x_2) > f_1(x_1)$ and $f_2(x_2) < f_2(x_1)$ are added. The optimization is executed as before and a second Pareto optimal solution is obtained. This is repeated until no further solution can be found, that conforms to the newly added constraints. The advantage of this approach is that it is able to find all Pareto optimal solutions. However, for optimization problems with $n > 2$ objectives, the procedure had to be executed $(n-1)$ times for each Pareto optimal solution. This leads to an exponential increase in individual optimization runs and degrades performance of the optimization.

As a consequence, more specialized methods have been presented in the literature that aim at generating the set of optimal solutions for the DM to decide. These approaches can be classified into exact and heuristic approaches. While heuristic approaches generally only yield an approximate Pareto front, exact approaches can generate the true Pareto front.

Heuristic Approaches Heuristic approaches basically aim at generating Pareto optimal solutions by randomly traversing the design space and automatically steering the search towards promising regions. The idea of all heuristic approaches is that small changes in the parameter space lead to small changes in the objective space. For example, mapping a single task onto a different CPU while maintaining the binding decisions of the remaining tasks, alters the entire set of objectives by a small degree. Hence, decisions are progressively adjusted throughout the execution of an optimization run where only the good solutions are kept for further modifications. This is repeated until an abortion criterion is reached such as a maximum number of generations (i.e., steps), a desired quality is achieved, no better decisions can be found, or a combination of all. While this basic methodology guarantees a constant improvement of the acquired design candidates, it is subject to the problem of local optima. Local optima are situations during the search where individual changes do not improve the current design but there exist areas in the parameter space that would lead to better designs. A remedy to this problem are randomized decisions that are injected into the search. While most of the time only local modifications to the decisions are made, once in a while, a (completely) random set of decisions is generated and will be used instead. Figure 2.7 depicts the problem of local optima with the aid of a single-objective minimization problem. For the sake of the argument, the parameter space P translates into the objective space O through an (upfront unknown) non-linear function, represented by the black curve. The function has three local optima, indicated by A, B, and C of which only C constitutes the global optimum. Assuming that a

[2-7] If the problem is non-convex, the obtained solution may only be locally Pareto-optimal.

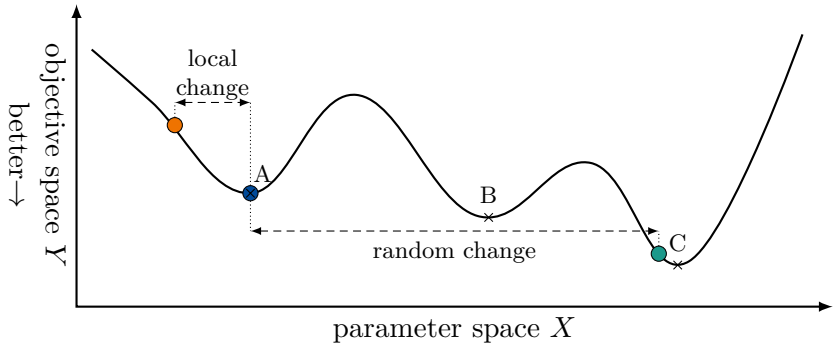


Figure 2.7: Local optima of a single-objective minimization problem

design candidate is already acquired in the optimum A (blue dot), a local modification of the parameters results in a deterioration (i.e., orange dot). On the contrary, a random change of the parameters may lead to an improvement as indicated by the turquoise dot that is located next to the global optimum C. Note that a random change does not guarantee an escape from the local optimum. Whenever the randomly generated design is worse than the reference design, the search will continue in the neighborhood of the latter. This leads to the first problem of heuristic approaches: the achievement of a global optimum cannot be proven if the exact objective function is unknown.

The utilization of an approach such as the above where only one solution is kept as reference (e.g., simulated annealing [62] and the downhill simplex algorithm [63]) is not suited for optimization problems with multiple objectives. For solving MOOPs, population-based heuristic prevail. Here, not only one design candidate (i.e., individual) is kept, but a set of individuals (i.e., a population) is initially generated that is evaluated and improved in every generation. Most of them are inspired by nature. Ant colony optimization (ACO) [64] is, for example, motivated by the behavior of ants searching for food. Individual ants that find promising food sources secrete pheromones to attract other ants to also move into that direction. Analogously, good design candidates indicate promising regions of the search space and the search is steered towards that direction. Particle swarm optimization (PSO) [65] simulates the behavior of swarms where many individuals cooperate to achieve a common goal.

A third important approach are multi-objective evolutionary algorithms (MOEAs) that are based on the natural selection process. The idea is that properties of good design candidates are passed on to subsequent generations and bad areas in the parameter space are discarded. An individual is first created as a genotype representing the decisions taken in the parameter space. In a second step, the genotype is than translated into a phenotype that can be used to assess the design candidate in the objective space. The search is steered by the concepts of recombination and mutation. While the former picks up on the idea of local changes in the parameter space, the latter is responsible for random changes to escape the local optima. During recombination, the genotypes of two or more individuals (i.e., parents) are merged to form a new child individual. In contrast, mutation creates an offspring by changing random genomes of one individual only. The newly created children form the next generation and are subsequently decoded, evaluated, and recombined again. Although, the general methodologies are similar throughout different evolutionary approaches, the details on how to manage Pareto optimal individuals and select parents for the succeeding generations vary.

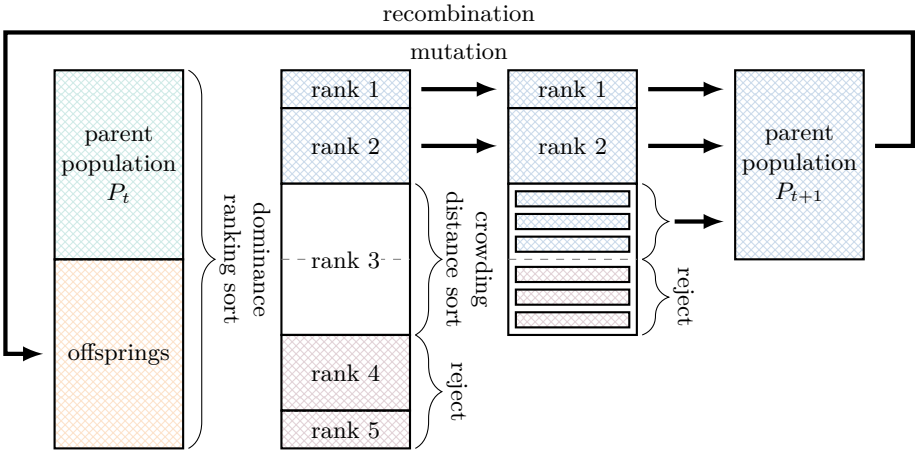


Figure 2.8: Execution of one generation in NSGA-II according to [69]

The strength Pareto evolutionary algorithm (SPEA) [66] and its successor SPEA2 [67] save the non-dominated solutions in an external bounded archive, i.e., separated from the population. The selection process is steered by the fitness value of each individual, where a better fitness increases the probability of being selected for recombination. An individual is assigned a better fitness when there exist few other solutions that dominate it. Hence, Pareto optimal solutions have the best fitness. Whenever the number of non-dominated solutions exceeds the bound of the archive, it is pruned in a way such that the diversity of solutions is kept. The non-dominated sorting genetic algorithm (NSGA) [68] and its successors [69, 70] pursue a different way. Here, no external archive of non-dominated solutions exists but instead, the population contains all individuals of a generation. The methodology of NSGA-II [69] is depicted in Figure 2.8. In each generation t, a parent population P_t of size $|P|$ generates an equally sized set of offsprings through recombination and mutation. The selection of the next parent population P_{t+1} is a two-step process. At first, the combined population is sorted by determining the dominance rank of each individual on the basis of its phenotype. Starting with rank 1, all ranks are successively added to P_{t+1} whilst the number of individuals in the corresponding rank does not exceed the population size. In the example given in Figure 2.8, adding all rank 3 individuals would exceed the population size $|P|$. In order to select the remaining individuals to be added, the borderline rank (i.e., rank 3) has to be sorted internally. Therefore, the crowding distance is determined that represents the diversity of each individual. Only the most diverse solutions are added to P_{t+1} while the rest is rejected.

The main advantages of heuristic approaches are continuously improving solutions and a high diversity through randomized decisions. Especially, population-based approaches such as MOEAs are able to approximate the true Pareto front in a reasonable time. However, these advantages are opposed by three major problems. First, the generation of an initial solution or population is problematic if only a few solutions are feasible and valid. Randomly generated solutions may be located in a region of the search space, that do not conform to feasibility and validity constraints. Although random decisions will eventually lead the search towards valid regions, local parameter changes will only result in further invalid solutions. Second, there is generally no way of knowing whether an acquired solution or front is the true optimum of the problem. This is explained by the lack of a systematical search. It is unknown which areas

of the search have already been visited and could contain additional optima. Furthermore, not finding any solutions with a heuristic approach does not prove that the given problem is insoluble. Finally, heuristics tend to run into saturation where the same regions of the search space are visited over and over. For example, recombination in MOEAs is increasingly less likely to produce new individuals when consisting solutions get more similar. Although, random decisions (i.e., mutations) may escape these regions, the probability of keeping these for the next population diminishes when numerous better (but similar) solutions already exist.

Exact Approaches In contrast to heuristics, formal approaches explore the search space systematically. This allows, at least in theory, to find all Pareto optimal solutions without repeatedly visiting the same regions of the search space.

In its simplest form, all possible decisions are enumerated, optimal solutions are saved, and the rest is rejected. Considering a synthesis problem where ten software tasks shall be mapped onto three processing elements, with the enumeration approach had to evaluate $3^{10} \approx 6 \cdot 10^4$ different solutions. If the number of tasks is doubled, it would already lead to $3^{20} \approx 3.5 \cdot 10^9$ solutions. Hence, the exponential growth makes the simple enumeration infeasible in practice. More advanced formal approaches utilize mathematical models to encode the problem and the corresponding constraints. These models are subsequently decoded by specialized solvers that search for solutions that fulfill the constraints. The advantage of these approaches is that the integrated constraint encoding prune effectively invalid regions of the design space such that many solutions are not unnecessarily visited. Linear programming (LP) and SAT are two prominent representatives. Linear programming encodes an optimization problem by a set of linear equations and inequalities. While there exist efficient algorithms to solve an LP, it is not suitable for problems with discrete parameters. Thus, integer linear programming (ILP) must be used instead that only allows integer variables but is shown to be NP-complete. Solving a single-objective ILP (exactly) is commonly done by first solving the relaxation in form of an LP to acquire upper and lower bounds. In subsequent steps, the problem is continuously narrowed by splitting it into subproblems (branch-and-bound algorithm [71]) or the introduction of additional inequalities (cutting plane algorithm [72]) until a discrete solution is found.

While SAT is primarily not an optimization technique, it can be utilized to explore the design space systematically. This can be achieved by applying a modified DPLL algorithm as detailed in section 2.2. In contrast to the default DPLL algorithm, the aim is not answering whether a given propositional formula is satisfiable but to find all satisfying assignments. To this end, whenever an assignment is found, the solution is evaluated and compared to previously found assignments and, if optimal, saved into an archive. Simultaneously, non-optimal assignments are removed from the archive. Finally, a conflict clause is added to the original problem that prevents finding the same assignment again and the solver executes the backtracking phase. If the extended problem becomes unsatisfiable, the optimal solution has been found. Note that this methodology is equally applicable for both SOOPs and MOOPs. In fact, the DPLL approach is applicable to multi-objective ILP problems [73] by traversing the graph of subproblems. The asprin framework [74] implements the handling of optimizations in ASP. It is able to define and evaluate preference relations over stable models of a logic program. The generation of optimal stable models is a stepwise process based on successive calls to an underlying ASP solver. Therefore, a stable model is supplied to an external preference program that aims to deliver a strictly better stable model, if one exists. For MOOPs, this implies that only after a solution is proven to be Pareto-optimal, incomparable solutions are explored. In

contrast to the DPLL-based approach above, optimal stable models are not saved to an archive but instead are saved via facts within the logic program.

Although the application of exact approaches, i.e., ILP, SAT, and ASP, improves the search space exploration significantly compared to simple enumeration, the complexity stays the same. Hence, a growing decision space exponentially increases the number of solution candidates to be considered. This prevents these techniques to exhaustively explore the design space especially for large problem instances. Furthermore, found solution candidates are often clustered in local regions of the decision space. While local regions are exited by design eventually, formal approaches might remain there for a long period. If that happens, and they are aborted prior to completion, they provide less diversified solutions than heuristic approaches would. Hybrid approaches are proposed as a remedy for this situation. Such hybrid approaches combine the high diversity of heuristics and the systematic search for feasible solutions. In contrast to pure heuristics, the population of solution candidates is not selected randomly. Instead, the problem is encoded formally and a systematic solver is used to generate solutions. To obtain diverse solutions, the internal decision strategy of the solver is subject to recombination and mutation operations. Hence, different decision strategies constitute the population of one generation. An example is presented in [75], where the problem is encoded by means of a propositional formula and search is executed with an MOEA. There, the genotype consists of the phases and priorities of the corresponding variables in the propositional formula. In turn, the phases and priorities are used as the search heuristic in the SAT solver transforming the genotype into the phenotype. This way, the approach solves the problem of finding a feasible initial population as the SAT solver delivers a satisfying assignment. However, the other problems of MOEAs remain. Thus, as the general methodology is still based on random decisions, the status of the exploration coverage is unknown. Furthermore, the hybrid approach retains the tendency to run into saturation if local optima are not escaped.

2.4 Related Work

Throughout the last decades, much research has been conducted in the field of embedded computer design and connecting areas. So far, this chapter provided the fundamentals of model-based design. The closing of the chapter shall present previous work directly related to the individual problems addressed in the remainder of this monograph. By no means, the discourse is exhaustive. It merely represents an overview of available approaches and sets them into context of the approaches proposed in the thesis at hand.

2.4.1 System Synthesis

The automated synthesis of a behavioral description into an implementation has been subject to research for more than four decades. It became a necessity after the digital circuits got too complex to be developed entirely manually. At first, the emphasis was put on the hardware side of digital systems. One of the first language approaches was developed with ISPS [76] that aims at describing the computer system at RTL. With the adoption of very-large-scale integration (VLSI) methodologies, other languages to model the circuit became available. Starting in 1985, the VHDL hardware description language was introduced that was standardized by the IEEE in 1987 [17] and is still used today in RTL design.

Despite the availability of languages, only in the late eighties of the 20^{th} century, automated

synthesis approaches became available. The approaches CATHEDRAL-II [77] and the Olympus Synthesis System [78] are two early examples. Both works address the architecture synthesis of DSPs. The approaches translate a behavioral description into an application-specific integrated circuit (ASIC). As such, the authors only focus on the creation of integrated circuits, a consideration of software components is therefore not part of the approaches.

In 1992, Gupta et al. proposed in [79] one of the first synthesis approaches at system level that incorporated hardware/software co-design principles. They considered behavioral applications modeled through control flow graphs. During synthesis, the individual blocks of the graph are partitioned into hardware and software. Therefore, the target platform consists of a programmable processor that is assisted by several application-specific hardware components. The hardware is further synthesized using the Olympus Synthesis System [78] while the software is assumed to be modeled through C. The partitioning of the application is executed iteratively. Starting with all components implemented in hardware, the algorithm strives to successively shift as many tasks as possible towards the software side as long as feasibility and timing constraints can still be satisfied. The work assumes a fixed platform where the processor communicates with the hardware accelerators through common interfaces over a central bus. A more general approach on communication synthesis was proposed by [80]. The paper proposes to allocate appropriate communication components from a given library. Hence, the approach creates an application-specific communication infrastructure. Communication between tasks is modeled through channels that are mapped to allocated communication components fitting the specified protocols, timing, and area constraints. The authors propose a mapping algorithm that is based on a decision tree and executed automatically until a feasible implementation is obtained.

Obtaining optimal designs, rather than only finding a feasible solution became an important research goal around the late nineties. The authors of [48] first showed that the synthesis was in fact an NP-complete problem. Furthermore, they proposed combining a model-based design with a MOEA. The idea in this work is to randomly choose an initial set of implementations consisting of arbitrary allocation and binding decisions, each encoded as a specialized genotype. The genotype is then decoded and evaluated through objective functions. To obtain the timing properties of the implementation, a second heuristic is used that enables the use of standard list-scheduling techniques. In each generation, the implementations, i.e., genotypes, are permuted to generate new design candidates. This allows a diverse exploration of the search space. Ultimately, a set of mutually non-dominated implementations is returned.

In the following years, multiple research groups proposed and developed system synthesis [81–83] and DSE frameworks [84–89]. While Metropolis [81] does not define any specific tools, it provides a framework with a common language to support verification, simulation, and the analysis of designs. The language provides the definition of the behavior of the application through tasks and channels modeled by process networks. As a hardware model, the language allows for the definition of processors, memories, and communication infrastructure. The authors assume a platform-based design where allocated resources and their topology is largely predefined. The interfaces of the individual modules of Metropolis further permit the integration of specified analysis techniques such as the generation of quasi-static schedules.

The authors of [82] proposed the synthesis system xPilot. It takes a behavioral description specified in synthesizable C or SystemC code as input. The framework analyzes the code and extracts processes, channels, and the communication topology. To finally synthesize a domain-specific system-on-chip, xPilot relies on manual partitioning. Hence, a decision maker has to

determine which components shall be implemented in software and hardware. With a given mapping, the framework generates a feasible schedule using LP. The result of a synthesis run is a description at RTL in either VHDL or SystemC.

Dömer et al. proposed the system-on-chip environment (SCE) [83] for the synthesis of multi-processor system-on-chips (SoCs). Similar to xPilot, in SCE, the design decisions are entered manually through an interactive user interface or shell scripts. However, SCE explicitly includes an additional automated model refinement step. That is, based on the decisions, the specification is transformed into an implementation using specified hardware and software databases and adheres to specified constraints. SCE allows the (manual) exploration of multiple substeps in the design process. This includes the architecture, scheduling, and network exploration as well as the communication, RTL, and software synthesis. The software schedule assumes the execution of a real-time operating system on each processor. There, the designer can choose the scheduling strategy that is utilized for the analysis. Ultimately, SCE generates an implementation model that consists of RTL Verilog code and binaries for each processor.

Daedalus [85, 90] and SystemCoDesinger [86], in contrast to the tools above, are DSE frameworks that automate the decision-making. Both works provide integrated environments that yield promising design alternatives at an early stage of the design process. Although the scopes of both works are put on streaming and data flow applications, the input of both approaches differs slightly. While the input of Daedalus is modeled through Kahn Process Networks, SystemCoDesigner uses actor oriented SystemC descriptions. The DSE is based on MOEAs to trade off the often conflicting design objectives. Additionally, Deadalus can be configured to perform an exhaustive DSE to further investigate certain areas of the design space. The results of the DSE are RTL implementations that can be used for rapid prototyping.

Pimentel et al. state in a more recent study [91] that DSE approaches can essentially be categorized into two types. First, (meta-)heuristics like evolutionary algorithms, particle swarm and ant colony optimization (e.g. [86, 92–94]) and second, exact methods such as ILP and branch-and-bound algorithms (e.g. [73, 95, 96]).

Most of the works presented in the field of meta-heuristics extend basic techniques in order to respect domain specific characteristics. For example, in [93], the authors extend genetic algorithms by utilizing domain knowledge. They state, that small differences in design decisions lead to similar system implementations. They overcome the problem of finding infeasible initial solutions by only considering homogeneous architectures where all tasks are executable on every processor. The assumption of a homogeneous hardware platform also permits the identification of symmetrical design points that can be pruned from the search space safely. While this is, in theory, also possible for heterogeneous architectures, symmetries are less likely to be found.

Another approach (e.g. [86, 89, 92]) of handling the infeasibility problem is to integrate dedicated constraint solvers into a MOEA. The work of Schlichter et al. [92] integrates, for example, a SAT solver into a MOEA. Therefore, the synthesis problem is encoded symbolically as, for example, proposed in [34]. Here, the decisions are not directly controlled by the randomized search algorithm of the MOEA, but the heuristic of the decision variables is subject to exploration. Altering the path through the decision tree permits finding more diverse design candidates while maintaining the feasibility of each design point.

The main problem with heuristic approaches is that the design space is not covered systematically. Designs may be found multiple times and are evaluated unnecessarily often. Exact approaches provide a solution to this problem by exploring the design space purely systematically. Here, the feasibility encoding is exhaustively explored. Commonly, the synthesis is

encoded through pseudo Boolean logic, for example in [73, 97, 98]. For a long time, those methods were restricted to single-objective optimization problems only. As one of the few exceptions, Lukasiewycz et al. [73] present a complete multi-objective Pseudo-Boolean solver based on branch-and-bound algorithms. The results show that this technique is able to find all the proven optimal solutions for small problems in a short time. While the encoding of feasible mapping and routing decisions into Boolean formulas led to efficient synthesis frameworks by leveraging enhancements in state-of-the-art SAT solvers, numerical and non-linear problems (such as scheduling) are not easily representable in Boolean logic.

As a consequence, SMT-based techniques, i.e. the combination of Boolean logic (traditionally SAT) and various background theories, have been developed in the domain of embedded systems synthesis [95, 99–102]. In one of the first works on SMT-based approaches, Reimann et al. [100] integrate a background theory solver to evaluate partial assignments of an underlying SAT-solver. In [103], they extend their approach by constructing schedules via an external theory solver. However, here, they only consider complete assignments, i.e., mapping and routing is already completed.

The authors of [95] present another SMT-based method for scheduling analysis in a background theory. Similar to the proposed approach in the thesis at hand, they utilize quantifier-free integer difference logic (QF–IDL) as a background theory to identify valid schedules. The main advantages of QF–IDL are its decidability in polynomial time [104] and the possibility to leverage the solving process to directly analyze and propagate observed conflicts. Yet, the work of [95] does neither support heterogeneous architectures nor multi-objective optimizations as proposed in this monograph.

The works of Andres and Biewer et al. in [101] and [102] are the most related to the approaches proposed in the thesis at hand. In fact, the problem encoding proposed by these two works builds the basis of the synthesis framework proposed in Chapter 3. Essentially, they suggest replacing the SAT-solver with an answer set programming (ASP) solver as it has been shown to decide routing options more efficiently. This is mainly justified by the varying assumptions of the solvers. While SAT assumes an open world, i.e., the truth value of each decision variable has to be explicitly assigned, ASP assumes a closed world. In the latter, decisions are assumed to be `FALSE` unless explicitly assigned to `TRUE`. Hence, reachability can be directly formulated in ASP. This allows a more effective decision-making of message routing when compared to SAT. To analyze timing constraints, they use separate ASP and QF–IDL-based SMT-solvers which communicate indicator variables through a shared text file. In combination with a homogeneous hardware architecture and unconstrained mapping options, the authors are able to exploit symmetries in the synthesis problem and use advanced heuristics which simplifies the solving accordingly. Again, an optimization step is not considered in their works.

To summarize, in the past decades, many approaches have been proposed to synthesis problems. Since the beginning of this century, the focus of the research was put on the electronic system level. Throughout these approaches, two main methodologies have prevailed. Heuristic approaches are well suited for finding diverse solutions, but tend to fail covering the entire design space. Initially finding regions with feasible solutions has been addressed by the integration of symbolic search techniques into the heuristics. Still, lacking a systematic search, design candidates are subject to be explored multiple times. Exact approaches aim at eliminating this problem. As the search is conducted purely automatically, no solutions are revisited twice. Furthermore, invalid regions can be safely pruned from the search through conflict analysis.

The DSE framework proposed in the thesis at hand extends the current state-of-the-art by merging the previously isolated problems of searching for Pareto-optimal design points and defining a feasibility preserving encoding. This is accomplished by the tight integration of background theories into the solving process of the foreground theory solver. Shared variables between the foreground and background theories thus allow exchanging information and advanced reasoning.

2.4.2 Archive Management

In the following, a number of methods are presented that have been developed to store and manage archives of non-dominated solutions. The most simple technique is the utilization of linear lists. Consequently, in the worst case, each novel solution is compared to all other solutions before it can be added to or rejected from the archive. The most significant advantage of linear lists is that solutions of the current archive that are dominated by a novel solution can be removed from it very efficiently in $\mathcal{O}(1)$ without the need for adjustment of other solutions (e.g., reordering). The main drawback is that the complexity w.r.t. the number of necessary dominance checks is always $\mathcal{O}(N)$ where N is the length of the list.

To overcome this shortcoming, Habenicht [105] first proposed to use Quad–Trees, a k-ary tree with k depending on the number of objective functions. In this data structure, solutions are stored according to an ordered policy that allows for faster dominance checks without the need to compare novel solutions against each existing element. However, deleting solutions from the tree, whenever they are dominated by a new element, is challenging. Therefore, Mostaghim and Teich [106] later introduced and compared three techniques to improve the performance of the Quad–Tree based archive management. As Quad–Trees are described thoroughly in Section 4.2, an extensive explanation is omitted here.

More recently, the authors of [107] proposed another approach for managing non-dominated solutions in MOEAs. The M–Front combines a list-based data structure with a k-dimensional binary search tree (k-d tree). Here, the solutions are stored into m sorted linked lists (M–List) and the k-d tree simultaneously where m is the number of objectives. The basic idea is to select one reference point for each novel solution and calculate a narrow area that contains a set of solutions for which the dominance checks have to be performed. To determine this area, they utilize geometric properties of the dominance relation to convert the problem into interval queries which can be answered using the M–List. An additional key requirement to minimize the number of solutions that must be checked and thus, the complexity of the algorithm is to find an appropriate reference point. In order to do so, the authors use a nearest neighbor search with the help of the k-d tree.

Jaszkiewicz and Lust [108] propose ND–Tree, a technique to update Pareto archives online. In the tree-based structure, each node represents a subset of solutions that are contained in a hypercube defined by its local nadir and optimal points (point-wise best/worst of all objective functions). The actual solutions are stored in lists in the leaf nodes whereas internal nodes represent a union of all of their children storing the accumulated optimal and nadir points. That is, traversing the tree from the root to a leaf isolates potentially dominated candidates without the need to check each solution individually. Whenever a new solution is, for example, dominated by the combined nadir point, it can be discarded. If it dominates the ideal point, the whole subtree is automatically dominated and can hence be deleted.

In the thesis at hand, the problem of efficiently managing the Pareto archives is addressed

when solving multi-objective combinational problems (MOCOP) (cf. [109] for an overview) based on CDCL and partial assignment checking. All previously mentioned methods have in common that they were developed to manage the archive for population-based meta-heuristics and do not work directly for CDCL-based solving techniques with partial assignment checking. M–Front has the drawback that for every novel solution an appropriate reference point has to be calculated. That is, this calculation has to be executed for each partial assignment which would ultimately result in a high complexity.

In recent years, there have been few works that deal with deterministic methods for solving MOCOPs. The authors of [110] propose a method to solve such problems. However, they do not consider the ability of partial assignments. Though mentioning the possibility to utilize Quad–Trees for managing the archive as an outlook for future work, they only use linear lists to archive found non-dominated solutions as their focus lies on the solving process itself.

"asprin" (*ASP* for *pr*eference handl*ing*) [74] is a framework for defining and computing preferred (optimal) solutions among stable models of logic programs. It offers a wide spectrum of different predefined preference types such as cardinality minimization as well as composite (i.e., multi-objective) preference types such as lexicographic and Pareto optimizations. With "asprin", a MOCOP can be solved using the stable model semantics. However, no archive is used to store non-dominated solutions. Rather, constraints are added that exclude found and dominated solutions from the search space.

2.4.3 Approximation

Besides the classification of DSE approaches in [91] as discussed above, in this section, the focus shall be put on the use of approximations in DSE.

In general, there is no DSE approach that guarantees obtaining the true Pareto front of highly complex embedded systems in a viable amount of time. Hence, each approach tries to find a set of implementations that approximates the true Pareto set as good as possible. It is important to state that the term "approximation" is not used consistently throughout literature. Two different interpretations can be identified: search- and evaluation-related approximations.

In the former, the central aim is to steer the search into regions of the parameter space where they expect Pareto optimal solutions. Representatives of works in these search-related approximations are [95, 111–117]. ReSPIR [111] and MULTICUBE [112] use the technique Design of Experiment (DoE) to choose appropriate sample configurations for simulating a multi processor SoC. The results of the simulations are then used to create an approximation of a Pareto front with response surface modeling (RSM). From this RSM, new configurations for simulations are derived to incrementally enhance the approximation of the Pareto front. In contrary, the approach proposed by the thesis at hand approximates the actual result to avoid costly simulations. Thereby, the DoE technique is utilized as well. That is, the approach is able to efficiently eliminate bad design candidates. However, the candidate choice is not steered by the shape of the current Pareto front approximation.

The authors of [113] and [114] present comparable approaches by utilizing stochastic kriging, a stochastic meta-model simulation, to select potentially good regions of the design space to search for new solutions. Initially, they sample the initial points using a maximin latin hypercube sample. These designs are then evaluated and used to fit the parameters of a stochastic kriging model. Afterwards, the kriging model is used to determine a potentially good region in the search space from which a new candidate solution is selected. After evaluating

this point, the accuracy of the meta-model is analyzed and updated if necessary. The results show that it finds up to 91% of all true Pareto points of common multi-objective optimization problems. Instead of stochastic simulations, the authors of [117] propose the use of generic and easily computable guiding functions that steer the search into promising regions of the design space. Compared to a reference approach running over a period of two weeks, they obtain a 20% better Pareto front approximation within four days. In [115], the search time is reduced by using a heuristic that prunes the search of potentially inferior regions. Thus, expensive evaluations can be reduced to a minimum. With their heuristic, the authors can achieve a speedup of up to 80 while simultaneously maintaining the quality of Pareto front approximations compared to a state-of-the-art MOEA. Liu et al. [95] present an approximation technique that is based on a compositional approach. The approach is tailored towards a component-based design, where most parts of the system are reused from legacy designs or IP libraries. Their algorithm only explores each subsystem once and, with that, is capable of finding good Pareto front approximations for the composed system. Compared to an exhaustive search, their results show a performance gain of 52% up to 87% and an error rate between 1.5% and 4.7%.

The approach proposed in the thesis at hand belongs to the second group which can be characterized as evaluation-related approximations. The DSE is accelerated by approximating the objective function calculations that are performed to obtain the quality of found solutions. The authors of [118] and [119] propose approximation techniques to accelerate the calculation of performance indicators and power consumption of multiprocessor systems-on-chip, respectively. Although the corresponding results show that their approximations reach error rates of less than 18% for performance and only 9% for power consumption, both works do not integrate their approximations into a DSE. The combination of objective approximations with a DSE are proposed for example in [13, 120–123]. The three works [122], [121], and [123] propose to combine the use of inexpensive approximations and accurate (but costly) simulations. In [122], initially, exact evaluations are performed that are used to train an estimator. After the training phase is finished, the estimator is then used instead of the exact simulation to save evaluation time. Only promising solutions are still evaluated exactly. The authors of [123], on contrary, save exploration time by statically evaluating only a given percentage of all designs exactly. Thus, the quality of the remaining designs is only approximated. In both [122] and [123], the exploration is partly conducted with approximated evaluations. As will be shown in Section 4.3, this can lead to incomplete and incorrect archives where optimal designs might be missing or non-optimal designs are included, respectively. The MILAN framework by Mohanty et al. [120] and the approach of Herrera [124] are, on first sight, similar to the approach proposed by the thesis at hand. The authors of [120] propose a hierarchical design space optimization where, in each phase, the evaluation accuracy is increased to gradually remove potentially inferior designs. The difference to the approach presented in this monograph is twofold. First, MILAN relies on the manual selection of designs after each phase, whereas the proposed methodology works fully automated. Second, the design points explored by MILAN are not guaranteed to be optimal. Herrera [124] also uses an analytical approximation filtering valid solutions. Subsequently, an exact simulation on each of these safe solutions is performed to find the optimal ones. However, if a solution is erroneously not considered to be safe by the approximation, it will not be found by this approach. Furthermore, postponing the exact simulation (after all safe solutions have been determined) prevents pruning the design space early. RAPIDITAS [125] steers the search for good solutions iteratively, where, in each iteration, the number of available processors in the target platform is reduced. At first, the application of n tasks is mapped to n processors

and simulated. In the next step, all mappings that contain $(n-1)$ processors are explored and evaluated using approximations. The best design is chosen and once again simulated. This is repeated until no solution with fewer processors can be found. As they only have to evaluate a relatively small number of solutions, they can reduce the DSE time by 72%. Compared to the proposed approach, RAPIDITAS does not guarantee completeness of the non-dominated front as only one design per processor count is considered. If, for example, two designs with n processors would be Pareto optimal, only one of them would be represented in the final non-dominated front. Furthermore, RAPIDITAS assumes homogeneous platform models whereas this thesis generally models heterogeneous computing resources. The work presented in [126] uses individual application profiling to accelerate the DSE. The key point of this approach is the extraction of execution traces during the DSE and shifting the actual mapping decisions to the runtime of the system. The execution traces approximate the time needed for individual applications assuming each task is executed on one distinct processor. At runtime, when the active applications are known, a platform manager is invoked that composes the mappings of the individual execution traces such that the resulting throughput of the application mix is minimal. This can significantly accelerate the exploration at design time. Again, this work considers homogeneous architectures. The most significant difference to the proposed approach is, however, the relaying of mapping decisions into the runtime of the system. Instead, the scope of this thesis is to provide a set of Pareto-optimal solutions at design time from which good compromise designs can be selected for subsequent steps in the development process of the overall digital systems.

To the best of the author's knowledge, the earliest work in this area has been conducted by Abraham et al. [13]. With the help of bounded approximations, they show that the true Pareto front can be obtained. The approximation functions must not exceed a user-defined error threshold Δ. They define a bounded Pareto dominance relation that filters all designs that lie beyond a $2{\cdot}\Delta$ threshold. Exact evaluations have to be performed only for the remaining designs. Note that this process fails if at least one design approximation is outside the threshold Δ. For real world objective functions, this requirement is (in general) not feasible as the estimation is highly dependent on design decisions made.

In the author's opinion, the approach presented in this monograph cannot compete directly with the aforementioned works as the scope differs too much. Although many of them aim at obtaining Pareto fronts as a result of a DSE, they do not guarantee completeness and correctness even if the entire search space is explored. Furthermore, different boundary conditions such as homogeneous hardware platforms (e.g., [125]), runtime decisions (e.g., [126]), manual refinement steps (e.g., [120]), or a different approximation scope (e.g., [111, 115, 117]) make a direct comparison of exploration time and quality of the results meaningless. Even a direct comparison with the early work of Abraham et al. [13] would be misleading in the author's opinion as the constraints put on the approximation function are hard to fulfill in real use cases. The approach proposed in this thesis shall not be understood as a direct competition. Instead, it is considered as orthogonal work that can be used in other frameworks with little overhead. For instance, the approximation approach may be included in the evaluation step of evolutionary algorithms (e.g., [91]) where many evaluations have to be carried out in each iteration.

2.4.4 Test Case Generation

While the research on DSE has lead to variant approaches for the exploration itself, systematic methods to generate test instances are sparse. There exist two graph generation tools [127, 128] that have been proposed to be used as standard tools for the generation of random task graphs and synchronous dataflow graphs (SDFGs), respectively. TGFF [127] was especially designed to serve as a reference system for generating random task graphs for mapping and scheduling purposes. It constructs task graphs on the basis of two different algorithms, that can be configured with a number of parameters. While the first algorithm specifies the properties of the graph by the maximum *In/Out Degree* of nodes, the second algorithm generates series-parallel task graphs recursively with the option of additional edges between parallel units.

The authors of [128] proposed SDF3, a tool to generate, analyze and visualize SDFGs. The generated graphs are connected, consistent, and deadlock free with user defined characteristics. Furthermore, the tool consists of (user expendable) functions that are able to assign specific properties to actors and channels of the graph.

In contrast to the approach in the thesis at hand, both SDF3 and TGFF, however, only generate the application and do not consider the hardware architecture or mapping possibilities. While the task graph model considered in the present thesis is similar to TGFF, SDF3 is especially orientated towards the data flow model of computation that additionally allows the generation of cyclic applications. Therefore, a holistic approach is needed that is able to concurrently generate the application, the hardware architecture and assign mapping possibilities. Section 3.4.1 proposes an approach that generates complete problem instances. These can be used as input for any DSE methodology that can read text-based input files. Note that an ad-hoc random approach is not desirable as it may generate problem instances that are not solvable due to mapping (and scheduling) constraints. Furthermore, ad-hoc approaches have little to no influence on the solving complexity of generated test instances.

3
System Synthesis with Partial Assignment Evaluation

An automated synthesis approach at the electronic system level (ESL) should allow for integrated constraint checks and decision evaluations. The formal modeling techniques reviewed in Section 2.2 of the previous chapter only partially correspond to these requirements. Although the encoding with answer set programming (ASP) does naturally permit defining feasibility constrains and testing for linear validity constraints, it does not present an effective approach for dealing with non-linear constraints. Especially scheduling constraints cannot be modeled and integrated into the reasoning process. Instead, earlier approaches used external tools to evaluate the validity of a solution candidate. This leads to isolated, merely loosely coupled, solving processes that cannot share internal state variables for sophisticated pruning techniques. To overcome these shortcomings, in this chapter, a novel, holistic approach to the synthesis problem at ESL is elaborated. The aim of this chapter is therefore defining an ESL synthesis approach that especially permits:

(a) the encoding of feasibility and (non-linear) validity constraints,

(b) the exchange of state variables for propagating decisions,

(c) a sophisticated pruning to skip infeasible regions of the search space.

The key concepts proposed in this chapter are utilized to holistically encode the specification model originally presented by Blickle et al. [48]. This includes the application description, a hardware template, as well as allocation, binding, routing, and scheduling constraints. To this end, the ASP-based encoding is composed of three key components. First, the ASP encoding itself that contains facts composing a specific problem instance and rules constituting the allocation, binding, and routing constraints. Second, the scheduling constraints that are encoded through theory atoms. While they are transparent to grounding, these theory atoms represent couple variables whose internal value (i.e., the task start times) are not determined by the ASP solver. When contained in a stable model, the theory atoms are instead checked by the third component, a tightly integrated theory solver. This theory solver makes use of the ASP modulo theories (ASPmT) paradigm and allows for the evaluation of partial assignments as well as the theory propagation. Hence, invalid regions are detected early in the decision process of the ASP solver. In conclusion, the combination of the three components enables a single encoding of the complete specification that allows for a succinct and elaboration-tolerant formulation.

The remainder of this chapter is organized as follows. In Section 3.1, the specification model that is utilized throughout this monograph is detailed. Therefore, the originally proposed model

by Blickle et al. [48] is reviewed and extended to support periodic and cyclic applications. Based on the model, the instance encoding will be detailed at the end of that section.

The specification model is followed by the constraints encoding corresponding to the ASPmT paradigm in Section 3.2. This section is separated into the encoding of allocation, binding, and routing constraints and scheduling constraints in the form of theory atoms. While the former are encoded via classical ASP rules, the latter require the definition of a specialized grammar. The background theory evaluating the satisfiability of the theory atoms is elaborated in Section 3.2.2. It utilizes a specialized quantifier-free integer difference logic (QF–IDL) solver that is tightly integrated into the theory propagation framework of the underlying ASP solver. The tight integration allows for an effective way to share information and propagate as well as evaluate partial decisions made during the search.

Section 3.3 presents the final synthesis framework, where the integration of the background theory solver is elaborated. That is, the connection between foreground ASP and background QF–IDL solvers is detailed. Particularly, this includes the theory propagation techniques used to implement an effective reasoning when utilizing partial assignments.

Based on a variety of test instances, the advantage of partial assignment evaluation over complete assignments is experimentally evaluated in Section 3.4. Currently, however, available benchmarks do not provide for a systematic method for generating such instances. To this end, in the first part of the evaluation, an ASP-based instance generator is proposed. This allows to define common properties of the generated problem instances. In particular, it provides a methodology to generate synthetic specifications with a specified degree of parallelism in the application graph. The generator is constructed modularly. Hence, it allows the addition or replacement of parts without altering the remaining parts. In its default configuration, it generates applications as series parallel graphs (SPGs) and a regular mesh-based hardware platform implementing a 3-dimensional network on chip (NoC). The properties of the individual elements are generated randomly to guarantee a diverse complexity of test instances. In the second half of the evaluation, the generator is utilized to generate the instances for evaluating the proposed synthesis framework.

Section 3.5 concludes this chapter and provides a summary of the proposed framework.

Relevant Publications

[2] Kai Neubauer, Philipp Wanko, Torsten Schaub, and Christian Haubelt. “Enhancing Symbolic System Synthesis through ASPmT with Partial Assignment Evaluation”. In: *Design, Automation and Test in Europe Conference (DATE)*. Lausanne, Switzerland, Mar. 2017, pp. 306–309. DOI: 10.23919/DATE.2017.7927005.

[5] Kai Neubauer, Christian Haubelt, Philipp Wanko, and Torsten Schaub. “Systematic Test Case Instance Generation for the Assessment of System-level Design Space Exploration Approaches”. In: *21. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*. Tübingen, Germany, Mar. 2018. DOI: 10.15496/publikation-25685.

3.1 System Model

As reviewed in Section 2.1.2, the system can be specified through a language- or model-based approach. In this work, a mathematical model with strict formal semantics is chosen to allow for a high analyzability. The abstract graph-theoretic modeling approach presented in the following is based on the work of [48]. In accordance with the X-Chart diagram (cf. Section 2.1.3, Figure 2.2), the model is composed of a behavioral system description and a definition of non-functional constraints. In summary, it supports the specification of deadline-constrained, periodic and cyclic streaming applications that are mapped onto heterogeneous hardware architectures.

Model of Computation The behavioral description of the system is considered as a set A of applications A_i in task-level granularity and is modeled as an application graph as follows:

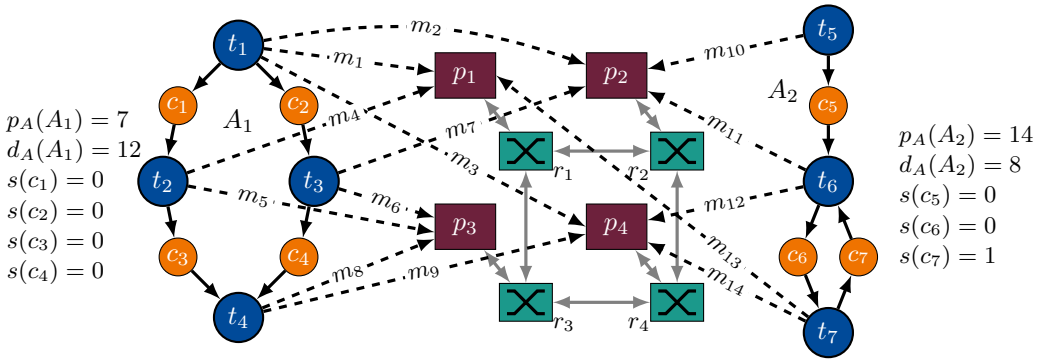
Definition 3.1.1 (Application Graph): An application graph $\mathcal{G}_A$ is defined as a triple $\mathcal{G}_A = (V_A, E_A, \mathcal{F}_A)$ where $V_A = T \cup C$ represents the set of vertices composed of tasks $t \in T$ and messages $c \in C$, $E_A \subseteq T \times C \cup C \times T$ represent the dependencies between tasks $t \in T$ and messages $c \in C$ and vice versa, and $\mathcal{F}_A$ is a set of functions to assign properties to the elements of the graph.

The set of property functions $\mathcal{F}$ may contain an arbitrary number of functions, depending on the specific properties to be evaluated. In the present chapter, the functions p_A, d_A, s are considered as they are necessary to evaluate the system regarding its performance. In detail, $p_A : T \to \mathbb{N}$, $d_A : T \to \mathbb{N}$, and $s : E_A \to \mathbb{N}_0$ are functions that assign a periodicity and deadline to each task $t \in T$, and an index delay to each message $c \in C$, respectively. While the periodicity of a task indicates the time after which a task repeats its execution in the subsequent iteration, the deadline defines the latest possible time when the corresponding task has to finish its execution in each iteration relative to its release time. The index delay, specified for messages, defines the number of iterations after which a message has to arrive at the receiving task. Thus, a message c sent by t_{tx} in iteration n through the edge (t_{tx}, c) has to arrive at task t_{rx} in iteration $n + s(c)$ through the edge (c, t_{rx}).

In order to obtain a valid application graph, it has to adhere to various requirements. To this end, consider the predecessor and successor functions $pred : V_A \to 2^{V_A}$ and $succ : V_A \to 2^{V_A}$ with their definitions $pred(v_i) = \{v_j \mid (v_j, v_i) \in E_A\}$ and $succ(v_i) = \{v_j \mid (v_i, v_j) \in E_A\}$. Each message c is required to have exactly one predecessor and one successor, i.e., one sending task $t_{tx} \in T$ and one receiving task $t_{rx} \in T$ ($|pred(c)| = |succ(c)| = 1$). On contrary, a task t may have an arbitrary number of predecessors or successors. Thus, inter process communication is characterized by point-to-point connections and multicast communication has to be modeled through individual messages. Furthermore, a valid application graph must only contain cyclic dependencies forming a closed trail $w = \langle t_i, c_j, \ldots, c_k, t_i \rangle$ such that the sum of the index delays along the trail is greater than zero:

$$\forall w \in \{\langle t_i, c_j, \ldots, c_k, t_i \rangle \mid (t_i, c_j), \ldots, (c_k, t_i) \in E_A\} : \sum_{c \in w} s(c) > 1.$$

Otherwise, the task t_i, concurrently acting as sender and receiver, would be waiting for its own transmission resulting in a contradicting deadlock. The final requirements concern connected subgraphs of $\mathcal{G}_A$ that represent individual applications A_i. Each application A_i consists of an exclusive subset of all vertices $V_i = T_i \cup C_i \subseteq V_A$ that are connected through directed edges



$$w : \{m_1 \mapsto 3, m_2 \mapsto 1, m_3 \mapsto 2, m_4 \mapsto 2, m_5 \mapsto 2, m_6 \mapsto 2, m_7 \mapsto 2, m_8 \mapsto 2, m_9 \mapsto 2, m_{10} \mapsto 3, m_{11} \mapsto 1, m_{12} \mapsto 2, m_{13} \mapsto 2, m_{14} \mapsto 2\}$$

Figure 3.1: Specification Graph based on the work of Blickle et al. [48]

$E_i \subseteq E_A$. More precisely, each vertex $v \in V_A$ and edge $e \in E_A$ is contained in exactly one application A_i and the predecessors and successors of each vertex $v_j \in V_i$ are required to be located in the same application A_i:

$$\bigcap_i V_i = \emptyset \wedge \bigcup_i V_i = V_A \wedge \bigcap_i E_i = \emptyset \wedge \bigcup_i E_i = E_A \wedge \forall A_i, v \in V_i : (pred(v) \subseteq V_i \wedge succ(v) \subseteq V_i).$$

In other words, there exists no trail containing tasks of two or more applications. Within an application A_i, the periodicity of the tasks $t \in T_i$ has to be identical, i.e., $\forall t_k, t_l \in T_i : p_A(t_k) = p_A(t_l)$, to avoid deadlocks in later iterations. Finally, the deadlines assigned to the tasks of an application A_i are generally not constrained. However, two aspects should be considered. First, a deadline should not be smaller than the task execution time (see below) as the problem would be directly unsatisfiable. Second, considering an open trail $\langle t_i, \ldots, t_j \rangle$ through the graph of application A_i, the deadline of t_j should be larger than the accumulated execution times of the preceding tasks.

Apart from the direct dependencies of the vertices V_A imposed by the edges E_A, the transitive hull of the graph $\mathcal{G}_A$ forms two kinds of indirect dependencies between the vertices.

Definition 3.1.2 (Inter-iteration Dependency): Two tasks or messages $v_i, v_j \in V_A$ are considered inter-iteration dependent (denoted by $v_i \leftrightarrow v_j$) of each other if a trail $\langle v_i, \ldots, v_j \rangle$ or $\langle v_j, \ldots, v_i \rangle$ through the graph $\mathcal{G}_A$ exists. Whenever no such trail exists, the two vertices are considered independent of each other, i.e., $v_i \nleftrightarrow v_j$.

Definition 3.1.3 (Intra-iteration Dependency): Two tasks or messages $v_i, v_j \in V_A$ are considered intra-iteration dependent (denoted by $v_i \overset{*}{\leftrightarrow} v_j$) of each other if a trail $\langle v_i, \ldots, v_j \rangle$ or $\langle v_j, \ldots, v_i \rangle$ through the graph $\mathcal{G}_A$ exists and the sum of index delays of all messages along the trail equals to zero. Otherwise, they are intra-iteration independent of each other, i.e., $v_i \overset{*}{\nleftrightarrow} v_j$.

An example of an application graph is depicted in Figure 3.1. The application graph $\mathcal{G}_A$ consists of two independent applications A_1 with $T_1 = \{t_1, t_2, t_3, t_4\}$, $C_1 = \{c_1, c_2, c_3, c_4\}$, and $E_1 = \{(t_1, c_1), \ldots, (c_4, t_4)\}$ and A_2 with $T_2 = \{t_5, t_6, t_7\}$, $C_2 = \{c_5, c_6, c_7\}$, and $E_2 = \{(t_5, c_5), \ldots, (c_6, t_7)\}$. The periodicity and deadlines of the tasks are constant for each application, i.e., $\forall t \in T_1 : p_A(t) = 7, d_A(t) = 12$ and $\forall t \in T_2 : p_A(t) = 14, d_A(t) = 8$. While the index

```
1  %%% Application
2  app(a1).
3  task(t1,a1). task(t2,a1). task(t3,a1). task(t4,a1).
4  message(c1,a1). message(c2,a1). message(c3,a1). message(c4,a1).
5  send(t1,c1). send(t1,c2). send(t2,c3). send(t3,c4).
6  receive(t2,c1). receive(t3,c2). receive(t4,c3). receive(t4,c4).
7  period(a1,7).
8  period(T,P) :- task(T,A), period(A,P).
9  deadline(a1,12).
10 deadline(T,D) :- task(T,A), deadline(A,P).
11 delay(c1,0). delay(c2,0). delay(c3,0). delay(c4,0).
```

Figure 3.2: Encoding of application A_1

delay of most messages $c \in C$ is equal to 0, there is one exception in application A_2. As defined above, a closed trail through a graph requires the sum of index delays along the path to be larger than 0 to avoid deadlocks. Hence, the index delay of message c_7 equals to one which indicates an inter-iteration communication among t_6 and t_7. In order to encode the application graph in ASP, the graph is translated into a set of rules. An example encoding of application A_1 of Figure 3.1 is listed in Figure 3.2. Independent applications are encoded through the unary predicate `app/1` (line 2). In the subsequent lines 3 and 4, the tasks and messages are encoded through `task/2` and `message/2`, respectively. While the first argument represents the unique identifier, the second argument matches the corresponding application. The edges E_A are separated into the two subsets $T \times C$ and $C \times T$, encoded by `send/2` and `receive/2`. Finally, the properties of the individual elements are defined. Here, the periodicity and deadline is encoded per application (cf., line 7 and 9). Subsequently, the periodicity of each task is defined by the rule in line 8. During grounding, the periodicity of each task is inferred by the corresponding periodicity of the application it belongs to. The index delay of the messages is encoded by `delay/2` as shown in line 11.

Model of Architecture Similar to the behavioral description, the hardware architecture is modeled as a directed platform graph.

Definition 3.1.4 (Architecture Graph): An architecture, or platform graph $\mathcal{G}_P$ is defined as a triple $\mathcal{G}_P = (V_P, E_P, \mathcal{F}_P)$, where $V_P = P \cup R$ represents the union of all computational processing elements P and communication elements R, $E_P \subseteq V_P \times V_P$ represents the communication links connecting the vertices, and $\mathcal{F}_P$ is a set of functions to assign properties to the elements of the graph.

In contrast to the application graph, only one function $d_P : R \to \mathbb{N}$ is considered to be included in the set $\mathcal{F}_P$ in the present chapter. It assigns a routing delay to each link $l = (v_i, v_j) \in E_P$, i.e., the time necessary to route a message from the input of v_i to the input of v_j via l. The communication resources are not constrained regarding their internal behavior through the system model but instead by the specific problem encoding presented in the next section. For example, the interpretation as on-chip routers or busses is possible. To act as routing devices, similar to [102], each communication resource r is assumed to internally consist of independent crossbar switches and corresponding input buffers to allow concurrent routing of messages. That is, two (or more) messages may be routed simultaneously over one resource if

```
%%% Hardware Platform
processor(p1). processor(p2). processor(p3). processor(p4).
router(r1). router(r2). router(r3). router(r4).
link(l1,p1,r1). link(l2,r1,p1). link(l3,r1,r2). link(l4,r2,r1). (...)
routingdelay(l1,2). routingdelay(l2,2). routingdelay(l3,2). (...)
```

Figure 3.3: Encoding of the hardware platform shown in Figure 3.1

they have different destinations. Busses, on the other hand, only allow for one communication at a given time step and messages have to be scheduled accordingly.

In the center of Figure 3.1, an example hardware platform is depicted. It consists of four processing elements $p_1, \ldots, p_4$ and four communication resources $r_1, \ldots, r_4$. The communication resources in this example act as routers with up to five ports, each connecting to one processing element and the neighboring routers. Note that the links depicted in the graph represent two directed edges. The routing delay is two time units for each link, i.e., $\forall l \in E_P : d_P(l) = 2$. Thus, sending a message from p_1 to p_2 via the route $\langle p_1, r_1, r_2, p_2 \rangle$ requires six time units.

The encoding of the platform graph through ASP facts is similar to the application and shown in Figure 3.3. The unary predicates `processor/1` and `router/1` encode the processing and communication elements, respectively. Communication links are established through `link/3` atoms where the arguments indicate the ID, the source, and sink of the link. Finally, the routing delay property is specified via `routingdelay/2`.

Specification Model To aggregate the models of computation and architecture, the specification graph is defined.

Definition 3.1.5 (Specification Graph): A specification graph is defined as a quadruple $\mathcal{G}_S = (\mathcal{G}_A, \mathcal{G}_P, M, \mathcal{F}_S)$, where $\mathcal{G}_A$ and $\mathcal{G}_P$ represent an application and platform graph, respectively, $M \subseteq T \times P$ is a set of options to map a task $t \in T$ onto a processing element $p \in P$, and $\mathcal{F}_S$ is a set of functions assigning properties to the former three elements.

The mapping options M connect the application and the platform graphs. To this end, a single mapping option $m = (t, p) \in M$ indicates that task t is able to execute on processing element p[3-1]. The function $w : M \to \mathbb{N} \in \mathcal{F}_S$ assigns a worst case execution time (WCET) to each mapping option. Specifying multiple mapping options $m_i = (t_j, p_k)$ to a single task t_j with their corresponding WCETs permits modeling heterogeneous architectures[3-2]. As an example, consider the mapping edges in Figure 3.1, represented by dashed edges. For task t_1, three mapping options $m_1 = (t_1, p_1)$, $m_2 = (t_1, p_2)$, and $m_3 = (t_1, p_4)$ are specified that allow for the execution on the processing elements p_1, p_2, and p_4, respectively. The function w assigns different WCETs to each of them, i.e., $w(m_1) = 3$, $w(m_2) = 1$, and $w(m_3) = 2$. Hence, the execution time of the task t_1 depends on the decision which mapping option is chosen. Furthermore, different mapping options can be assigned to the pair (t, p) to model, for example, different compiler or logic synthesis options.

The ASP encoding of the mapping options M is given in Figure 3.4. Each mapping option is

[3-1] Similar to the mapping options of the tasks, the routing of messages could be constrained to specific links. For the sake of brevity, no such constraints are employed by the model at hand nor studied in the experiments.

[3-2] Note that, apart from scheduling constraints, heterogeneity can also be modeled by assigning different properties (e.g. area and power consumption) to the elements $v \in V_P$, see Chapter 4.

```
17 %%% Mapping Options
18 map(m1,t1,p1). map(m2,t1,p2). map(m3,t1,p4). map(m4,t2,p1). (...)
19 wcet(m1,3). wcet(m2,1). wcet(m3,2). (...)
```

Figure 3.4: Encoding of the mapping options specified in Figure 3.1

encoded through a ternary predicate `map/3` representing a unique identifier, the corresponding task, and a processing element. The WCET is defined by `wcet/2` containing the identifier of the mapping option as the first and the execution time as the second argument.

3.2 Synthesis Encoding

The aim of the system synthesis is the transformation of a specification into an implementation of a system. In this section, an automated approach for solving the synthesis problem based on the ASP paradigm is proposed. As is common in ASP, the problem encoding is split into two parts. First, the section presents an encoding of the graph-theoretic specification model representing the specific problem instance. Due to the first-order input language of ASP, the specification graph translates seamlessly into a set of ASP facts. Afterwards, the section elaborates on the general problem definition of system synthesis, i.e., the allocation, binding, routing, and scheduling. Although possible in theory, the encoding of non-linear constraints through regular ASP atoms is usually not viable in practice. Hence, the encoding of such constraints is conducted via theory atoms. In particular, the scheduling substep is encoded as QF–IDL terms and solved by a specialized, yet fully integrated QF–IDL solver. Compared to separate solvers, the integration offers the advantage of partial assignment checking. That is, partial binding and routing decisions leading to unsatisfiable scheduling constraints can be directly detected which allows learning infeasible regions of the design space early in solving process. This is enabled by the utilization of the ASPmT paradigm embedded into the ASP solver clingo 5 [42]. In summary, the framework presented in the following allows for a unified encoding of the synthesis problem resulting in a succinct and elaboration tolerant formulation.

3.2.1 Encoding Allocation, Binding and Routing

As reviewed in Section 2.1.3, an implementation $\mathcal{I} = (\alpha, \beta, \gamma, \tau)$ of a system consists of four parts – an allocation α, a binding β, a routing γ, and a schedule τ. In short, an allocation $\alpha \subseteq V_P$ selects computation and communication resources, a binding $\beta \subseteq M$ assigns tasks to computation resources, the routing $\gamma \subseteq C \times 2^{E_P}$ assigns messages to communication elements, and a schedule τ determines the temporal execution order of the tasks and messages.

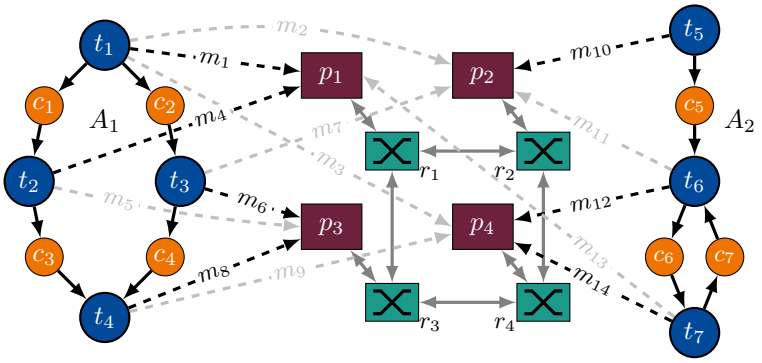
The constraints imposed by the binding are twofold. First, each task must be assigned to exactly one processing element and, second, a task must not be bound to an incompatible processing element. The ASP encoding of binding and routing constraints is based on the work conducted by Andres et al. [129] and shown in Figure 3.5. The choice rule in line 2 states that for each atom `task/1`, at least and at most (i.e., exactly) one `bind/3` must be selected. Therefore, the grounder generates a set of `bind/3` atoms for each `map/3` atom containing the corresponding task T in the inner part of the braces. Assume task t_1 and the corresponding atoms `task(t1)`, `map(m1,t1,p1)`, `map(m2,t1,p2)` and `map(m3,t1,p4)` of the running specification example, cf. Figure 3.1. Grounding the rule in line 2 produces the variable free representation

```
1  %%% Binding
2  1 {bind(M,T,R) : map(M,T,R)} 1 :- task(T).
3  %%% Routing
4  root(C,R) :- send(T,C), bind(_,T,R).
5  % resources of communication task per target
6  sink(C,R) :- read(T,C), bind(_,T,R).
7  % backward hops of communication task
8  1 { reached(C,L,R,S) : link(L,R,S) } 1 :- sink(C,S), not root(C,S).
9  sink(C,R) :- sink(C,S), reached(C,L,R,S).
10 % reach communication root resource
11 :- read(T,C), root(C,R), not sink(C,R).
12 %%% Allocation
13 allocate(R) :- bind(_,_,R).
14 allocate(R) :- sink(_,R).
```

Figure 3.5: ASP encoding of allocation, binding, and routing decisions

`1 {bind(m1,t1,p1), bind(m2,t1,p2), bind(m3,t1,p4)} 1`. An analogue transformation is executed for each task of the specification. Note that the grounder removes the body as it is not necessary for a semantically equivalent representation. The routing substep depends on the binding decisions of dependent tasks, indicated by $(t_{tx}, c), (c, t_{rx}) \in E_A$. If the two tasks are bound onto the same processing element i.e., $(t_{tx}, p), (t_{rx}, p) \in \beta$, the dependency can be resolved locally and data produced by t_{tx} can directly be used by t_{rx}. Hence, a communication through the communication infrastructure is not necessary. However, whenever the two tasks are not bound to the same processing element, i.e., $(t_{tx}, p_i), (t_{rx}, p_j) \in \beta \mid p_i \neq p_j$, a route between the corresponding processing elements has to be established for the message c. The function $\pi : C \times \gamma \to 2^{E_P}$ represents the route of a message $c \in C$ and yields an ordered list of links that consists of an arbitrary number of hops. The function $h : 2^{E_P} \times \mathbb{N} \to E_P$ returns the n-th hop of a given route indicated by the corresponding link. Note that a route $\pi(c, \gamma)$ (short π_c) may be empty when both sending and receiving task are bound to same resource, i.e., $\pi_c = \emptyset \Leftrightarrow \{(t_i, c), (c, t_j)\} \subseteq E_A \wedge \{(t_i, p), (t_j, p)\} \subseteq \beta$. In the following, a hop is denoted by $h_{c,i}$ as a shorthand for $h(\pi_c, i)$. Formally, each route corresponds to an open trail through the platform graph. Hence, start and end vertices are distinct and edges must not repeat. The ASP encoding is given in lines 3 – 11. In short, it determines a trail from the processing element of the sending task to the processing element of the receiving task of a message by recursively traversing the graph from the end to the start. In detail, the encoding consists of five individual rules. The rule in line 4 infers the root node of the route by assessing the current binding of the sending task. In line 6, the final destination is determined similarly, depending on the current binding of the receiving task. The main traversal is encoded in lines 8 and 9. If a resource `S` is already considered a sink by `sink(C,S)` and `S` is not equivalent to the root node, exactly one link is chosen that contains `S` as destination. The subsequent line defines the newly connected node as a further sink. This is repeated until no more `sink/2` atoms can be inferred. Finally, the integrity constraint in line 11 guarantees that the root node is reached. Note that a closed trail is inherently prevented by the combination of the choice rule in line 8 and the integrity constraint in line 11. While the choice rule guarantees that only one outgoing link is selected for each sink, the integrity constraint enforces that the root is reached.

For the running example, assume that the binding atoms `bind(m1,t1,p1)`, `bind(m4,t2,p1)`, and `bind(m6,t3,p3)` are selected. Hence, the messages c_1 and c_2 shall be routed. The root



$\pi_{c_1} = \langle\rangle, \pi_{c_2} = \langle(p_1, r_1), (r_1, r_3), (r_3, p_3)\rangle, \pi_{c_3} = \langle(p_1, r_1), (r_1, r_3), (r_3, p_3)\rangle, \pi_{c_4} = \langle\rangle,$
$\pi_{c_5} = \langle(p_2, r_2), (r_2, r_4), (r_4, p_4)\rangle, \pi_{c_6} = \langle\rangle, \pi_{c_7} = \langle\rangle$

Figure 3.6: A feasible binding and routing of the Specification Graph in Figure 3.1

of both messages is determined to be processing element p_1, i.e., the atoms `root(c1,p1)` and `root(c2,p1)` are inferred by the rule in line 4. The final sink of message c_1 is analogously determined as processing element p_1 inferring the atom `sink(c1,p1)`. As both `sink(c1,p1)` and `root(c1,p1)` are inferred, the head of the choice rule in line 8 does not infer any `reached/4` atom. Furthermore, the integrity constraint in line 11 is satisfied. Hence, the route $\pi_{c_1} = \emptyset$ of the message c_1 is empty because the sending and receiving task are bound to the same processing element. In contrast, the route π_{c_2} of message c_2 must contain multiple hops as its corresponding tasks are bound to different resources. The rule in line 6 infers the atom `sink(c2,p3)`. Subsequently, the first hop is selected using the choice rule. As there is only one link containing p_3 as destination, the corresponding link is selected and the atoms `reached(c2,l12,r3,p3)` and `sink(c2,r3)` are inferred. Subsequently, the next hop is determined. Here, the ASP solver has to choose between the links (r_1, r_3) and (r_4, r_3). For the sake of brevity, assume the former is chosen leading to the atoms `reached(c2,l5,r1,r3)` and `sink(c2,r1)`. In order to reach the root node and satisfy the integrity constraint, in the final step, the link $(p1, r1)$ is selected and the corresponding atoms `reached(c2,l1,p1,r1)` as well as `sink(c2,p1)` are inferred. Now, the integrity constraint in line 11 is satisfied and the route is complete: $\pi_{c_2} = \langle(p_1, r_1), (r_1, r_3), (r_3, p_3)\rangle$.

The allocation α is determined by the last two lines in Figure 3.5. In this encoding, only necessary resources are allocated, i.e., each processing element that is used in at least one binding and each communication element that is contained in at least on route. The remaining resources are assumed to be irrelevant. With the binding and routing decisions made in the running example, the atoms `allocated(p1)`, `allocated(r1)`, `allocated(p3)`, and `allocated(r3)` are inferred. Hence, resources p_2, r_2, p_4, and r_4 are not present in the allocation. Note that this may change with further binding and routing decisions made for the remaining tasks and messages.

3.2.2 Encoding Scheduling Constraints with Integer Difference Logic

In contrast to binding and routing, scheduling is a non-linear problem as the superposition principle does not hold when resource sharing and dependent tasks are considered. Hence, the start time of a task depends not only on its binding but also on its dependencies and resources shared with other tasks. A task t_i that expects data from another task t_j must not start before

the corresponding message has arrived. Thus, the predecessor task t_j has to finish its execution and the message has to be routed and transmitted through the communication infrastructure. Furthermore, even if two tasks t_i and t_j are independent of each other (i.e., there is no trail from t_i to t_j or vice versa) but are bound to the same computation resource, an indirect dependency is formed. As they cannot be executed simultaneously on the same resource, an execution order has to be established. This may lead to situations where the assignment of a task to another resource influences the start times of any task even when there are no direct dependencies imposed by the application graph. To summarize, a task t_i must execute after its predecessors but must not simultaneously run with another task on the same resource.

While this can be modeled with pure ASP rules, the encoding results in an exponentially increasing number of decision variables. The idea behind an encoding in ASP is that for each task and possible point in time, an individual atom is created. During solving, a subset of these atoms is selected that fulfills the aforementioned constraints. The problem with this encoding is that the exponential growth of the search space is not only dependent on the number of tasks in the application but also on the WCETs of the individual tasks. Furthermore, an upper threshold of the time slots must be defined. In general, this is a complicated problem as a too conservative estimation increases the search space too much and a too optimistic estimation can exclude solutions from the search. Hence, this encoding is especially not suited for large and complex problem instances.

To tackle this issue, in the following, an ASPmT[3-3] approach is proposed, where the schedulability analysis is performed through quantifier-free integer difference logic (QF–IDL) in a tightly coupled background theory. This way, the dependence on the WCET is eliminated and the encoding needs fewer variables. Moreover, the QF–IDL solver employed in the background theory is able to determine the schedulability of a given binding and routing in polynomial time. Furthermore, it provides an effective conflict analysis and reasoning whenever a given binding is found to be unschedulable allowing the foreground search to be steered away from unsatisfiable regions.

3.2.2.1 Quantifier-free Integer Difference Logic

The theory QF–IDL deals with conjunctions of constraints c that impose a maximum numerical difference between two variables. The constraints are of the form

$$c = (x - y \leq k)$$

with variables $x, y \in \mathbb{Z}$ and a constant $k \in \mathbb{Z}$. Thus, QF–IDL is a subset of integer linear programming (ILP) and is generally utilized to encode timing related, numeral problems such as scheduling. A conjunction of QF–IDL constraints is either unsatisfiable or contains infinite solutions [130].

Satisfiability Testing The conjunction $\mathcal{C} = c_1 \wedge \ldots \wedge c_n$ is characterized through a directed and weighted constraint graph $\mathcal{G}_\mathcal{C} = (V_\mathcal{C}, E_\mathcal{C})$. The set of vertices $V_\mathcal{C}$ of the constraint graph contains all variables of $\mathcal{C}$, i.e. $\forall(x_i - y_j \leq k_l) \in \mathcal{C} : v_i, v_j \in V_\mathcal{C}$. For each constraint $x_i - y_j \leq k_l \in \mathcal{C}$, there exists a weighted edge $(x_i, y_j, k_l) \in E_\mathcal{C}$, with x_i, y_j, and k_l indicating the source vertex, the destination vertex, and the weight, respectively. The constraint graph contains an

[3-3] Additionally, with the ASPmT, the usage of non-integer-valued execution times is possible that is not supported by plain ASP. Yet, this is out of the scope of the present monograph.

$$
\begin{aligned}
c_1 &: x_1 - x_4 \leq 3 \\
c_2 &: x_2 - x_3 \leq -5 \\
c_3 &: x_3 - x_1 \leq 1 \\
c_4 &: x_4 - x_3 \leq -4 \\
(c_5 &: x_1 - x_2 \leq -4)
\end{aligned}
$$

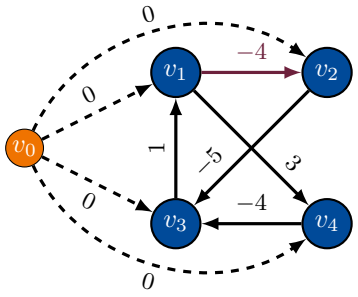


Figure 3.7: QF–IDL Example with corresponding constraint graph: introducing the additional difference constraint c_5 generates a negative cycle in the graph indicating that the conjunction of constraints is unsatisfiable.

additional vertex v_0 with accompanying edges to every other vertex with a weight of zero. By determining the shortest path originating from the additional vertex v_0 to each other vertex, a feasible solution can be determined. The accumulated weight along the shortest path from v_0 to each vertex $v \in V_{\mathcal{C}} \setminus \{v_0\}$ equals the value of the corresponding variable in $\mathcal{C}$. If, however, the shortest path algorithm detects a cycle with an accumulated negative weight, the conjunction $\mathcal{C}$ is unsatisfiable [130]. The complexity of QF–IDL is therefore dependent on the utilized shortest path algorithm. By applying the Bellman-Ford algorithm [131, 132] to determine the shortest path, a runtime of $O(n \cdot m)$ can be achieved, where n and m are the numbers of variables and constraints, i.e. vertices and edges, respectively.

As an example, consider the difference constraints $c_1, \ldots, c_4$ (the constraint c_5 and the corresponding edge $(v_1, v_2, -4)$ is not considered at first) and their corresponding constraints graph in Figure 3.7. While the blue vertices $v_1, \ldots, v_4$ and solid edges correspond to the variables $x_1, \ldots, x_4$ and the difference constraints, respectively, the orange vertex and the dashed edges represent the additional vertex and its connection to each other vertex with a weight of zero. Applying a shortest path algorithm from the special vertex v_0 to all other vertices results in no negative cycles. Hence, a feasible solution to the conjunction $\mathcal{C} = c_1 \wedge \ldots \wedge c_4$ exists with the variable assignments $\{x_1 = -4, x_2 = 0, x_3 = -5, x_4 = -1\}$. Note that this represents only one possible solution. Adding an arbitrary integer z to each variable x_i produces further solutions, e.g. $\{x_1 = 0, x_2 = 4, x_3 = -1, x_4 = 3\}$ by adding 4 to each variable. Adding the constraint c_5 and the corresponding edge $(v_1, v_2, -4)$ to the problem, a negative cycle is formed between the vertices $\{v_1, v_2, v_3\}$. Thus, the conjunction $\mathcal{C} = c_1 \wedge \ldots \wedge c_5$ does not have a feasible solution.

Difference constraints are a special case of general linear integer constraints of the form $\sum x_i \bowtie k$, where $i = \mathbb{N}$, $k \in \mathbb{Z}$, $x_i \in \mathbb{Z}$ are variables, and $\bowtie \in \{<, >, \leq, \geq, =\}$. Linear constraints with up to two variables can be transformed into QF–IDL as shown below.

$$
\begin{aligned}
x - y \geq k &\Rightarrow y - x \leq -k \\
x - y < k &\Rightarrow x - y \leq k - 1 \\
x - y > k &\Rightarrow y - x \leq -k - 1 \\
x - y = k &\Rightarrow x - y \leq k \wedge y - x \leq -k \\
x \leq k &\Rightarrow x - z_0 \leq k \mid z_0 \stackrel{!}{=} 0
\end{aligned}
\tag{3.1}
$$

The transformation of $x \leq k$ introduces a new variable z_0. This variable is required to be

```
1  #theory dl {
2      integer  {- : 1, unary};
3      diff_term {- : 1, binary, left};
4      &diff/0 : diff_term, {<=}, integer, any;
5  }.
6
7  &diff { x1 - x4 } <=  3.
8  &diff { x2 - x3 } <= -5.
9  &diff { x3 - x1 } <=  1.
10 &diff { x4 - x3 } <= -4.
```

Figure 3.8: QF–IDL Theory definition in ASP with *clingo 5*.

constant zero, as otherwise, the inequality would be semantically different. Note that z_0 and the special vertex v_0 are basically constrained by the same rules. The only difference is that v_0 contains zero-weighted edges whereas z_0 is specific to a particular constraint. However, both can be combined in the constraint graph $\mathcal{G}_C$ to reduce the vertex count. To this end, the zero-weighted edge originating from z_0 is replaced by the weight k of the corresponding constraint.

Theory Encoding Based on the definitions given above and the review on ASPmT in Section 2.2.3, difference logic is defined as a background theory in ASP as shown in lines 1–5 of Figure 3.8. The theory has the name `dl` and contains two theory term definitions `integer` and `diff_term` defined in lines 2 and 3 as well as one theory atom definition `&diff/0` in line 4. The latter represents a difference constraint that consists of the two theory terms of types `diff_term` and `integer` that are connected by the theory operator `<=`. The `any` keyword at the end permits the occurrence of a theory atom `&diff/0` both in the head and the body of an ASP rule. The theory term definition `diff_term` allows theory terms that contain the binary, left associative theory operator `-` (i.e., the mathematical minus). On the contrary, the `integer` term definition only permits the unary operator `-` (i.e., the mathematical negation). Note that the occurrence of the defined theory operators are optional in the theory term. Hence, an `integer` theory term may be build without the use of negation and represent an arbitrary variable or constant integer.

Consider again the conjunction of the difference constraints $c_1, \ldots, c_4$ in Figure 3.7. The problem is encoded using the four theory atoms in lines 7–10 as defined by the theory `dl`. For example, the theory atom in line 7, the theory term `x1 - x4` and the integer `3` are connected through the operator `<=`. Note that the difference constraints here are encoded as a fact (i.e., only a head is present) as they are directly specified. However, the theory variables generally are subject to grounding.

3.2.2.2 Schedulability Analysis

The schedulability analysis of the bound and routed application is imperative to evaluate the overall feasibility of the implementation. To this end, start times are assigned to each job j, that has to be executed in the system. Here, a job j is considered to be a task $t \in T$ or an individual hop $h_{c_i} \in \pi_c$ of a routed message c over a link. Thus, in the following the start time of a job $\tau(j)$ corresponds either to a task execution start time or a message hop, depending on the context. Note that here a periodic non-preemptive schedule is considered with a store-

and-forward switching strategy for routed messages. Hence, a task t with its corresponding period $p_A(t)$ is assumed to be restarted after exactly $p_A(t)$ time units on the same resource and, once started, a job is executed without interrupts. Furthermore, each hop of a message is considered to be unique, i.e., the complete message is sent from the source to the destination of one hop before the subsequent hop is taken. This implies the assumption of an unlimited buffer size at each resource to store message packets before they are consumed or transmitted further. The following considerations also assume the deadlines of the individual tasks to not exceed the specified periodicity of the applications. The definition of tasks that allow larger deadlines introduces further constraints (e.g., happening in pipelined executions). However, at the end of this section, an approach is presented to avoid the necessity of such constraints and allows handling of these applications.

Dependency Constraints For a given binding and routing to be feasible, individual jobs must adhere to their dependencies defined by the application graph $\mathcal{G}_A$ while allowing an overlap-free execution. When considering single-core systems without the possibility of parallelization, the schedulability analysis is trivial, as here, the accumulated execution times of all tasks must be at most equal to the considered periodicity[3-4] of the application $\sum w(t_i, r) \leq P$. A similarly trivial case exists if a multi-core hardware platform, but no dependencies between individual tasks are assumed. Here, the system is guaranteed to be schedulable if the sum of WCETs of all tasks mapped to a single resource never exceeds the specified periodicity of the system, i.e.,

$$\forall r \in \alpha : \sum_{m=(t,r)\in\beta} w(m) \leq P. \tag{3.2}$$

If both parallelization and dependencies are combined, Equation (3.2) becomes a necessary rather than a sufficient criterion for a schedulable implementation. To acquire a sufficient set of criteria that guarantee the correct job order the constraints in Equations (3.3) – (3.8) are required.

$$\forall t \in T : \boldsymbol{z_0 - \tau(t) \leq 0} \tag{3.3}$$

$$\forall (t,p) \in \beta : \boldsymbol{\tau(t) - z_0 \leq d_A(t) - w((t,p))} \tag{3.4}$$

$$\begin{gathered}\forall c \in C \mid \{(t_i,c),(c,t_j)\} \subseteq E_A, \{(t_i,p),(t_j,p)\} \subseteq \beta : \\ \boldsymbol{\tau(t_i) - \tau(t_j) \leq -w((t_i,p)) + s(c) \cdot p_A(t_j)}\end{gathered} \tag{3.5}$$

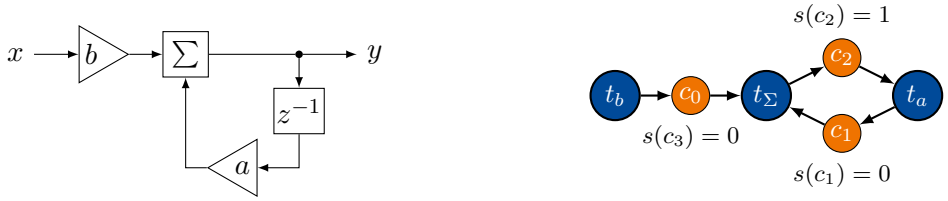
$$\forall c \in C \mid (t_i,c) \in E_A, \pi_c \neq \emptyset, (t_i,p) \in \beta : \boldsymbol{\tau(t_i) - \tau(h_{c,1}) \leq -w((t_i,p))} \tag{3.6}$$

$$\forall c \in C \mid |\pi_c| > 1, \forall i \in [2, |\pi_c|], h_{c,i-1} = (c,l) : \boldsymbol{\tau(h_{c,i-1}) - \tau(h_{c,i}) \leq -d_P(l)} \tag{3.7}$$

$$\begin{gathered}\forall c \in C \mid \{(t_i,c),(c,t_j)\} \subseteq E_A, (t_j,p) \in \beta, \pi_c \neq \emptyset, h_{c,|\pi_c|} = (c,l) : \\ \boldsymbol{\tau(h_{c,|\pi_c|}) - \tau(t_j) \leq -d_P(l) + s(c) \cdot p_A(t_j)}\end{gathered} \tag{3.8}$$

The first constraint in Equation (3.3) ensures that all tasks $t \in T$ start after time point 0. Therefore, the special variable z_0 is introduced. Note that this constraint directly complies to the dedicated zero vertex v_0 and its corresponding edges that are included into the constraint graph $\mathcal{G}_\mathcal{C}$. The second constraint in Equation (3.4) is needed to ensure that each task finishes its execution before its corresponding deadline. Equation (3.5) defines the constraints that

[3-4] When multiple independent applications are considered, the hyper-period $P_H = lcm(A_1, \ldots, A_n)$ of all applications has to be used instead.



(a) First-order IIR filter **(b)** Modeled application graph of the IIR filter

Figure 3.9: A first-order IIR filter depicted as block diagram and modeled by an application graph with three tasks and three messages.

have to be included for each pair of dependent tasks that are mapped to the same resource. As task t_j depends on the results of task t_i, it must not start before t_i finishes its execution. However, if they form an inter-iteration dependency, the index delay of the connecting message c has to be accounted for. As all tasks are required to restart at their specified periodicity rate p_A, this implies for the current iteration that the temporal distance must be smaller than the index delay multiplied by the periodicity of t_j.

As an example, assume a first-order IIR filter as depicted in Figure 3.9. The filter response is acquired by adding the weighted input of the filter with the weighted filter response of the previous iteration. For the sake of simplicity, only the two tasks t_Σ and t_a connected by two messages c_1 and c_2 with their corresponding index delays $s(c_1) = 0$ and $s(c_2) = 1$ are considered in following, i.e., t_b and c_0 are assumed to execute independently. Further, assume both tasks are bound to the same resource p, WCETs of $w(t_\Sigma, p) = 1$ and $w(t_a, p) = 4$, and have a periodicity of 6 time units. According to Equation (3.5), two constraints are added: (1) $\tau(t_a) - \tau(t_\Sigma) \leq -4$ and (2) $\tau(t_\Sigma) - \tau(t_a) \leq -1 + 1 \cdot 6$. While the first constraint guarantees that the multiplication is started some time after the addition, the second constraint ensures that it is not delayed by more than 5 time units. Now assume the tasks are bound to another resource r, with $w(t_+, r) = 2$ and $w(t_\times, r) = 6$. This induces the constraints (1) $\tau(t_a) - \tau(t_\Sigma) \leq -6$ and (2) $\tau(t_\Sigma) - \tau(t_a) \leq -2 + 1 \cdot 6$, that do not have a feasible solution. Hence, the system is unschedulable in this configuration[3-5].

The remaining constraints regard the execution order of tasks that are not bound to the same resource, i.e., where the route π_c of the connecting message is not empty. Equation (3.6) concerns the initial hop. Intuitively, the first hop $h_{c,1}$ must not be sent before the sending task finishes its execution. The second message constraint in Equation (3.7) considers all subsequent hops. A message hop $h_{c,i}$ can only be sent after the previous hop $h_{c,i-1}$ has been received. Hence, the routing delay d_P of the corresponding hop must be used. Finally, Equation (3.8) addresses the start time of the receiving task. Similar to the task constraint in Equation (3.5), the index delay of the message is considered here to guarantee inter-iteration dependencies. Note that the index delay must not be involved in multiple constraints of the same message. If the index delay was considered in each message hop, the message would be delayed n times the index delay $s(c)$, with n equals the number of hops of the corresponding message. This would lead to incorrect schedules.

Theorem 3.2.1: The conjunction of constraints defined in Equations (3.3) to (3.8) guarantees an overlap-free execution of intra-iteration dependent jobs.

[3-5]For the sake of the argument, the necessary constraint defined in Equation (3.2) is assumed not to be tested.

Proof. The theorem is proven by induction. Let $a = \langle v_1, v_2, \ldots, v_n \rangle$ with $\sum_{c \in A} s(c) = 0$ be a trail through an application graph $\mathcal{G}_A$. According to Definition 3.1.3, each pair of vertices $(v_i, v_j) \subseteq a$ are therefore intra-iteration dependent of each other, i.e., $v_i \overset{*}{\leftrightarrow} v_j$.
Initial step: Vertices v_1 and v_2 are directly dependent of each other. Independent of the job types of v_1 and v_2, a constraint of form $\tau(v_1) - \tau(v_2) \leq -k_1$ is generated, where k_1 signifies the execution time of v_1. The terms $s \cdot p_A$ of Equations (3.5) and (3.8) are irrelevant as the index delay is guaranteed to be zero for intra-iteration dependent jobs. Hence, the start time of $\tau(v_j)$ must be at least equal to the end time of v_1, i.e., $\tau(v_2) \geq \tau(v_1) + k_1$. Therefore, two subsequent jobs v_{i-1}, v_i cannot overlap.
Again, for the successor of v_i, a similar constraint of the form $\tau(v_i) - \tau(v_{i+1}) \leq k_i$ is generated. As $\tau(v_i)$ is already shown to be larger than the end time of its predecessor v_{i-1}, and $k_i \geq 0$, the inequality chain results in $\tau(v_{i+1}) \geq \tau(v_i) + k_i \geq \tau(v_{i-1}) + k_{i-1}$. Hence, the successor $v_{(i+1)}$ of a job v_i cannot overlap with the predecessor v_{i-1}. □

Resource Sharing Constraints Equations (3.3) – (3.8) guarantee that dependent jobs are scheduled in the correct order and intra-iteration dependent jobs do not overlap. However, independent (both intra- and inter-iteration) jobs are not considered and may form conflicts when sharing identical resources. Therefore, Equations (3.9) and (3.10) are defined additionally.

$$\begin{gathered} \forall \{(t_i, p), (t_j, p)\} \subseteq \beta \mid t_i \overset{*}{\leftrightarrow} t_j : \\ \boldsymbol{\tau(t_i) - \tau(t_j) \leq -w((t_i, p)) \veebar \tau(t_j) - \tau(t_i) \leq -w((t_j, p))} \end{gathered} \tag{3.9}$$

$$\begin{gathered} \forall \{h_{c_i,k}, h_{c_j,l}\} \subseteq \bigcup_{c \in C} \pi_c \mid c_i \overset{*}{\leftrightarrow} c_j, h_{c_i,k} = (c_i, e), h_{c_j,l} = (c_j, e) \in E_P : \\ \boldsymbol{\tau(h_{c_i,k}) - \tau(h_{c_j,l}) \leq -d_P(e) \veebar \tau(h_{c_j,l}) - \tau(h_{c_i,k}) \leq -d_P(e)} \end{gathered} \tag{3.10}$$

While Equation (3.9) regards independent tasks mapped onto identical processing elements, Equation (3.10) handles the schedulability analysis of independent message hops routed over identical links. The rationale behind both constraints, however, is identical. If two intra-iteration independent jobs j_k and j_l are supposed to be executed on the same resource, the jobs have to be sequenced. Therefore, either job j_k or job j_l has to be delayed until the other is finished, encoded by two difference constraints. As the difference constraints are connected through an *"exclusive or"* relation, they must not be satisfied simultaneously. Assume the tasks t_2 and t_3 of application A_1 in the specification example in Figure 3.1 to be bound to the processing element p_3. Although the dependency constraints in Equations (3.3) – (3.8) guarantee the execution to be delayed after the messages c_1 and c_2 have been arrived, respectively, the tasks may still overlap temporally. Hence, either the difference constraint $\tau(t_2) - \tau(t_3) \leq -2$ or $\tau(t_3) - \tau(t_2) \leq -2$ has to be included. While the former forces t_3 to be executed after t_2, the latter delays t_2 analogously.

Schedulability Encoding The encoding of the difference constraints defined in Equations (3.3) to (3.8) is depicted in Figure 3.10. Note that the theory definition `dl` as shown in Figure 3.8 is used in the following. Each dependency constraint can be directly encoded by one ASP rule in lines 1–6, respectively. During the grounding step, the constraint of Equation (3.3) is generated for each task by the rule in line 1. The special zero vertex v_0 is explicitly defined as `0` in the first theory term. On the contrary, the theory atoms that are generated by the grounder in line 2 depend on the current binding. Hence, only if a mapping $m \in M$ is

```
1 &diff { 0 - T } <= 0 :- task(T,A).
2 &diff { T - 0 } <= V :- bind(M,T,_), deadline(T,D), wcet(M,W), V=D-W.
3 &diff { Ti - Tj } <= V :- send(Ti,C), read(Tj,C), bind(Mi,Ti,R), bind(Mj,Tj,R),
    ↪ period(Tj,P), delay(C,S), wcet(Mi,W), V=S*P-W.
4 &diff { T - (C,L) } <= -W :- send(T, C), reached(C,L,R,_), bind(M,T,R),
    ↪ wcet(M,W).
5 &diff { (C,Li) - (C,Lj) } <= -D :- reached(C,Li,_,R), reached(C,Lj,R,_),
    ↪ routingdelay(Li,D).
6 &diff { (C,L) - T } <= V :- read(T,C), reached(C,L,_,R), bind(M,T,R),
    ↪ routingdelay(L,D), delay(C,S), V=S*P-D.
```

Figure 3.10: Encoding of Dependency Constraints

selected for the binding, i.e., $m \in \beta$, the corresponding theory atom is inferred. Within a theory atom, the grounder only performs string replacements and cannot be used to evaluate variables. Thus, the value of V used as the right-hand side of the inequality within the theory atom must be calculated outside. This is done in the body of the ASP rule through `V=D-W` which uses the deadline of each task and the WCET of its mapping options. As an example, consider the mapping options m_1, m_2, and m_3 of a task t_1 with $w(m_1) = 3$, $w(m_2) = 1$, $w(m_3) = 2$, and $d_A(t_1) = 12$. Grounding line 2 will generate three variable-free rules for task t_1: `&diff {t1-0} <= 9 :- bind(m1,t1,p1).`, `&diff {t1-0} <= 11 :- bind(m2,t1,p2).`, and `&diff {t1-0} <= 10 :- bind(m3,t1,p4).`

The rules generated by line 3 are only inferred if the two corresponding tasks t_i and t_j are bound to the same resource r and are directly connected through a message c, i.e., $(t_i, r), (t_j, r) \in \beta \wedge (t_i, c), (c, t_j) \in E_A$. The mechanism to calculate the constant value is similar to the rule in line 2 but utilizes the index delay of the message c, the WCET of task t_i, and the periodicity of task t_j as defined by Equation (3.5).

The three final lines implement the dependency constraints of the individual message hops as defined by the Equations (3.6) to (3.8), respectively. The constraints defined in Equation (3.6) must only be satisfied if two dependent tasks are not bound to the resource. This implies that the connecting message is routed over at least one hop that originates from the resource of the sending task. Hence, the existence of a `reached/4` atom with the corresponding source resource suffices to guarantee that. If two consecutive `reached/4` atoms have been inferred for the same message, the rules generated by the grounder in line 5 are inferred. Finally, line 6 implements Equation (3.8). It basically plays the counterpart to line 4, and encodes the final hop of a message with the resource of the reading task as destination.

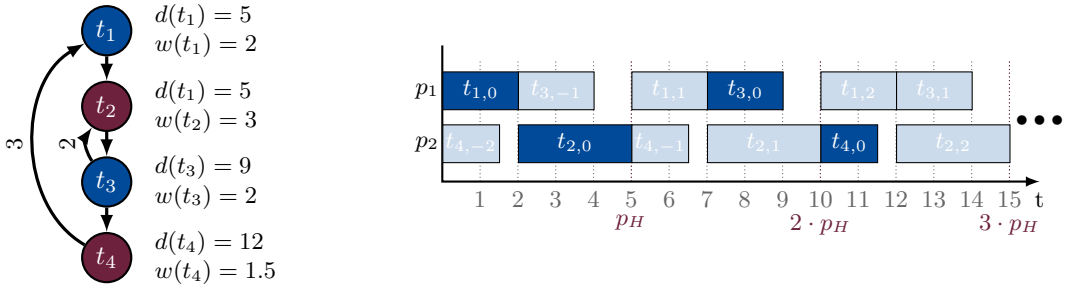
While the dependency constraints are encoded by a single rule, the resource sharing constraints defined by Equations (3.9) and (3.10) need a set of additional decision atoms to be decided first as shown in Figure 3.11. The encoding here is grouped into four parts. The first three lines determine the transitive dependency relations of all vertices of the application. Note that the atom `depends/2` encodes the intra-iteration dependency ($v_i \overset{*}{\leftrightarrow} v_j$), as in line 8, it is only inferred between a message and a reading task if the index delay of the message is equal to zero. Lines 11 and 12 determine if two tasks are bound to the same resource or individual message hops are routed over identical links, respectively. The corresponding `conflict/2` atoms are only inferred if both jobs are not intra-dependent of each other and the terms $Ti < Tj$ and $Ci < Cj$ ensure that the jobs are unique. Subsequently, in line 14, a priority is defined for each conflict. That is, a partial order is decided which job shall get precedence

```
7  depends(Ti,C) :- send(Ti,C).
8  depends(C,Tj) :- read(Tj,C), delay(C,0).
9  depends(A,C) :- depends(A,B), depends(B,C).
10
11 conflict(Ti,Tj) :- bind(Mi, Ti, P), bind(Mj, Tj, P), Ti < Tj,
       ↪ not depends(Ti,Tj), not depends(Tj,Ti).
12 conflict(Ci,Cj) :- reached(Ci,L,_,_), reached(Cj,L,_,_), Ci < Cj,
       ↪ not depends(Ci,Cj), not depends(Cj,Ci).
13
14 1{ prio(A,B), prio(B,A) }1 :- conflict(A,B).
15 #edge (A,B) : prio(A,B).
16
17 &diff { Ti - Tj } <= -W :- prio(Ti,Tj), bind(M,Ti,_), wcet(M,W).
18 &diff { (Ci,L) - (Cj,L) } <= -D :- prio(Ci,Cj), reached(Ci,L,_,_),
       ↪ reached(Cj,L,_,_), routingdelay(L,D).
```

Figure 3.11: Encoding of Resource Sharing Constraints

over the other. This is realized through the choice rule in the head, that only selects either `prio(A,B)` or `prio(B,A)`. A set of partial orders may result in unsatisfiable priority chains, such as `prio(a,b). prio(b,c). prio(c,a).` leading to non-functional deadlocks. The built-in acyclicity rule [133] in line 15 is used to prevent such cycles. The actual resource sharing constraints defined by Equations (3.9) and (3.10) are implemented in lines 17 and 18, respectively. Each time, a `prio/2` atom has been inferred, a difference term is generated that sequences the execution of the corresponding two tasks. While the rule in line 17 is only inferred if a `bind/3` atom exists, the rule in line 18 is only inferred if the two corresponding `reached/4` atoms exist. If, for instance, the atom `prio(ti,tj)` is inferred, a difference constraint of the form $\tau(t_i) - \tau(t_j) \leq -d_A(t_i)$ is generated that sequences the execution of t_i and t_j, i.e., it shifts the start time $\tau(t_j)$ behind the execution of t_i encoded by the difference theory atom `&diff {ti-tj} <= -W`.

Iteration Unrolling So far, the constraints described above can only handle problem instances where the deadlines of the tasks do not exceed the specified periodicity of the applications. However, the application model proposed in Section 3.1 allows arbitrary deadlines. As an example, assume the application in Figure 3.12(a). For the sake of simplicity, only tasks are depicted here and the binding is assumed to be fixed. In detail, the tasks t_1 and t_3 are bound to the processing element p_1 while t_2 and t_4 are bound to processing element p_2. The execution times of the tasks $t_1, \ldots, t_4$ are given as $w(t_1) = w(t_3) = 2$, $w(t_2) = 3$, and $w(t_4) = 1$. The corresponding deadlines are $d(t_1) = d(t_2) = 5$, $d(t_3) = 9$, and $d(t_4) = 11$. As the tasks are dependent on each other, the periodicity of each task is identical and is specified as $p(t_i) = 5$. Hence, tasks t_3 and t_4 have a deadline that is larger than the periodicity of the application. Furthermore, the application consists of two inter-iteration dependencies with index delays of 2 and 3, respectively. That is, the fourth execution of t_1 depends on the first execution of t_4 and the third execution of t_2 depends on the second execution of t_3. Tasks that share the same colors are assumed to be bound to identical processing elements. Figure 3.12(b) depicts a valid schedule. The interesting parts of the schedule are that first execution of t_3 (i.e., $t_{3,1}$) is started after the second execution of t_1 ($t_{1,2}$) and the first execution of t_4 is shifted behind the second execution of t_2. However, this is not possible to express with the constraints that have been introduced in the paragraphs above. Instead, $t_{3,1}$ and $t_{4,1}$ would overlap with $t_{1,2}$



(a) Application Graph **(b)** Valid Schedule

Figure 3.12: Example specification with deadlines larger than the periodicity.

and $t_{2,2}$, respectively. Thus, a correct analysis of the resulting latency would not be possible. To detect these collisions, one possibility is to project all task executions into one common iteration. This is visualized in the Gantt chart by the virtual preceding iterations, i.e., $t_{3,0}$ and $t_{4,-1}$, respectively. However, this is only possible after all decisions have been made rendering the evaluation of partial assignments infeasible.

A remedy to this problem is to unroll the application in a way that all resource sharing constraints can be adhered. Similar to loop unrolling done by compilers, the application is multiplied. Afterwards, virtual dependencies between the iterations are inserted and the periodicity, deadlines and inter-iteration dependencies have to be adapted. The factor f by which the application tasks are multiplied depends on the ratio between the latest deadline and the periodicity p_H, i.e.:

$$f = \left\lceil \frac{\max(d(t_i))}{p_H} \right\rceil .$$

This way, all potential collisions can be detected. Thus, in the running example, the applications had to be multiplied three times as $\lceil 11/5 \rceil = 3$. Next, each execution of a copied task $t_{i,j}$ has to be guaranteed to adhere to the specified periodicity, i.e., it has to be executed exactly $j \cdot p_H$ time units after the original task t_i. This can be expressed by the linear term $\tau(t_i) - \tau(t_{i,j}) = -j \cdot p_H$[3-6]. As multiple iterations are now unrolled, the periodicity $\widetilde{p_H}$ of the adapted application has to be changed to f times the original periodicity, i.e., $\widetilde{p_H} = f \cdot p_H$. The deadlines of the copied tasks $t_{i,j}$ are given by $d(t_{i,j}) = d(i) + j \cdot p_H$. The remaining properties of the tasks are identical to the original ones. Particularly, the new tasks have to be bound to the same processing elements as the original ones. Therefore, in the encoding, an additional rule of form `bind(Tij,R):-bind(Ti,R)` is added to the logic program. Hence, if a mapping $m = (t_i, p)$ is assigned `TRUE` for the task t_i, the mapping $m = (t_{i,j}, p)$ has to be assigned `TRUE` simultaneously. Finally, for any inter-iteration dependency $\langle t_s, c, t_r \rangle$ with $s(c) > 0$, the corresponding message c and the index delay $s(c)$ thereof have to be adapted. To this end, the original edge (c, t_r) is removed from E_A. The new receiving task $t_{r,j}$ is determined by analyzing the index delay $s(c)$ and the multiplication factor f. The index j is calculated by $j = s(c) \bmod f$, with mod being the modulo operation, and the new index delay $\widetilde{s(c)}$ is updated with $\widetilde{s(c)} = \lfloor \frac{s(c)}{f} \rfloor$. Finally, the new edge $(c, t_{r,j})$ is added to E_A.

Figure 3.13 shows the result of the iteration unrolling process for the running example. In

[3-6] This is transformed into two difference constraints as shown in Equation (3.1) on page 53.

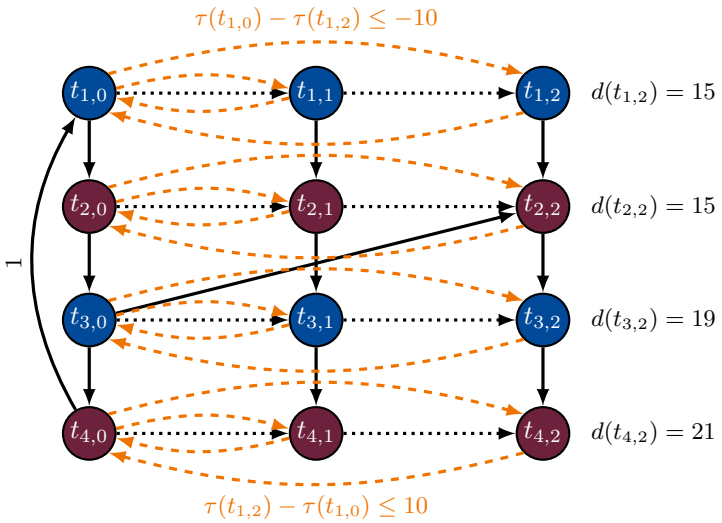


Figure 3.13: The adapted application graph allowing the application of Figure 3.12(a) to be analyzed by the proposed approach.

the figure, the dashed orange lines represent the difference terms while the black solid lines depict dependencies. The black dotted lines shall not to be considered at the moment. For the sake of clarity, only the adapted deadlines of the tasks $t_{1,2}$ to $t_{4,2}$ are shown. Similarly, only two difference terms are specified explicitly. The remaining ones are defined analogously. The two inter-iteration dependencies of the original application have been changed. While the first dependency still renders an inter-iteration dependency, the second one has been changed into a default dependency, i.e., with $s = 0$. Yet, the fourth execution of the task t_1 still depends on the first execution of t_4 and the third execution of t_2 still depends on the first execution of t_3. Additional dependencies are not necessary to analyze the latency of the application correctly. For example, there is no edge between the tasks $t_{4,1}$ and $t_{1,1}$ as the timing constraints are already guaranteed by the fixed difference constraints between the individual task copies. Furthermore, the resource sharing constraints, defined in Equation (3.9), can now correctly detect and prevent collisions between tasks $t_{1,1}$ and $t_{3,0}$ as well as $t_{2,1}$ and $t_{4,0}$. As, for example, tasks $t_{4,0}$ and $t_{2,1}$ are mapped to the same resources, the ASP solver can explicitly decide which is executed first.

The disadvantage of this approach is an increasing number of tasks, messages, and edges, and in turn, decision variables that have to be assigned. However, many of the decisions do not impact the complexity of the problem. That is, the binding of the additional tasks are essentially unit clauses that are automatically assigned whenever the original task is bound to a processing element. The same holds true for the difference constraints forcing the periodic execution of the tasks. These do not have to be decided at all but instead are fixed from the beginning of the search. Only the additional resource sharing constraints remain. To reduce the number of additional decisions further, consider the dotted black lines in Figure 3.13. Including these dependencies, copied tasks become dependent of each other. Hence, additional resource sharing constraints can be omitted in these cases as only independent tasks must be considered (q.v., Equation (3.9)). In this example, this limits the additionally introduced constraints to resource sharing constraints between the pairs $(t_{3,0}, t_{1,1})$, $(t_{3,0}, t_{1,2})$, $(t_{3,1}, t_{1,2})$, $(t_{4,0}, t_{2,1})$, $(t_{4,0}, t_{2,2})$, and $(t_{4,1}, t_{2,2})$.

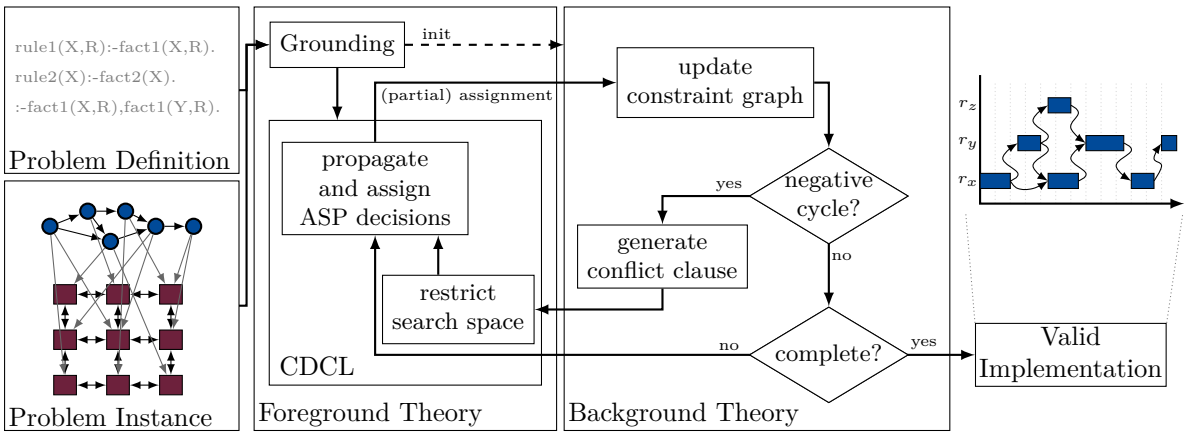


Figure 3.14: Architecture Overview of the synthesis framework.

As a conclusion, the presented techniques can also be utilized to handle applications where task deadlines exceed their periodicity. Yet, they demand a preprocessing of the problem instance that leads to a slight increase in the number decisions to be made. This, however, does not impact the search itself as it has to be done on the instance prior to solving. Hence, in the following, the instances are assumed to be preprocessed before applying the proposed synthesis approach.

3.3 Theory Propagation

The ASP encoding presented in the previous section allows a holistic definition of both specification and feasibility constraints that encompass all steps of the synthesis process, viz. allocation, binding, routing, and scheduling. This section elaborates on the integration of the background theory into the foreground ASP solver. Only the tight integration enables the use of sophisticated solving techniques such as theory propagation and the evaluation of partial assignments. At first, an overview of the entire framework is given before the individual parts are discussed in more detail.

3.3.1 Framework Overview

The overview of the synthesis framework is depicted in Figure 3.14. The solving process is logically split into two parts. While the foreground theory solver assigns and propagates ASP decisions, the background theory solver evaluates the satisfiability of the corresponding scheduling constraints. Both solvers are intertwined through the theory atoms `&diff` as defined in the previous section. On the one side, the inferred difference atoms are relayed from the foreground to the background theory for evaluation. On the other side, unsatisfiable conjunctions found by the background theory are reported back to the foreground solver in the form of conflict clauses. The process is started by grounding the input in the form of the general problem definition and the specific problem instance resulting into a variable-free representation of the specification. The foreground ASP solver subsequently assigns and propagates decisions conforming to the conflict-driven clause learning (CDCL) paradigm. The background difference logic solver man-

ages a constraint graph generated from the corresponding difference atoms. Thus, whenever an additional difference atom is inferred, the constraint graph is updated with the corresponding vertices and edge. A shortest path algorithm is subsequently applied to the graph that is used to compute job start times and to determine if the constraints form a negative cycle rendering the conjunction of constraints unsatisfiable. If a negative cycle is detected, the background theory analyzes the reason of the negative cycle, generates a corresponding conflict, and relays it back to the foreground solver to exclude that specific set of constraints from the search. If no negative cycle is detected, the schedule is checked whether all periodicity constraints are fulfilled in subsequent iterations of the execution of the system. Failing this test leads to additional difference constraints that have to be added to the constraint graph and the negative cycle checks have to be conducted anew. If the test has passed and there are no further decisions to be taken, the solving process terminates with a valid implementation found. Otherwise, the search is continued in the foreground solver until the entire search space has been covered.

3.3.2 Stateful Propagation

The background theory is implemented as a stateful propagator. That is, the constraint graph is iteratively updated and not build from scratch each time a new theory atom is inferred. If the foreground solver has to backjump to a previous decision level due to an encountered conflict, the constraint graph has to revert the corresponding changes as well. Therefore, the background theory keeps track of all changes throughout the solving process.

The implementation of the propagator consists of four phases: `init`, `propagate`, `undo`, and `check`. The `init` phase is executed at the start of solving. Here, all `&diff` atoms are extracted from the variable-free representation of the problem and the associated information regarding corresponding vertices and edge weights is saved. Moreover, the literals that correspond to the difference atoms are inserted into a watch list, in order to invoke the theory solver whenever they are assigned throughout solving.

The actual theory evaluation is performed in the `propagate` phase. Whenever an atom in the watch list gets assigned by the ASP solver, the theory propagation is invoked where the conjunction of difference constraints is checked for consistency. As detailed in Section 3.2.2, this is done by translating the difference atoms into a constraint graph and performing a shortest path analysis originating from a dedicated vertex v_0. As the propagator is invoked each time a difference constraint is assigned, only a partial assignment is available. Thus, the constraint graph is updated incrementally throughout the solving process. The advantage of this methodology is that invalid subsets of difference constraints are recognized early, potentially pruning large regions of the search space without the need to wait for a full assignment. Although the Bellman-Ford algorithm [131, 132] is considered the fastest algorithm for that problem, it does not support an incremental operation. Hence, applying Bellman-Ford would require the re-evaluation of the entire graph each time new a difference constraint was assigned and its corresponding edge was added. Instead, whenever a new constraint is added to the constraint graph, an algorithm based on the *incremental consistency checking* algorithm proposed by [134] and [135] is utilized. The algorithm, sketched in Figure 3.15, is able to detect negative cycles and determine the shortest path to each vertex if no negative cycle exists in an incremental process.

Moreover, it allows ignoring vertices that are not influenced by the addition of edges implied by the assignment of additional constraints. A potential, initialized with zero, is assigned to each vertex of the constraint graph. Each time a difference constraint $u - v \leq w$ is inferred,

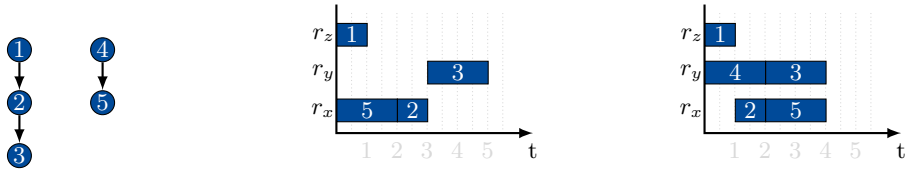
```
1: function ADD_EDGE(u -w-> v)
2:     if u.potential + w < v.potential then
3:         v.potential ← u.potential + w
4:         heap.push ← v
5:         v.from ← u -w-> v
6:     while heap ∧ ¬u.from do
7:         s ← heap.pop
8:         for ∀s -c-> t do
9:             if s.potential + c < t.potential then
10:                if ¬t.from then
11:                    t.potential ← s.potential + c
12:                    heap.push ← t
13:                    t.from ← s -c-> t
14:                else
15:                    heap.revert ← t
```

Figure 3.15: Pseudocode of the incremental consistency checking algorithm based on [135]

the constraint graph is updated with the corresponding edge, denoted as $u \xrightarrow{w} v$. Subsequently, the graph is traversed starting at the source vertex u of the new edge. Thereby the vertex potential of the successor vertex v is updated if the weight of the corresponding edge decreases its potential. Only if the potential has been decreased, the corresponding vertex is marked as a starting point for further traversal, rendering a conditional breadth-first search. This procedure is repeated until either the initial source vertex u is reached or the there are no more vertices marked for traversal. In the former case, the algorithm has detected a negative cycle in the graph, i.e., the conjunction of the underlying difference constraints of the graph is proven to be unsatisfiable (inconsistent). The latter case implies that the addition of the new edge $u \xrightarrow{w} v$ does not lead to a negative cycle and the vertex potentials correspond to their shortest path. In the worst case, each vertex of the graph is traversed once, leading to a runtime of $\mathcal{O}(m+n \log n)$, with m and n representing the number of edges and vertices, respectively [135]. Although the complexity is identical to the Dijkstra algorithm, it allows handling of negative edge weights.

If a negative cycle is detected, the conjunction of constraints inferred by the foreground solver are known to be unsatisfiable. Hence, a conflict clause has to be generated to prevent the solver from making the same decisions again. In its simplest form, the entire conjunction of difference constraints is used as conflict clause. However, providing a minimal reason is beneficial to prune the search space more effectively. The advantage of the algorithm at hand is the ability to automatically provide the edges that lead to a negative cycle. This information can be directly used to generate a minimal conflict clause as only the conjunction of edges in the negative cycle render the constraint graph infeasible.

The conflict clause returned to the foreground solver triggers a backjump in the search. As a result, theory atoms in the form of difference constraints inferred previously may become unsatisfied. Thus, the constraint graph has to be reverted accordingly and the `undo` phase is executed. To this end, the theory propagator keeps track of the history tied to the various decision levels of the ASP solver. Whenever an edge is added to the graph and a potential of a vertex is updated according to the partial assignment of the current decision level, the change is pushed onto a stacked trail. Jumping back to a previous decision level, the foreground solver informs the background theory about the desired decision level. Using the trail of changes, the state of the constraint graph at this decision level can be easily regained without the need to



(a) Application Graph **(b)** Partial assignment **(c)** Complete Assignement

Figure 3.16: Scheduling anomaly shown on the example of a simple application graph consisting of two independent applications.

rebuild the entire graph.

Note that the validity of this approach using partial assignments for constraint and dominance checks depends on the properties of the corresponding objective functions. Only if the objective function is assignment monotonic, pruning decisions based on the incomplete implementation is possible. Hence, the quality of an implementation candidate must not improve with further decisions made. With respect to area and energy requirements, this property holds trivially, i.e., the selection of additional mappings implies more energy to be consumed and potentially requires additional resources to be allocated. However, regarding the latency in self-timed systems, a *scheduling anomaly* may occur. It is a phenomenon where executing an additional task or sending a message packet over a link may influence the schedules of subsequent jobs. In turn, this influence may reduce the overall end-to-end latency of the entire system, and thus, preventing an assignment monotonic behavior. An example of a scheduling anomaly is given in Figure 3.16. The application in Figure 3.16(a) shall be implemented on the three resources r_x, r_y, and r_z, with corresponding execution times of $w(1, r_x) = 1$, $w(2, r_z) = 1$, $w(3, r_y) = 2$, $w(4, r_y) = 2$, and $w(5, r_z) = 2$. When only the first four mappings are decided, a self-timed schedule as given in Figure 3.16(b) is deduced. As task t_4 is not yet bound to a resource, the start time of its successor t_5 is not constrained and can start at time step 0. In turn, this delays the execution of t_2 and subsequently t_3 resulting in an end-to-end latency of 5 time units. Assume, the deadline of the task t_3 to be 4. Hence, the partial assignment would not adhere to the constraints, and should be removed. When, in the next step, the task t_4 is assigned to resource r_y, the execution of t_5 is delayed by 2 time units. Hence, t_2 can be executed earlier and both t_3 and t_5 can start in parallel. This leads to reduced end-to-end latency of 4. Prematurely removing the solutions would have incorrectly excluded the solution from the search. Hence, excluding the implementation, e.g., due to a missed deadline or similar, before a complete assignment is available is not valid if a fully self-timed behavior is considered.

A possible remedy to this problem is the waiver of partial assignment checking and only applying the synthesis on complete assignments. However, this would limit the pruning capabilities of the approach. This can be avoided by the introduction of fixed priorities between tasks forcing a specific execution order. This way, delays that are introduced by additional decisions cannot decrease the end-to-end latency of the entire system. For the running example of Figure 3.16, assume the fixed priority $t_5 \succ t_2$), i.e., t_5 is always executed before t_2. Binding task t_4 now does not influence the execution order. Instead, t_2 is executed at time step 4, increasing the end-to-end latency of the system. Hence, the partial assignment could be safely pruned from the search. The search would then continue, eventually assigning the priority $t_2 \succ t_5$, always forcing t_5 to be executed after t_2. In the end, with this, the solution in Figure 3.16(c) is obtained.

3.4 Evaluation

In this section, the proposed ASP-based approach is evaluated. To allow for an extensive supply of specification instances for the evaluation, the section starts with addressing the problem of creating test instances. Here, a systematic instance generator is proposed that permits designing test instances following regular patterns and properties. In particular, the degree of parallelism of the generated instances can be specified, although each instance is unique. Thus, not only one problem instance is generated but a set of individual specifications with similar characteristics. The remainder of this section evaluates the synthesis approach with a set of instances obtained by the proposed instance generator. Here, the superiority of partial over full assignment checking is shown on the basis of a number of problem instances.

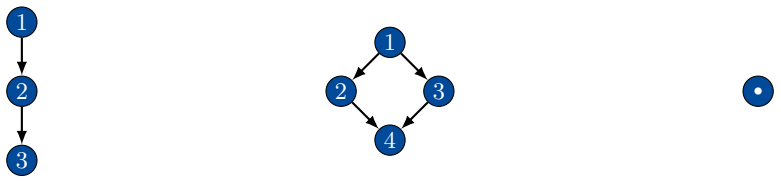
3.4.1 Test case generation

While, in recent time, many design space exploration (DSE) techniques to the system-level synthesis problem of embedded systems design have been proposed, a systematic approach on how to produce a viable set of variant test cases with definite similar properties is not available. In the following, a methodology is proposed for the test case generation of DSE techniques and a versatile and easily expendable instance generator based on ASP is presented that is able to produce hard synthesis problem instances.

However, the specific problem instance in use is another important factor when evaluating a DSE technique as differently structured applications and assumptions can lead to both easier and harder optimization problems. On the one hand, in order to evaluate the performance of different DSE approaches with respect to each other, it is imperative that the DSE inputs are similar and easily reproducible. On the other hand, for the development of new techniques, test cases that represent a specific class of problems are desired to cover a large input space. Thus, variant test cases with similar properties must be used to get meaningful results on the performance of a DSE technique for specific classes of problem instances.

In this section, a methodology for generating system-level problem instances is proposed. The generator utilizes the constraint solving capabilities of ASP to define the desired characteristics of generated test cases. The most important properties of the generator are listed below:

- **Systematic generation:** Rules encoded in ASP guarantee the compliance to desired characteristics of the resulting test case. This involves the number of tasks, messages, processing and communication elements as well as communication behavior of the application.
- **Varied test cases:** Utilizing the stable model semantics of ASP, the generator produces variant instances with similar characteristics as described above while the shape of the generated applications differ. This is important to evaluate DSE techniques with respect to various classes of applications.
- **Modularity:** The generation of application and architecture templates are independent of each other, allowing the exchange of rules individually. Furthermore, as of the modular structure, additional domain-specific properties can be added easily.
- **Platform independence:** The generator is implemented in ASP allowing it to be executed on each system on which a compatible ASP solver is installed.



(a) Series pattern **(b)** Parallel pattern **(c)** Terminal pattern

Figure 3.17: Different patterns considered in the series-parallel graph generation.

The generator is designed to generate specification instances that comply to the system model presented in Section 3.1. That is, it generates an application graph, a platform template, and the mapping options connecting the former two. Therefore, the generator is separated into three individual modules: the application module (Section 3.4.1.1), the architecture module (Section 3.4.1.2), and the mapping module (Section 3.4.1.3). The constraint generation of the resulting instances is more complicated as the generated problem shall not be trivially solvable but simultaneously not unsatisfiable. Optimally, the *hardness* should be adjustable to be able to generate more diverse instances. While randomly assigning hard constraints may be possible, a more elaborate technique is proposed in Section 3.4.1.4.

3.4.1.1 Application Module

For the application generator module, applications are considered to be modeled as series parallel graphs (SPGs). In this way, a wide range of application characteristics can be described by one versatile encoding. An SPG is a fully connected graph that consists of series and parallel patterns[3-7]. While a series pattern can be used to model direct dependencies between two tasks (or subapplications) of an application, a parallel pattern models possible concurrent execution of two subapplications. As an example, consider applications A_1 and A_2 in Figure 3.1 on page 46. The former can be described as a parallel pattern where tasks t_2 and t_3 are able to execute concurrently (and in arbitrary order). That is, they are only dependent on their common predecessor task t_1. The latter application, on the other hand, represents a series pattern where task t_5, t_6, and t_7 have to be executed in a strict order as they form a dependency chain. Depending on the dominance of series or parallel patterns, the application allows for less or more concurrency, respectively.

An SPG is constructed recursively: Initially, an SPG has the form of a series (Figure 3.17(b)), parallel (Fig. 3.17(a)) or terminal (Fig. 3.17(c)) pattern. While terminal nodes determine the stop criterion of the recursive construction process, each node of the series and parallel patterns contains itself one of the patterns. Hence, nodes 1, 2, 3, and 4 in Figure 3.17 are in turn replaced by another pattern. Given a fixed number of series and parallel patterns to be generated, s and p respectively, this results in an overall number of $1+2\cdot s+3\cdot p$ tasks and $2\cdot s+4\cdot p$ messages.

The encoding of the series-parallel is depicted in Figure 3.18. At first, series and parallel patterns are inferred from the input `patterns/3` facts, defining the number of series and parallel patterns for application $A_i \in A$, viz. `NS`, `NP`, and `A`. Lines 3 and 4 define that both series and parallel are valid patterns. Afterwards, for each node of the parallel (line 5) and series (line 6) patterns, a child pattern is selected. That is, either another pattern or a terminal node

[3-7] Note that without loss of generality, we only consider binary parallel patterns in this paper. This can be easily extended to parallel patterns with more than two branches.

```
1  series(P,A)    :- patterns(NS,NP,A), P=1..NS.
2  parallel(P,A)  :- patterns(NS,NP,A), P=NS+1..NP+NS.
3  pattern(P,A)   :- series(P,A).
4  pattern(P,A)   :- parallel(P,A).
5  1 {contains(P1,P2,N,A) : pattern(P2,A), P1!=P2; contains(P1,term(N),N,A)} 1 :-
     ↪ parallel(P1,A), N=1..4.
6  1 {contains(P1,P2,N,A) : pattern(P2,A), P1!=P2; contains(P1,term(N),N,A)} 1 :-
     ↪ series(P1,A), N=1..3.
7  :- contains(P1,P2,X,A), contains(P1,P2,Y,A), X!=Y.
8  contains(P1,P2,A) :- contains(P1,P2,_,A).
9  :- contains(P1,P2,A), contains(P3,P2,A), pattern(P2,A), P1!=P3.
10 1{start(P,A) : pattern(P,A)}1 :- patterns(_,_,A).
11 reachable(P,A)  :- start(P,A).
12 reachable(P2,A) :- contains(P1,P2,A), reachable(P1,A).
13 :- not reachable(P,A), pattern(P,A).
14 contains_trans(P1, P2, A) :- contains(P1, P2, A), pattern(P2,A).
15 contains_trans(P1, P3, A) :- contains_trans(P1,P2,A), contains(P2, P3, A),
     ↪ pattern(P3, A).
16 :- contains_trans(P, P, A).
17 [...]
18 0 {instructions(task(TID,A),TYPE,N) :
     ↪ N=@getValue(instr_min,instr_max,seed,(TID,A,TYPE))} 1 :- task(TID,A),
     ↪ TYPE=1..instr_nr.
19 :- 0 #count {TYPE,(TID,A):instructions(task(TID,A),TYPE,_)} 0, task(TID,A).
20 instruction_exist(TYPE) :- instructions(_,TYPE,_).
21 :- not instruction_exist(TYPE), TYPE=1..instr_nr.
```

Figure 3.18: Application Generator Module

is selected. To prevent invalid pattern constructs, lines 7-16 contain rules to prohibit several decisions. Line 7 provides that no pattern is contained at two different positions of another pattern. Lines 8 and 9 prohibit the situation that a pattern is contained in two different patterns at the same time, while lines 10-13 guarantee that the graph is fully connected for each application. Finally, lines 14-16 provide the transitive closure of the graph and make sure no cycle is present (a pattern must not contain itself transitively).

Note that for sake of brevity, the deduction of output predicates `task/2` and `message/2` encoding tasks and messages for a specific application is not given in Figure 3.18. In short, each terminal node represents one task and the messages (including `send/2` and `receive/2` atoms) are deduced from the edges between terminal patterns. Lines 18 to 21 assign intermediate properties to each task that are subsequently used by the mapping generator. That is, each task is characterized by its individual instruction mix. Therefore, the generator expects a number of different instruction types as input, i.e., `instr_nr`. As encoded in line 18, each task may contain a specific number of instructions of each instruction type in the range from `instr_min` to `instr_max`. Again, `instr_min` and `instr_max` are inputs of the generator. As an example, a task may consist of 50 type-1 instructions (e.g., integer) and 20 type-2 instructions (e.g., floating point) but has no type-3 instructions (e.g., memory operations). Another task, on the other hand, may only consist of 30 type-2 instructions. The actual number of instruction of each type is selected randomly by a function implemented as a Python callback[3-8]. The 4-ary callback

[3-8]The results of callback functions are evaluated during grounding but is not part of standard ASP syntax. Instead, it is unique to the utilized ASP solver *clingo*.

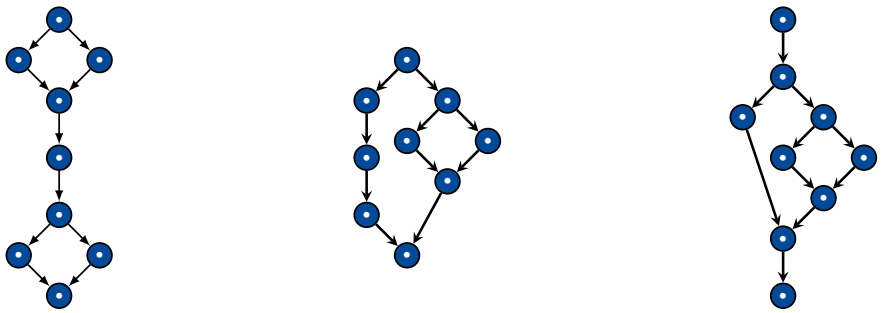


Figure 3.19: Three variant applications with similar characteristics. Each application contains one series and two parallel patterns, i.e., as input for the generator the atom `patterns(1,2,1)` was used.

function `getValue(min,max,seed,ID)` returns a random integer value between *min* and *max* based on the *seed*. To reduce grounding overhead, this is done only once for each task *ID*. Lines 19 and 20 guarantee that each task consists of at least one instruction type and that each instruction type is contained in at least one task throughout the application graph, respectively.

An example of three applications with similar characteristics is depicted in Figure 3.19. Here, the input `patterns(1,2,1)` is used, specifying to generate an application containing one series pattern, two parallel patterns and having the ID "1". Although, the exact characteristics vary between the three applications, the degree of parallelism is identical. Furthermore, each application consists of $1 + 2 \cdot 1 + 3 \cdot 2 = 9$ terminal nodes and $2 \cdot 1 + 4 \cdot 2 = 10$ edges.

3.4.1.2 Architecture Module

The hardware architecture module generates a 3-dimensional grid of routers that are each connected to a processing element via two independent communication links. An example of a resulting platform is given in Figure 3.1 on page 46. Here, the routers $r_1 - r_4$ form a grid of size $2 \times 2 \times 1$ and are connected to four processing elements $p_1 - p_4$. The encoding is shown in Figure 3.20. At first, the number and IDs of routers are determined by interpretation of the input atom `resources/3` that represents the grid size in each dimension X, Y, and Z. For each router, an accompanying processing element (`processor/4`) is inferred in line 2 before

```
1  router(ID,X,Y,Z) :- ID=X+NX*(Y-1)+NX*NY*(Z-1), resources(NX,NY,NZ), X=1..NX,
       ↪ Y=1..NY, Z=1..NZ.
2  processor(ID,X,Y,Z) :- router(ID,X,Y,Z).
3  router(ID) :- router(ID,_,_,_).
4  processor(ID) :- processor(ID,_,_,_).
5  link(router(N),processor(N)) :- router(N,_,_,_), processor(N,_,_,_).
6  link(router(M),router(N)) :- router(M,X,Y,Z), router(N,X+1,Y,Z).
7  link(router(M),router(N)) :- router(M,X,Y,Z), router(N,X,Y+1,Z).
8  link(router(M),router(N)) :- router(M,X,Y,Z), router(N,X,Y,Z+1).
9  link(X,Y) :- link(Y,X).
10 0{ cpi(processor(RID),TYPE,N) : N=@getValue(cpi_min,cpi_max,seed,(RID,TYPE)) }1
       ↪ :- processor(RID), TYPE=1..instr_nr.
```

Figure 3.20: Architecture Generator Module

```
1 N{ map(ID,task(T,A),processor(R)) : processor(R),ID=@getId(m,(T,A,R)) }N :-
    ↪ task(T,A), N=@getValue(map_min, map_max, seed,(T,A)).
2 :- map(ID, T, R), instructions(T,TYPE,_), not cpi(R,TYPE,_).
3 wcet(MID,TIME) :- map(MID, T, R),
    ↪ TIME=#sum{CYCLES,TYPE:instructions(T,TYPE,INS), cpi(R,TYPE,IPC),
    ↪ CYCLES=INS*CPI}.
```

Figure 3.21: Mapping Generator Module

both 4-ary predicates `router/4` and `processor/4` are projected to unary predicates in lines 3 and 4. Lines 5 to 9 define the communication links (`link/2`) between routers and processing elements. At first, a link is created between a router and its processing element having the same ID. Afterwards, in lines 6 to 8, a link between each two neighboring routers is added. Finally, the last link rule assures bidirectional links between elements. The last lines generate random cycles per instruction (CPI)[3-9] values for each processing element and instruction type. Similar to the task generated by the application module, the actual values are chosen randomly in the range of specified minimum and maximum values by a callback function. Although, a processing may not support a specific instruction type, each processing element must support at least one instruction type and each instruction type has to be supported by at least one processing element. As an example, assume two processing elements. While the first may represent a general-purpose CPU and, thus, supports any instruction type, the second one may represent an ASIC that only supports one type of instructions (e.g., AES operations).

Note that while the architecture generator presented here, can only generate NoCs organized in a regular mesh, it can be adapted easily to support other topologies. This does neither interfere with the application nor the mapping module.

3.4.1.3 Mapping Module

Given the application and architecture module, for the completion of a problem instance, mapping options have to be determined. To this end, in the mapping module, a random number of mapping options is generated for each task that has been created by the application module. Afterwards, timing requirements are determined based on instruction count, instruction type, and CPI values as generated in the application and architecture modules.

The encoding is given in Figure 3.21. Line 1 generates a random number of mapping options `map/3`. Therefore, two callback functions are evaluated during grounding. First, the function `getValue/4` works similar to the callbacks in the application and architecture modules and returns a random number ranging from `map_min` to `map_max`. It is used to determine the number of mapping options generated for a specific task. The binary callback function `getID/2` determines a unique identifier for each mapping option. It is based on the identifiers of the task, the resource it is mapped to, and the application. The first parameter `m` is the prefix of the mapping identifier. As an example, a mapping option allowing a task with ID 2 to be executed on a resource with ID 5 may result in the mapping ID “m1x2x5”. Line 2 guarantees that no mapping is inferred, that binds a task to an incompatible processing element. That is, if the processing element does not support a specific instruction type, no task containing such

[3-9] At system level, the timing is generally not cycle accurate. However, as the term *CPI* is commonly known, it is used here as a performance metric in a more general sense, i.e., *time units per instruction*.

instructions must be mapped to that resource. Finally, the WCET for a specific mapping option are calculated in line 3. Here, the instruction count of the task and the CPI of the resource are combined. For each present instruction type, the instruction count is multiplied with the corresponding CPI. Then, by accumulating the interim results, the WCET of the mapping option is determined. This way, the WCETs of different task-resource combination are more consistent than generating the values randomly. For instance, consider a generated mapping option $m = (t, p_i)$ The task t is characterized by an instruction count of 50, 100, 20, and 0 for the instruction types 1, 2, 3, and 4, respectively. The processing element p_i supports all instruction types and has corresponding CPIs of 2, 4, 3, and 1. Now, the resulting WCET of m corresponds to $2 \cdot 50 + 4 \cdot 100 + 3 \cdot 20 + 1 \cdot 0 = 560$. Another processing element p_j may not support instruction type 3, i.e., the atom `cpi(processor(pj),3,N)` is not part of the stable model. Thus, the mapping option $m = (t, p_j)$ is not viable due to the integrity constraint in line 2.

3.4.1.4 Constraint Generation

Generally, in order to evaluate system synthesis approaches, it becomes imperative to provide a large variety of test cases. To this end, the generator is used to generate different problem classes with varying characteristics. That is, each problem instance that belongs to a specific class has similar characteristics including the number of series and parallel patterns, the size of the architecture, and the range of property values. Utilizing each of the generated problem instances as input, the synthesis is carried out and can be evaluated with respect to varying problem instances.

In addition to the properties discussed above, real system implementations have to fulfill several constraints, like maximum power dissipation, timing, and cost requirements. One possible approach to deal with such demands in the generator is to utilize random numbers. However, a random constraint generation does not work properly as the problem instance would tend to either be over-constrained (i.e., there are no feasible solutions to be found) or under-constrained (i.e., every solution found during the search is feasible). Unfortunately, as the system synthesis problem is NP-complete, non-trivial upper bounds (and lower bounds) of constraints are hard to calculate.

As a remedy, the present generator utilizes a different approach. To allow the generation of instances of varying hardness, i.e., the difficulty of finding a valid implementation, a two-phased approach is proposed in the following. In a first step, a problem instance is generated as elaborated above. Here, no validity constraints or extra-functional constraints are specified. Hence, an implementation is assumed to be valid as soon as it is feasible. In a second step, each problem instance is synthesized a predefined number of times. The resulting implementations are then analyzed regarding their extra-functional properties. To create the final problem instances, this information is used to calculate validity constraints by weighting the analysis results with a user-specified complexity factor. The actual synthesis framework used to obtain these implementations is not significant. However, the obtained implementations should cover a diverse as possible area of the search space. A purely systematic approach, as employed by an ASP solver in its default configuration, is therefore not the optimal choice as small variations between two solutions generally only result in marginal changes of the properties. Instead, for generating the test instances for the following evaluations, the solver is configured in a way that a share of the decision variables are assigned randomly. Hence, the search does not strictly follow the CDCL approach. This way, the solver *jumps* more frequently through the search

Table 3.1: Problem instances for the synthesis benchmark

Group	$\|A\|$	series	parallel	$\|T\|$	$\|C\|$
I	1	2	4	17	20
II	2	5	5	27	30
III	3	9	7	42	46
IV	4	10	11	57	64
V	5	13	12	67	74

space and individual implementations are more diverse. The downside of this approach is that the search results in more conflict states which deteriorate the performance of the search. However, the experiments in the following show that the generation of the test instances only takes up a fraction of the time spent on the actual synthesis. As an alternative approach, the interim implementations may be synthesized by meta-heuristic approaches such as evolutionary algorithms (e.g., [92]). Yet, this is out of scope of the present thesis.

3.4.2 Experiments

The following section provides an experimental evaluation of the proposed synthesis framework with the focus on the partial assignment checking. Therefore, it presents the difference between two semantically equivalent solving methodologies. The first implements partial assignment checking enabled by the proposed tight integration of foreground and background solvers into one uniform framework. The second approach assumes loosely coupled solvers where the background propagators are only able to evaluate full assignments. Both approaches utilize an identical encoding and the basic framework presented in this chapter. In the remainder, first the experimental setup is presented before the results are discussed.

3.4.2.1 Experimental Setup

For the evaluation of the proposed approach, five sets of problem instances have been generated with the generator presented in Section 3.4.1. The characteristics of the problem instances are given in Table 3.1. As depicted, the problem instances of each group share common properties, i.e., the number of independent applications $|A|$, the degree of parallelism (viz. series and parallel pattern count), and the number of tasks and messages $|T|$ and $|C|$, respectively. In each group I – V, ten individual problem instances have been generated. Furthermore, the hardness of the resulting instances has been varied. Therefore, the complexity factor has been set to 1.00 (least hard), 0.90, 0.94, and 0.88 (most hard) to specify the validity constraint of the maximum allowed latency. Ultimately, this results in 200 individual problem instances that shall be synthesized once with the proposed partial checking approach and once with an equivalent full assignment checking approach. Hence, in total, 400 synthesis runs have been executed.

The full assignment approach uses the very same input files as the proposed approach. However, the implemented background propagators are only triggered as soon as the foreground ASP solver has decided the truth value of all decision variables. An early detection of invalid assignments is therefore not possible. However, the QF–IDL solver still provides minimal reasons for invalid designs. Thus, even with full assignment checking, the conflict clauses of the propagators can prune entire invalid regions (i.e., induced by negative cycles) from the search

space, but only at a later time. Although both approaches provide minimal reasons for invalid solutions, the traversal of the search space might differ. Depending on which negative cycle, and hence, conflict clause, is detected first, the CDCL search takes alternating paths through the design space. As a result, the actual acquired implementation[3-10] can vary between partial and full assignment approaches.

The direct comparison of the proposed synthesis framework with other approaches from the related work, such as [75, 92, 102] are not considered in the following. The reason for this decision is two-fold. First, the input models of different approaches differ from the proposed approach. This makes an adaption of the problem instances problematic as assumptions cannot be easily translated from one approach to the other. For example, the communication infrastructures in [75, 92] are modeled as busses while the proposed model considers a more elaborate NoC with on-chip routers. The work of Biewer et al. [102] assumes acyclic applications while the proposed approach can model cyclic behavior in the application. Second, the utilization of different solving techniques inevitably leads to different paths through the search. For example, in [102], an external QF–IDL (respectively ILP) solver is used to determine validity of potential implementations. Together, these differences potentially mask the effects that shall be studied by the experiments in the following. Hence, the setup is chosen to particularly compare the differences of full and partial assignment checking.

Each synthesis run has been executed on an Intel Core i7-4770 at 3.4 GHz with 32 GiB main memory running Ubuntu 16.04.7 LTS. The timeout for each run was set to 1800 s.

3.4.2.2 Results

The results of the synthesis runs are depicted in Table 3.2. Here, each degree of hardness is contemplated individually and further subdivided by the different specification groups as explained above. The full and partial assignment approaches are represented by four columns. The first column indicates the number of instances that could be solved within the 1800 s time limit. In the second one, the average time required to solve each individual problem instance is shown, i.e., the time after which a valid implementation has been found by the synthesis framework. The last two columns state the average numbers of choices made and conflicts encountered by the foreground ASP solver, respectively. Note that for the calculation of the time, choices, and conflicts, only the synthesis runs were considered for which a valid implementation was found (or the instance was proven to be unsatisfiable). For example, in the first row, the average solving time for the full assignment approach only considers six problem instances, while in the partial assignment approach, nine runs are considered. A more detailed summary of the results can be found in Tables A.1 and A.2 in the appendix where all runs are considered for the calculation.

The most apparent result shown in Table 3.2 is the difference in the number of solved instances between full and partial assignment approaches. While the proposed approach utilizing partial assignments finds valid implementations for 92.5 % (185 out of 200) of the problem instances, the full assingment approach can only solve 24 % (48 / 200) within the specified time limit. Furthermore, the average time required by the partial assignment approach is almost always less than for the other approach. There are a few exceptions to this trend: group I instances, and group III instances with a hardness of 0.88. However, these are only artifacts due

[3-10] Note that the synthesis stops after the first valid implementation is found. Not stopping the search will eventually provide the exact same set of implementations for both approaches.

Table 3.2: Results of the test instances for the full and partial assignment checking synthesis runs. The individual columns represent the mean values of all *solved* instances.

		Full Assignments				Partial Assignments			
Hardness	Group	Solved	Time [s]	Choices	Conflicts	Solved	Time [s]	Choices	Conflicts
1.00	I	6	0.319	$7.83 \cdot 10^3$	$1.38 \cdot 10^3$	9	1.045	$1.82 \cdot 10^4$	$9.87 \cdot 10^3$
	II	9	463.117	$2.33 \cdot 10^7$	$2.64 \cdot 10^5$	10	135.955	$1.17 \cdot 10^6$	$9.73 \cdot 10^5$
	III	6	8.654	$6.35 \cdot 10^4$	$5.67 \cdot 10^1$	10	4.723	$1.25 \cdot 10^4$	$4.81 \cdot 10^3$
	IV	1	28.017	$9.24 \cdot 10^4$	$7.30 \cdot 10^1$	10	9.038	$1.28 \cdot 10^4$	$1.40 \cdot 10^3$
	V	0	–	–	–	10	20.350	$2.81 \cdot 10^4$	$2.27 \cdot 10^3$
0.94	I	5	0.901	$3.97 \cdot 10^4$	$3.76 \cdot 10^3$	9	65.610	$8.40 \cdot 10^5$	$7.14 \cdot 10^5$
	II	5	730.881	$3.55 \cdot 10^7$	$4.16 \cdot 10^5$	10	100.205	$8.23 \cdot 10^5$	$6.60 \cdot 10^5$
	III	3	6.775	$5.75 \cdot 10^4$	$5.70 \cdot 10^1$	10	5.211	$1.46 \cdot 10^4$	$4.88 \cdot 10^3$
	IV	1	16.601	$6.87 \cdot 10^4$	$4.00 \cdot 10^1$	10	4.939	$8.97 \cdot 10^2$	$2.78 \cdot 10^1$
	V	0	–	–	–	10	30.128	$4.88 \cdot 10^4$	$5.25 \cdot 10^3$
0.88	I	4	186.428	$1.57 \cdot 10^7$	$1.80 \cdot 10^5$	9	222.738	$4.18 \cdot 10^6$	$3.58 \cdot 10^6$
	II	3	358.781	$1.70 \cdot 10^7$	$1.92 \cdot 10^5$	8	243.216	$1.78 \cdot 10^6$	$1.53 \cdot 10^6$
	III	3	7.570	$5.23 \cdot 10^4$	$5.37 \cdot 10^1$	10	25.021	$1.06 \cdot 10^5$	$5.24 \cdot 10^4$
	IV	0	–	–	–	10	5.101	$1.31 \cdot 10^3$	$6.90 \cdot 10^1$
	V	0	–	–	–	10	32.800	$6.30 \cdot 10^4$	$8.27 \cdot 10^3$
0.83	I	1	4.027	$2.13 \cdot 10^5$	$1.32 \cdot 10^4$	5	346.815	$4.58 \cdot 10^6$	$4.07 \cdot 10^6$
	II	0	–	–	–	5	310.054	$2.60 \cdot 10^6$	$2.12 \cdot 10^6$
	III	1	6.560	$5.04 \cdot 10^4$	$8.60 \cdot 10^1$	10	99.586	$4.80 \cdot 10^5$	$3.56 \cdot 10^5$
	IV	0	–	–	–	10	10.196	$1.54 \cdot 10^4$	$2.23 \cdot 10^3$
	V	0	–	–	–	10	73.721	$1.53 \cdot 10^5$	$1.81 \cdot 10^4$
Total		48				185			

to the utilized calculation methodology. As explained above, only solved instances are considered here. Hence, if a specific instance cannot be solved by the full but can be solved by the partial approach, the solving time is influenced in the latter. This is the case for all the outliers. For example, one particular instance has been solved by the partial assignment approach in around 7 s while it could not be solved by the full assignment approach. In fact, each specific problem instance has been solved faster with the proposed approach.

A second observation regards the complexity of the problem instances. The complexity of the problem instances within a hardness category, i.e., the amount of decision variables induced by the number of tasks and messages, grows with an increasing group ID. However, the results show that this does not necessarily hold for the solving complexity of the problem instances. Especially, this becomes apparent for the problem instances of group II. On average, this group takes significantly more time than the other problem instances, even if they are composed of much more tasks and messages. This clearly shows that the characteristics alone is not decisive to categorize a problem instance as more or less complex to solve. Instead, other factors may play a major role such as the particular mapping options as well as routing decisions that have to be taken and the heuristics used by the solver. That is, if an unfavorable heuristic is chosen, the design space is traversed ineffectively and valid regions may not be found early. Hence, altering the heuristic can lead to better, but also worse, solving times for different instances. While a different heuristic may, on the one hand, be beneficial for finding an initial solution, on the other hand, this often does not hold if the design space is to be traversed completely, e.g., to prove unsatisfiability or perform a DSE (q.v. Chapter 4). The detailed consideration of heuristics is,

however, out of scope of the present thesis and the interested reader is referred to [136].

Furthermore, altering the hardness of the problem instances yields noticeable changes. Note again that the problem instances in the different hardness categories are identical but with varying strict validity constraints. In general, the smaller the latency threshold for a valid design is, the fewer instances can be solved within the given time limit. While this behavior is more prominent for the full assignment approach, where only 20 % of all problem instances have been solved in the 0.88 category, the number of solved instances by the partial assignment approach also decreases slightly. However, the average solving time does not strictly follow that trend. The only problem instances that require more solving time on average with an increasing hardness are the ones of group I. For remaining problem instances, the influence on the solving time appears somewhat random. For example, the problem instances of group II have been solved faster (by the partial approach) in the 0.94 category than in the previous two. Again, this behavior can be explained by the chosen path through the decision tree. Consider the necessary schedulability test as defined in Equation (3.2) that can be easily encoded directly in ASP. If the accumulated WCETs of a partial binding already exceeds the specified periodicity on any allocated processing element, the assignment can be pruned from the search. This happens earlier when the constraints are stricter. Hence, the partial binding is already excluded by the ASP solver and a different search path is taken. Ultimately, this change steers the solver farther away from the invalid region earlier. Note, however, that this does not hold for other problem instances or even stricter constraints as shown for category 0.88 instances, where for two problem instances, no valid implementation has been found at all. More restrictive validity constraints do not generally translate to more complex searches. On the one hand, it may help in finding better solutions faster but, on the other hand, it may also lead to more timeouts, depending on the specific problem instance.

As a conclusion, the experiments show a clear advantage of the partial assignment checking for the synthesis at ESL. In particular, the number of solved problem instances within the given time limit of 3600 s is nearly four times higher than with the conventional approach. Furthermore, even the solving time of the individual instances is usually lower. While the advantage of partial assignment checking is less apparent for small problem instances, large instances significantly profit from the proposed methodology.

3.5 Chapter Summary

This chapter proposed a systematic approach to the system synthesis problem, i.e., the transformation of a behavioral specification into a structural implementation that has to adhere to several feasibility and validity constraints. The considered specification model provides for the definition of cyclic, deadline-constraint, and periodic applications, interconnected hardware platform templates, and mapping options connecting tasks and processing elements. The novelty of the approach is the tight integration of feasibility and validity checks into one uniform encoding. This is enabled by the use of ASPmT. It allows for the exchange of common variables between specialized solvers. In particular, timing constraints are hard to verify with Boolean logic as it would result in a further design space explosion that is dependent on the specific timing constraint. Instead, QF–IDL can determine a schedule in polynomial time and be used to automatically verify timing constraints. Furthermore, the solving process of QF–IDL automatically provides a minimal reason whenever an implementation candidate is identified to be invalid. Therefore, the solver determines the schedule on the basis of a constraint graph and

a shortest-path algorithm. If the latter detects a negative cycle, the involved nodes represent a minimal reason for the unsatisfiability of the problem instance. The tight integration of foreground ASP and background QF–IDL solvers further permits the evaluation of partial assignments. That is, even if the ASP solver only has been assigned a subset of all decisions, the background propagator can evaluate whether it is still possible to acquire a vaild schedule. In case a negative cycle is already detected for the partial assignment, the remaining decision variables do not have to be assigned further. Instead, a conflict clause, excluding the partial assignment only, is provided early during search and allows pruning entire invalid regions from the search space. Note that this is only possible if the problem is assignment monotonic, i.e., deciding subsequent variables must not improve the properties of the solution. In this thesis, this is enforced by the utilization of introduced priorities to the tasks that prevent scheduling anomalies.

The proposed framework is finally evaluated through a total of 200 randomly generated problem instances, categorized into sets of ten instances with common properties. Therefore, a modular instance generator, also based on ASP, is proposed that allows the generation of series-parallel applications that shall be implemented onto regularly interconnected processing elements implementing a NoC. The framework differs from related work mainly by tightly coupled foreground and background solvers that allow the checking of partial assignments. To avoid effects that potentially mask the benefits of the partial assignment checking, a direct comparison to related work has not been conducted. Instead, specifically the influence of the partial assignment has been evaluated. The results clearly show the benefit of the proposed approach over a synthesis where only full assignments can be analyzed, e.g., when the schedulability analysis is outsourced to an external integer difference logic (IDL) solver, as in [102]. While the proposed approach is able to synthesize more than 90 % of the specified problem instance, the synthesis analyzing full assignments is incapable to determine valid implementations of more than 75 % of the problem instances. Especially, problem instances with strict timing constraints provide a significant challenge if only full assignments can be checked.

4
Symbolic Design Space Exploration

The previous chapter addressed the problem of finding one valid solution to the system synthesis problem. From a given specification consisting of an application, an architectural hardware template, a set of mapping options and their corresponding properties, an allocation, a binding, and a schedule is derived. The resulting implementation adheres to given functional requirements but extra-functional requirements are not considered. In reality, extra-functional properties additionally contribute to the quality of the system and are used to compare different valid implementations with one another. In addition to timing properties, commonly evaluated properties are power consumption, production costs (which are mainly impacted by area requirements), reliability, interoperability, versatility, and others. The classification into functional and extra-functional properties as well as their individual importance is highly application-specific. While, for example, the power that is consumed by a system is generally not considered to be functionally important, it becomes essential if only a limited power source (e.g., battery-powered system, implants) is available. Even when two properties are considered extra-functional, they may be of different importance to the quality of the entire system. Timing properties, for example, contribute more to the quality of a high-performance computing cluster than its monetary costs. However, monetary costs play a more important role in small-scale embedded systems.

This chapter, however, pursues a more general approach where each specified objective is considered to be equally important. Hence, the aim is to find an implementation that is optimal with respect to the whole set of objectives. Optimizing multiple objectives simultaneously, generally prevents the existence of one optimal implementation as individual objectives often conflict with each other. For instance, an implementation providing a high performance prevents power or cost-efficient designs and a small area design may reduce reliability as a failure of a component cannot be compensated. As outlined in Section 2.3 of this thesis, the result of a multi-objective optimization run is therefore a set of compromise implementations organized in a Pareto-front. Optimally, the entire design space is explored to guarantee finding all Pareto-optimal candidates. Even though this chapter deals with the exploration at system-level, a complete search is normally only possible for small-sized systems in reasonable time. In other cases, only an approximate Pareto-front can be obtained.

In addition to the challenges above, the development of (embedded) systems is a highly competitive area where the timeframe for new designs is critical. Hence, an efficient design approach is mandatory to provide for a low time-to-market. This is best achieved through automated approaches that reduce the necessity of manual interaction to a minimum and the realization of the exploration at a preferably high level of abstraction. At system-level, the most influential design decisions can be weighed effectively without the need for detailed structural information of the individual components. The goal is to find the design properties

of all potential design candidates and provide alternatives to the decision maker (DM) to select candidates for further investigation at lower abstraction levels.

To summarize, an optimal DSE with multi-objective optimization should provide for the three following properties:

- Provide for the ability to define both the specification of the system and arbitrary design objectives.
- Automatically explore the search space for feasible design candidates and evaluate them with respect to the desired objectives.
- Generate a diverse approximation of the true Pareto-optimal front that contains design alternatives to be selected by the DM and investigated at lower abstraction levels.

To this end, the remainder of this chapter is separated into three sections. First, the ASPmT-based approach presented in Chapter 3 is extended towards a multi-objective DSE. In comparison to previous related work (e.g., [102]), the optimization is tightly integrated into the solving process by the development of an additional background propagator. The utilization of ASPmT provides the same benefits as in the synthesis, i.e., implementations of highly constrained specifications are guaranteed to be found if they exist and design points are not unnecessarily revisited once found. Furthermore, it enables dominance checks of partial assignments and allows the pruning of non-optimal areas of the search early in the decision process through effective reasoning. Finally, the integrated encoding of background theories and individual propagators provides succinct formulation of the multi-objective optimization problem. In the course of the first section, various alternative exploration techniques are investigated that show the versatility of the approach.

As dominance checks are already executed on partial assignments, the demands on the archive holding the non-dominated design candidates increase as these checks have to be executed more often when compared to conservative approaches. To this end, the second section proposes an archive management technique based on the Quad–Tree data structure [106]. It is shown that intermediate dominance checks of partial assignments do not require a complete dominance check. Hence, a significant portion of the time can be saved when adapting the checking mechanism accordingly without sacrificing correctness of the approach.

Besides the search for novel designs and the dominance filtering of valid design candidates, the evaluation process is the third bottleneck in the DSE. Each design candidate has to be evaluated with respect to its objectives to generate the input for validity and Pareto filters. Similar to the archive management, the utilization of partial assignments increase the number of evaluations significantly for each design candidate. Hence, the final section proposes a remedy to this problem. Here, safe approximations are utilized in a way that allows a much faster evaluation performance of the specified objectives. The goal is to provide a methodology that allows for a fast evaluation of a partial assignment while maintaining the correct Pareto-front in the final archive. To this end, the objective value of each partial assignment is estimated through a safe approximation until the full assignment is obtained and evaluated accurately. While this technique requires an objective function to fulfill certain properties to guarantee correctness, it shows a significant performance improvement in the evaluation of the end-to-end latency of an implementation.

Relevant Publications

[3] Kai Neubauer, Christian Haubelt, Philipp Wanko, and Torsten Schaub. "Utilizing Quad-Trees for Efficient Design Space Exploration with Partial Assignment Evaluation". In: *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jeju, Korea, Jan. 2018, pp. 434–439. DOI: 10.1109/ASPDAC.2018.8297362.

[4] Kai Neubauer, Philipp Wanko, Torsten Schaub, and Christian Haubelt. "Exact Multi-Objective Design Space Exploration using ASPmT". In: *Design, Automation and Test in Europe Conference (DATE)*. Dresden, Germany, Mar. 2018, pp. 257–260. DOI: 10.23919/DATE.2018.8342014.

[6] Kai Neubauer, Christian Haubelt, Philipp Wanko, and Torsten Schaub. "Work-in-Progress: On Leveraging Approximations for Exact System-level Design Space Exploration". In: *International Conference on Hardware Software Codesign and System Synthesis (CODES/ISSS)*. Sept. 2018, pp. 1–2. DOI: 10.1109/CODESISSS.2018.8525974.

[7] Kai Neubauer, Benjamin Beichler, and Christian Haubelt. "Exact Design Space Exploration Based on Consistent Approximations". In: *Electronics* 9.7 (June 2020), p. 1057. ISSN: 2079-9292. DOI: 10.3390/electronics9071057.

[8] Christian Haubelt, Kai Neubauer, Torsten Schaub, and Philipp Wanko. "Design Space Exploration with Answer Set Programming". In: *KI - Künstliche Intelligenz*. Vol. 32. 2-3. Berlin Heidelberg: Springer Nature, May 2018, pp. 205–206. DOI: 10.1007/s13218-018-0530-3.

4.1 Search Space Pruning Through Pareto Filtering

In order to cope with the ever-increasing complexity of embedded systems, system-level descriptions are utilized to diminish the complexity of finding potentially good solutions which can then be used as initial starting points for further optimization in lower abstraction levels. At system level, applications are composed of communicating tasks while the hardware architecture contains heterogeneous processing elements (e.g. CPU, DSP, GPU) as well as a communication infrastructure like routers and links. Depending on the decisions that have been made, the qualitative properties of the resulting system implementation may vary considerably from solution to solution resulting into a multi-objective optimization problem (MOOP). Note that, without loss of generality, the decisions made during the search are evaluated through the three objectives latency, energy consumption, and area requirements. These three properties are commonly used to classify the quality of an implementation. Although other objectives such as reliability are not considered directly, the presented framework can be easily adapted to support additional propagators and objective functions.

4.1.1 Exploration Model

The exploration model is based on the system model presented in Section 3.1. Hence, the input is structured as a specification graph $\mathcal{G}_S = (\mathcal{G}_A, \mathcal{G}_P, M, \mathcal{F}_S)$ consisting of an application graph $\mathcal{G}_A$, a platform graph $\mathcal{G}_P$, a set of mapping options M, and a set of functions $\mathcal{F}_S$ assigning properties to each of the elements. Yet, to allow for the evaluation of the additional energy and area requirements objectives, the specification has to be extended with respective

properties. This primarily affects the architecture graph and the mapping options, while the content and constraints regarding the application graph are unchanged. In the architecture graph, the functions $c : V_P \to \mathbb{N}$ and $E_{stat} : V_P \to \mathbb{N}$ are added to the function set $\mathcal{F}_P$. These assign area costs as well static energy requirements to the computational and communication elements of the graph, respectively, that have to be considered if a resource is allocated in the implementation. Regarding dynamic energy consumption, the function $E_{dyn} : M \to \mathbb{N}$ is added to the function set $\mathcal{F}_S$ in the specification graph. Hence, whenever a certain mapping $m = (t, p)$ is selected, the execution of task t on resource p consumes the specified amount of energy. Note that the functions E_{dyn} and E_{stat} model energy instead of power. In the former function, the execution time is known according to the worst-case execution time defined by the function w. Thus, the energy of an individual mapping option can be calculated at specification time. In the latter case, a resource is allocated for the entire time period of the execution. As the model considers a periodic execution of the application, and the periodicity of the applications is specified initially (or calculated through the hyper-period), the values assigned to each element are considered to be valid for one iteration. Furthermore, similar to the specification of individual execution times, different dynamic energy consumption assigned to individual mapping options model heterogeneity in the hardware platform.

The ASP encoding of the novel properties is conducted similarly to those introduced in Chapter 3. Thus, the functions c, E_{dyn}, and E_{stat} are encoded through the binary predicates `area/2`, `dynamicEnergy/2`, and `staticEnergy/2`. In each case, the first parameter identifies the affected elements, i.e., a particular mapping option identifier in the former predicate and a resource identifier in the latter two predicates. The second parameter defines the respective value.

A feasible implementation to a given specification as defined above is evaluated through the objective functions *latency*, *area*, and *energy*. The values of the respective objectives are dependent on the actual decisions made regarding binding, routing, and scheduling. They constitute soft constraints that are to be optimized during the DSE. Without loss of generality, the DSE is formulated as a minimization problem as follows:

$$\begin{aligned}&\textbf{minimize } f(x) = (latency(x), area(x), energy(x)),\\ &\textbf{subject to:}\\ &\qquad x \text{ is a feasible system implementation,}\end{aligned}$$

where a feasible system implementation is a solution that adheres to all given mapping, routing, and timing constraints. Hence, the result of the DSE is, as outlined in Section 2.3, a set of Pareto-optimal solutions. The next section elaborates on the ASPmT-based optimization framework used to solve the multi-objective minimization problem.

The area cost yields the accumulated area requirements of every allocated device, which is calculated as:

$$area(x) = \sum_{d \in \alpha_D} area(d), \tag{4.1}$$

with α_D denoting the allocated devices, i.e., computational and communicating resources. Devices that are not allocated in the mapping and routing are assumed to be not implemented at all and thus, do not increase the area requirements. Accordingly, the indicator variables that are exchanged between ASP and background theory only need to contain the area requirements of the devices.

The calculation of the energy consumption $E(x)$ is separated into the static E_{stat} and the dynamic part E_{dyn}.

$$E(x) = E_{stat}(x) + E_{dyn}(x) \tag{4.2}$$

$$E_{stat}(x) = \mathcal{P} \cdot \sum_{d \in \alpha_D} P_{stat}(d) \tag{4.3}$$

$$E_{dyn}(x) = \sum_{m \in \beta} E_{dyn}(m) + E_{trans} \cdot \sum_{c \in C} hops(c) \tag{4.4}$$

Again, the static energy arises from the sum of individual power requirements of allocated devices multiplied with the periodicity of the system. That is, the energy is calculated per iteration. The dynamic energy stems from the selected mapping and routing options. As defined above, the required dynamic energy is associated to each mapping option and is accumulated to the overall dynamic mapping energy. Furthermore, a message packet that is transferred over the network also adds to the dynamic energy. Thus, the entire energy consumption of each message is dependent on the length of the route it takes from the sending to the receiving device which is obtained by the function $hops : C \mapsto \mathbb{N}$. Note that we assume homogeneous links and routers here, such that the number of hops can be multiplied with the defined value E_{trans}, encoded by `routingenergy/1` atoms. The number of hops per message are calculated by the ASP solver through integrated aggregate atoms that count the number of `reached` atoms.

Finally, the latency of an implementation is arises from the maximum end time of all tasks, i.e., the start time of each task obtained through QF–IDL and the WCET of its binding.

$$latency(x) = \max_{m=(t,r) \in \beta} \tau(t) + w(m) \tag{4.5}$$

4.1.2 Optimization Framework

The general overview of the proposed DSE is depicted in Figure 4.1 and essentially consists of three pillars – the ASP solver clingo in the foreground theory, a set of theory propagators, and an optimization propagator in the background theory. Similar to the synthesis framework discussed in the previous chapter (q.v. Figure 3.14), the foreground theory employs the ASP solver clingo. Thus, a general problem definition and a specific problem, both encoded through ASP rules, constitute the input of the framework. The input is first grounded into a variable-free representation which is used to initialize the background theory propagators and lays the foundation for the foreground ASP solver clingo. The ASP solver utilizes a CDCL-based solving strategy to explore the search space by assigning and propagating decisions while the background theory checks these solutions for validity and optimality. In comparison to the synthesis framework, the background theory is split into two consecutive parts while the foreground theory as well as the input language remains similar with marginal changes necessary to encode additional objectives. The theory propagators, as the first part of the background theory, handle the evaluation of individual objectives. This includes the QF–IDL–based latency calculation detailed in Section 3.2.2 but is extended by further propagators handling the evaluation of the system with respect to area and energy requirements. After evaluation, the theory propagators check whether the implementation adheres to given hard constraints such as a maximum end-to-end latency or area requirements. If an implementation candidate fails this check, the framework generates a conflict clause, excluding the conjunction of decisions made from the search space. Note that the theory propagators work on partial assignments. Hence,

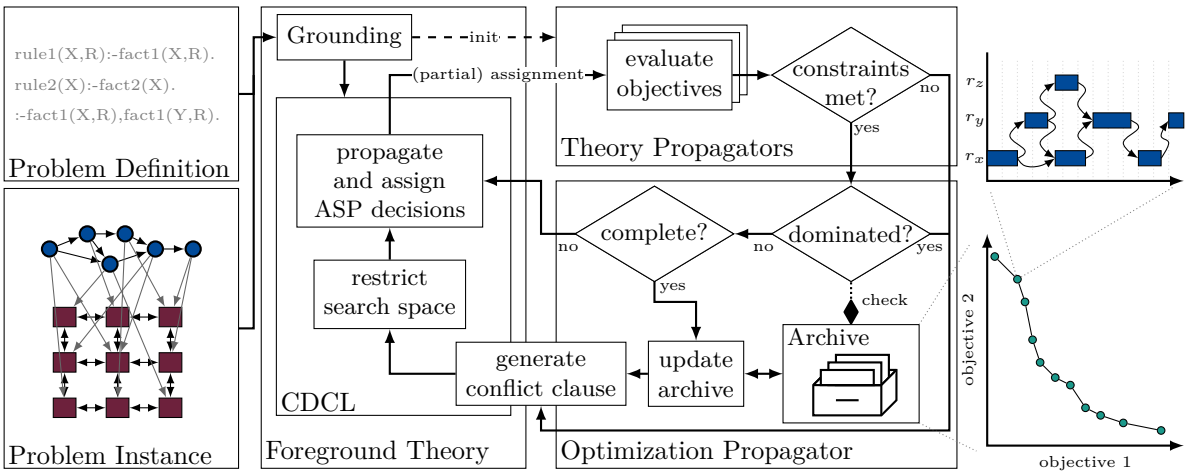
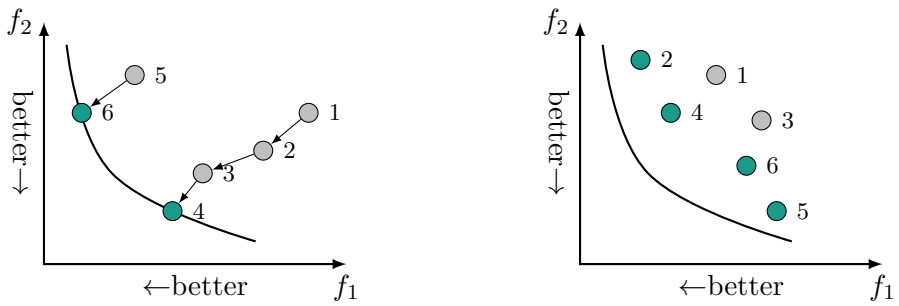


Figure 4.1: Overview of the Design Space Exploration Framework working with partial assignments. The theory propagator evaluates the assignments with respect to validity while the optimization propagator takes control of dominance checks and manages the archive. At any time, the archive contains all Pareto-optimal found so far.

a set of infeasible decisions is potentially detected early during the search allowing for a more effective pruning of entire areas of the search space.

After a positive feasibility check, the implementation candidate is relayed to the optimization propagator. The optimization propagator is realized in a two-step principle. First, it performs a dominance check of the novel implementation candidate with respect to previously found, non-dominated solutions that are saved in the archive. Again, if the partial candidate is already dominated by another solution in the archive, a conflict clause is generated, and the search space is pruned. Subsequently, if the dominance check has been passed, the propagator checks whether the assignment is already complete. If not, the search is continued by the foreground theory. Otherwise, only now, the archive is updated with the novel implementation, i.e., dominated solutions are removed from the archive and the current implementation candidate is added to the archive. A final conflict clause is generated that prohibits finding the same solution again.

This process is repeated until either all possible solutions are explored or an abortion criterion has been reached. In the former case, the archive contains the true Pareto-optimal front of possible implementation candidates. Although the optimality cannot be guaranteed in the latter case, the archive also only contains mutually non-dominated implementation candidates. They constitute an approximation of the true Pareto-optimal front. In comparison to conservative meta-heuristic approaches, the main advantages of a complete approach is its ability to prove the infeasibility of a given specification. While a meta-heuristic does not track whether a specific design point has already been explored, the proposed ASPmT-based approach does not explore the same design point twice. This eventually results in the termination of the search and can be used to prove that the constraints imposed on the specification are not satisfiable.



(a) Depth-First **(b)** Breadth-First

Figure 4.2: Optimization Strategies

4.1.2.1 Optimization Strategies

In general, there are two exploration strategies that can be employed when searching for the Pareto-optimal set of implementation candidates. A depth-first strategy, as for example implemented by the ASP-based preference framework *asprin* [74], primarily searches for solutions that improve the convergence of the archive towards the true Pareto-front. After finding an initial implementation x_i, the search engine requires the next solution x_j to strictly dominate the previous one, i.e., $x_j \succ x_i$. Incomparable solutions to x_i, i.e., $x_i \parallel x_j$, are ignored until the engine cannot find a strictly better implementation. A schematic sketch of the search is depicted in Figure 4.2(a). The initial implementation, marked by 1, is only replaced by solution 2, 3, and eventually 4. As solution 4 is situated on the true Pareto-front, visualized as solid line, the search engine cannot find a strictly better implementation. Afterwards, it starts searching for an incomparable solution, resulting in implementation 5, followed by 6. Ultimately, only solutions 4 and 6 remain in the archive of non-dominated solutions and are simultaneously proven to be Pareto-optimal. In contrast, a breadth-first strategy does not require a subsequent solution to be strictly better than the previously found one. Instead, each non-dominated solution, i.e., either dominating or incomparable, is saved in the archive. Consider Figure 4.2(b) as an example. Implementation 2 is saved to the archive as an incomparable solution to 1. Subsequently, solution 3 is incomparable to both 1 and 2. As soon as solution 4 is found by the search engine, the first solution is ousted as it is now dominated. Finally, solutions 5 and 6 are found, with 6 dominating and replacing solution 3. Ultimately, the archive consists of solutions 2, 4, 6, and 5. Unless the entire search space has been explored, none of these solutions is proven to be Pareto-optimal. Both the breadth-first and the depth-first strategies are complete. Hence, the true Pareto-front is returned eventually. However, when only a fraction of the search space is explored, they show different behavior. As indicated in Figure 4.2, a depth-first strategy may converge faster to the true Pareto-front while a breadth-frist strategy finds diverse solutions more frequently.

Based on the two search strategies, in the following, three deduced implementation variants are proposed. First, a hybrid approach ($\mathcal{H}$) employs the preference framework asprin [74] to evaluate linear objectives directly using the ASP solver clingo. Only non-linear objectives (such as latency) are relayed to a specific background theory to be evaluated. This approach conforms to a depth-first search strategy. Each time an implementation candidate is found, the framework adds a constraint to steer the search towards a strictly better solution. Considering

```
1: function OPTIMIZE(problem Instance I, problem definition E, n)
2:     GP ← GROUND(E ∪ I)
3:     TO, PO ← INSTANTIATE(ANALYZE(GP))
4:     o ← 0; i ← 0; S ← ∅; unsat ← False
5:     while True do
6:         ret ← SOLVE(GP, TO, PO)
7:         if ret = Unsatisfiable then
8:             if i = 0 or unsat then
9:                 return S
10:            S ← S ∪ GET_SOLUTION(i)
11:            o ← o + 1
12:            if o = n then
13:                return S
14:            unsat ← True
15:            REMOVE_COMPARE_SOLUTION(GP, PO, i)
16:            OPTIMUM_FOUND(GP, PO, i)
17:        else
18:            unsat ← False
19:            i ← i + 1
20:            SAVE_SOLUTION(ret, i)
21:            if i > 1 then
22:                REMOVE_COMPARE_SOLUTION(GP, PO, i − 1)
23:            ADD_COMPARE_SOLUTION(GP, PO, i)
```

Figure 4.3: Exploration algorithm used by the optimization framework.

a two-dimensional minimization problem ($\min(f_1(x), f_2(x))$), for example, an implementation candidate x_i with the objective vector ($f_1(x_i) = 5, f_2(x_i) = 5$) causes the addition of the two constraints $:- f_1(x) \geq 5$ and $:- f_2(x) \geq 5$. Thus, a subsequent implementation x_j must evaluate better than 5 in both objectives. If the additional constraints renders the problem unsatisfiable, the previously found solution is proven to be Pareto-optimal. It is saved via facts within the logic program and the search is restarted for incomparable solutions. To this end, the constraints added previously are replaced by the less restrictive constraint $:- f_1(x) \geq 5, f_2(x) \geq 5$.

The other two variants do not use the asprin framework to evaluate the linear objective. Instead, all objective evaluations and the dominance checks execute in specialized background propagators. Non-dominated implementation candidates are saved explicitly in an archive. Within the two *theory*-based approaches, one conforms to a depth-first search ($\mathcal{T}_{depth}$) while the other follows a breadth-first search ($\mathcal{T}_{breadth}$). Hence, the former approach behaves similar to ($\mathcal{H}$) such that it adds constraints upon finding an implementation candidate to restrict the search to strictly better solutions. However, as the dominance checks are managed explicitly through an archive, Pareto-optimal designs are not saved via rules and no constraints are added after finding a Pareto-optimal implementation. Dominated solutions are discovered throughout the exploration. The latter approach ($\mathcal{T}_{breadth}$) works similarly but does find non-dominated (i.e., including incomparable) solutions throughout the search.

4.1.2.2 Optimization Algorithm

All three optimization strategies $\mathcal{H}$, $\mathcal{T}_{depth}$, and $\mathcal{T}_{breadth}$ use variants of the algorithm presented in Figure 4.3. Essentially, it consists of an initialization phase and a loop constantly searching for optimal solutions. The input parameters of the algorithm are the problem instance I, the problem definition E, and an integer n. The parameter n represents the maxi-

mum number of optimal solutions to be found where $n = 0$ signifies to find all optimal solutions. The main difference in the actual implementation of the algorithm lies in the semantic of an optimal solution. In the hybrid approach $\mathcal{H}$, each Pareto-optimal design point is considered to be an optimal solution as after each unsatisfiable solving step, the search is restarted to find further solutions. In contrast, in the theory-based approaches $\mathcal{T}_{depth}$ and $\mathcal{T}_{breadth}$, the whole set of Pareto-optimal solutions is considered to be one single optimal solution. Here, the solution archive is stored and managed outside the ASP solver clingo and the search is steered by adding constraints that trigger the backtracking mechanism of CDCL-based search.

The algorithm first grounds the combined logic program of E and I (Line 1). The optimization parameters contained in the resulting ground program GP are analyzed, and the necessary theory TO and optimization background propagators PO are instantiated (Line 2). Elements of TO and PO are background propagators [42] that are called during solving to ensure that the current partial assignment is, first, consistent with the theories, and second, not worse than the previous solution. After the initialization of the state variables, i.e., current number of optimal solutions o, number of solutions found i, the set of optimal solutions $\mathcal{S}$ and the Boolean variable *unsat* to indicate unsatisfiabilty, the main loop starts by trying to find a new solution (Line 6). To this end, as described in Chapter 3, the ground logic program is solved by the ASP solver while constraint and dominance checks are constituted in the background theory propagators TO and PO, respectively. The returned value of one solving call equates either to *Unsatisfiable* or contains a solution that is better than the previous one and incomparable to already found optimal solutions. In the former case, i.e., if no solution could be found (lines 7–16), three possibilities arise: first, the original problem is unsatisfiable, viz. $i = 0$, second, it was proven that no further optimal solutions exist, viz. $unsat = True$, and third, a new optimal solution was found. In the first two cases, the algorithm stops and the set of optimal solutions $\mathcal{S}$ is returned (Line 9). In the latter case, solution i is optimal and is added to $\mathcal{S}$ and the number of optimal solutions o is incremented by one. If o equals n, the desired number of optimal solutions is returned. To save the information that no intermediate solution has been found yet, *unsat* is set to *True*. Since the recent solving step could not find a new solution that is better than solution i, the comparison to solution i is removed from the ground program and the optimization propagator (Line 15). Subsequently, program and optimization propagator are configured to ensure that every subsequent model is incomparable to solution i (Line 16).

In case a new solution is found, *unsat* is set to *False*, and the solution counter i is increased by one (lines 18–19). The new solution is saved under identifier i (Line 20). If i equals to 1, this amounts to finding a random solution to the ground program GP that is consistent with the theories in TO. If there has been a previous intermediate solution ($i > 1$), the comparison to it is removed from the optimization propagator and ground program (Line 21). Nonetheless, the novel solution becomes the optimal solution candidate (Line 22). After that, the main loop starts anew by trying to compute the next (better) solution.

Accordingly, the algorithm either computes the desired number of optimal solutions or proves that there exist less than n. Note that the algorithm is exact and complete. Hence, given enough time, all optimal models are computed and proven to be optimal. As this is often unrealistic in practice, the algorithm supports reporting intermediate results. Thus, if the search is interrupted at any time, the exploration returns the currently best known approximation set, i.e., the solution i and the set $\mathcal{S}$.

Note that the solving step (line 6) supports all solving modes of clingo including domain-specific heuristics [101] and multi-threading [137]. While domain specific heuristics help in

diminishing the overall runtime of the algorithm, multiple threads are used to set up the solver with different configurations in each thread that leads to varying approaches for covering the decision space.

4.1.2.3 Preference Specification

Preferences are a core concept of the optimization framework as they form the link between the foreground and background theories. A preference defines policies on whether a solution is better than, worse than, or equal to another solution. In general, it consists of a name p, a type pt and a set of attributes mapped to it. In its simplest form, a preference correlates with a specific objective, such as the area or energy requirements of the system. For example, the specification of the preference `area` is of type `sum`. Hence, it shall accumulate the cost of each allocated resource in an implementation to allow for a comparison with another implementation regarding their area costs. However, a preference type in general is more complex as it may aggregate multiple sub-preferences and dependencies as one attribute. In the present DSE framework, for example, the preference type `Pareto` aggregates three sub-preferences accounting for latency, area, and power consumption. The `latency` preference type, in turn depends on the QF–IDL theory to acquire the schedule of the individual tasks as detailed in Section 3.2.2. Each preference type is implemented as a dedicated propagator in the background theory that takes control over the evaluation of the found solution. This implies validity checks of hard constraints and the related generation of conflict clauses if the implementation candidate does not fulfill necessary requirements.

While the preference types are defined in the background theory, the specification (or instantiation) is done in the foreground theory. The preference specification declares the specific instances of preference type to be used, maps the required attributes in the form of ASP atoms to the corresponding preferences, and decides which preference will be optimized. Note that only one preference, the *lead* preference, can be optimized. Hence, the remaining ones must be declared as sub-preferences of the *lead* resulting in a hierarchical preference graph. According to the desired objectives, four preferences are defined: `latency`, `area`, `energy`, and the lead preference of type `ParetoBreadth` or `ParetoDepth`, depending on the search strategy. Besides the dependency of latency with respect to the QF–IDL theory, the latency calculation needs the mapping of each task as an attribute. Both `area` and `energy` are of type `sum`. The area preference needs the information of the allocation status of every processing and communication element to determine the cost of the implementation. In contrast, to calculate the energy of the system, the information of the allocation, mapping, and routing must be known. The allocation is necessary to infer the static energy consumption while mapping and routing are utilized to determine the dynamic energy and communication energy of an implementation candidate. The preference types `ParetoDepth` and `ParetoBreadth` aggregate the previous preference types and compare two solutions regarding their dominance relation. While the former handles the comparison of individual solutions (i.e., $x_1 \succ x_2$), the latter is extended towards sets of incomparable solutions. Hence, given two sets of non-dominated solutions S_1 and S_2, S_1 is preferred over S_2 if S_1 can be obtained by removing dominated solutions from $S_1 \cup S_2$ and S_1 contains at least one non-dominated solution not in S_2. For example, the set $S_1 = \{(0,3),(2,0)\}$ is preferred over $S_2 = \{(0,3),(3,1),(2,2)\}$ as the former contains one additional non-dominated solution, i.e., $(2,0)$, and does not contain dominated solutions from the union $S_1 \cup S_2$.

The resulting preference graph is depicted in Figure 4.4. Each time, a partial assignment

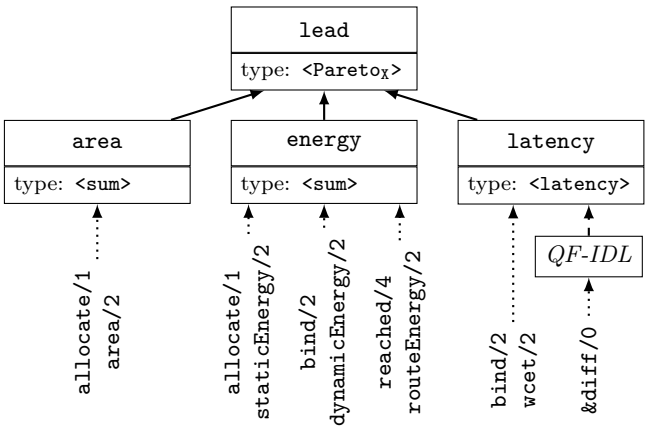


Figure 4.4: The resulting preference graph with the lead preference aggregating the three preferences `area`, `energy`, and `latency`. The attributes of each preference consist of ASP atoms that carry the necessary information to evaluate the objectives.

is decided by the ASP solver in the foreground theory, the lead preference is called. In order to make a decision, the lead preference, in turn, calls the attached sub-preferences that first evaluate the (partial) implementation according to the corresponding decisions represented by the atoms. If the preference depends on a specialized theory (e.g., `latency`), the theory first checks the validity before the preference can evaluate the implementation. The sub-preferences return the results to the lead preference which ultimately returns a final result. Depending on the lead preference, i.e., `ParetoDepth` or `ParetoBreadth`, the implementation is compared to the currently watched solution or the current set of non-dominated solutions, respectively.

4.1.3 Evaluation

In this section, the proposed DSE and optimization framework is evaluated on the basis of randomly generated test instances. The focus of the evaluation lies on the performance of the different optimization strategies $\mathcal{H}$, $\mathcal{T}_{Depth}$, and $\mathcal{T}_{Breadth}$. As the test instances are generally too large to be explored completely, all exploration runs will result in non-dominated sets that represent only an approximation of the true Pareto front. Hence, as a second experiment, the complexity of the underlying synthesis is reduced in two steps. To this end, the message routing is restricted to a shortest path routing (SPR) in a first and to a dimenstion order routing (DOR) in a second step. Both methods limit the number of routing alternatives that a message can take from its sender to the receiver. While SPR still allows all paths that have the shortest path, DOR only defines one route per sender-receiver pair. On the one hand, this drastically reduces the complexity of the problem and helps in exploring the search space faster when compared to an arbitrary length routing (ALR) as presented in Section 3.2 where each possible path is considered. On the other hand, by restricting the search space, DOR and SPR may introduce congestion on the links without the ability to find alternative routes. This can ultimately limit the achievable latency and may lead to worse Pareto front approximations. Hence, if the entire search was explored, ALR is expected to find solutions that SPR and DOR cannot.

4.1.3.1 Experimental Setup

To evaluate the optimization framework, 30 test instances are generated randomly with the instance generator presented in Section 3.4.1. They are composed of series-parallel applications and a heterogeneous platform template organized in a regular grid. Each test instance consists of one to four applications A that are comprised of $|S|$ series and $|P|$ parallel patterns, resulting into a number of $|A|+2\cdot|S|+3\cdot|P|$ tasks and $2\cdot|S|+4\cdot|P|$ messages per instance. Depending on the number of applications, the platform template size is accordingly adjusted. For all test instances with up to two applications, a grid size of $3\times3\times1$ is chosen while for instances with three and four applications, grid sizes of $3\times3\times2$ and $3\times3\times3$ are chosen, respectively. The size of the applications considered in this evaluation ranges from medium-sized applications with 34 tasks and 40 messages to large-sized applications with up to 166 tasks and 200 messages. To allow for more realistic specifications, a task is defined by varying number of instructions randomly selected from four different instruction types. Simultaneously, cycles per instruction (CPI) and energy per instruction (EPI) values are assigned to each processing element for each type of instruction. These values are used to calculate the WCET and energy requirements for each mapping option. All remaining attributes such as area and static power requirements are generated randomly within a specific range.

The design space exploration is conducted by the proposed framework with every possible combination of the presented search (i.e.,$\mathcal{H}$, $\mathcal{T}_{depth}$, and $\mathcal{T}_{breadth}$) and routing (i.e., ALR, SPR, and DOR) strategies. Hence, for each test instance, nine individual optimization runs have been conducted. To evaluate the performance of the variant strategies, the quality of the obtained approximated Pareto-set is calculated with respect to convergence and diversity. As the true Pareto front is unknown, all non-dominated solutions of each individual optimization run are accumulated and filtered through a final dominance check. Thus, the combined front contains all non-dominated solutions and represents the reference front. The convergence of is calculated by the binary ε-dominance [47] between the reference front and each individual approximated Pareto set. The ε-dominance represents the closeness of the approximation set towards the reference set. When comparing the resulting ε values of two approximation sets, the lower value signifies a better convergence. As no individual set can contain a solution that dominates a design point in the reference front, the minimum value 1 indicates that all solutions of the approximation lie on reference front. The diversity of the solutions is calculated by the entropy method [54]. As detailed in Section 2.3.1, the entropy calculates the flatness of the distribution of solutions in the non-dominated set. A grid size of 10 is chosen for the calculation of the entropy. Furthermore, the Gaussian influence function is defined as

$$\Omega(r) = \frac{1}{\sigma\cdot\sqrt{2\pi}}\exp(\frac{-r^2}{\sigma^2})$$

with $\sigma = 1/6$ and r representing the Euclidean distance between a grid point and a solution. With these parameters, the entropy results in a minimum value of around 2.46 if the approximation set only contains one solution[4-1] and increases with larger number of solutions and a more diverse distribution. Note that the calculation of the entropy is normally not defined for an empty approximation set, i.e., if no solution has been found. In this case, the entropy is set to a value of 0. All optimization runs were configured to use eight threads and have been

[4-1] The exact value depends on the specific location of the solution.

Table 4.1: Quality for some test instances achieved by the different configurations

$\|A\|$	$\|S\|$	$\|P\|$	Platform	I	*Hybrid*			$Theory_{\text{breadth}}$			$Theory_{\text{depth}}$		
					ALR	SPR	DOR	ALR	SPR	DOR	ALR	SPR	DOR
1	8	6	$3 \times 3 \times 1$	ε	1.487	1.280	1.139	1.487	**1.131**	1.275	1.877	1.505	1.894
				H	2.472	2.473	2.998	2.989	3.531	**3.768**	2.473	2.471	2.473
1	10	10	$3 \times 3 \times 1$	ε	inf	1.248	1.288	inf	1.153	**1.085**	inf	1.437	1.391
				H	0.000	2.472	2.473	0.000	**3.528**	3.514	0.000	2.473	2.471
2	11	9	$3 \times 3 \times 1$	ε	**1.099**	1.194	1.117	1.269	1.178	1.174	1.767	1.767	1.761
				H	2.473	2.473	2.473	3.091	**3.826**	3.741	2.473	2.472	2.472
3	22	27	$3 \times 3 \times 2$	ε	1.855	1.650	1.477	1.835	**1.279**	1.309	1.855	1.637	1.594
				H	2.473	2.463	2.473	2.501	**3.166**	2.572	2.473	2.473	2.473
4	21	20	$3 \times 3 \times 3$	ε	1.610	1.614	**1.000**	1.510	1.301	1.151	1.615	1.579	1.435
				H	2.473	2.471	2.473	2.743	3.043	**3.110**	2.473	2.472	2.473
4	24	38	$3 \times 3 \times 3$	ε	inf	1.892	1.533	inf	1.515	**1.103**	1.895	1.894	1.837
				H	0.000	2.472	2.472	0.000	2.915	**2.976**	2.473	2.473	2.473

executed on an Intel Core i7-4770 with 32 GiB RAM running Ubuntu 14.04. The timeout has been set to 30 minutes.

4.1.3.2 Results

For sake of brevity, the results of the optimization runs presented in Table 4.1 only constitute a representative part of all optimization runs. The complete set of results is depicted in the appendix in Table A.3. For each test instance, represented by the number of independent applications $|A|$, the number of series and parallel patterns $|S|, |P|$, and the platform template size, the convergence ε and entropy H are given for each individual exploration run. If the search does not return any valid implementation candidate, the ε-Dominance and Entropy are set to infinity and 0, respectively.

The results show the differences between the two major search strategies depth-first and breadth-first. In the two depth-first search strategies $\mathcal{H}$ and $\mathcal{T}_{depth}$, nearly every exploration only returns approximation sets that contain one implementation. An exception is the smallest test instance with one application, eight series, and six parallel patterns paired with DOR. Here, the set contains two implementations indicated by higher diversity of around 3 compared to around 2.4 for the rest. In turn, this indicates that a true Pareto-optimal implementation has been found as otherwise, no incomparable solution would have been explored (q.v., Figure 4.2(a)). In all other combinations, the Pareto-optimality of the design points can neither be guaranteed nor falsified as not the entire search space has been explored. Besides the proven Pareto-optimal design point for the smallest test instance, the implementation candidates found by the hybrid search strategy $\mathcal{H}$ outperform the other two approaches with respect to convergence in eleven further instances. Exemplarily, in the second to last instance in Table 4.1, the solution yielded by $\mathcal{H}$ in combination with DOR is located directly on the reference front. The corresponding ε-Dominance of 1 indicates that it dominates all remaining solutions found by the other approaches combined. The other depth-first search strategy, $\mathcal{T}_{depth}$, cannot find any Pareto-optimal solutions nor solutions that are close to the reference front.

As expected, the breadth-first search strategy, $\mathcal{T}_{breadth}$, finds more diverse approximation sets than other strategies. However, $\mathcal{T}_{breadth}$ also outperforms the hybrid approach with respect to convergence in most (16) of the test instances. This indicates that a less restrictive search strategy may prune the search space more effectively and, thus, favors the exploration of

Table 4.2: Quality by search strategy

	$\mathcal{H}$	$\mathcal{T}_{breadth}$	$\mathcal{T}_{depth}$
Diversity	0	27	0
Convergence	11	16	0

Table 4.3: Quality by communication model

	ALR	SPR	DOR
Diversity	0	15	12
Convergence	1	9	17

promising regions earlier. A summary of the exploration results with respect to search strategy is given in Table 4.2.

The second aspect of the experiment is summarized in Table 4.3. With respect to the applied routing strategy, the restricted, and thus, less complex approaches SPR and DOR clearly outperform the unrestricted approach ALR. Note that, for each instance, the optimization runs of all three routing approaches contribute to the reference front. While SPR has a small advantage over DOR with respect to the diversity of the found solutions, applying DOR yields solutions closer to the reference front. Although DOR provides the least routing alternatives for collision avoidance, it yields the best convergence results in more than 50% of the test instances. The possible reason is twofold. First, preventing the exploration of minor routing changes allows for larger steps through the decision space. This, in turn leads to a higher coverage of the search space and the discovery of better implementations early. Second, a minor change in the decision space often effects only a minor change in the objective space or does not change the objective vector at all. Hence, many costly but unnecessary evaluations are skipped compared to SPR and, especially, ALR. Only in one of the 30 instances, the unrestricted ALR is able to exploit the larger amount of routing options and yield the highest convergence. Due to the high complexity of ALR, it did not find any solution in eleven optimization runs and was outperformed by a factor of two in most other cases. If the optimizations were run completely, ALR is expected to yield equal or higher convergence for each test instance as it can find routing alternatives that the other two cannot. However, with the problem size considered in the experiment, the runtime of a complete optimization exceeded 30 days even for the smallest instance considering the least complex routing scheme DOR. Hence, a complete DSE for larger instances and more complex routing schemes is not viable due to the exponential character of the problem. Finally, note that three of the 30 instances did not yield any implementation candidates for any of the search and routing strategies within the time limit of 30 minutes.

4.1.4 Section Summary

In this section, an extension of the ASPmT–based synthesis framework has been proposed. The major advantage of the approach is the capability to tightly integrate a complete DSE with multi-objective optimization within the solving process of the ASP solver clingo. This allows for a succinct and holistic problem formulation within a single framework. Furthermore, the coupling of the individual solving steps in the fore- and background theories through common indicator variables provides the ability to share information for sophisticated reasoning.

While the foreground ASP solver explores the decision space systematically, the associated background propagators evaluate the assignments with respect to validity. As the background propagators already work on partial assignments, the detection of invalid regions prunes the search space more effectively when compared to conventional, full-assignment-based approaches with downstream theories. This advantage not only applies to the validity checks but also to the optimization propagator. Hence, if an incomplete solution is recognized to be dominated by an already found solution in the archive, the search can be pruned early. However, the evaluation of partial assignments is shown to be only valid if the considered objectives are assignment monotonic. Hence, the assignment of an additional decision must not improve the solution in any case. To this end, especially the latency evaluation has to be adapted as it may be subject to a phenomenon called scheduling anomaly. While introducing partial priorities has been shown to prevent this problem, the restriction has to be considered if further objectives shall be added to the evaluation and dominance checks. Hence, if one of the objective functions does not conform to assignment monotony, the optimization propagator must not make any decisions based on partial assignments.

The realization of the optimization framework has been evaluated with respect to three search strategies. Although the depth-first search strategies (i.e., $\mathcal{H}$ and $\mathcal{T}_{depth}$) theoretically prefer the optimization of the convergence first, they have been shown to be outperformed by a breadth-first strategy in the majority of test instances. The latter yields a higher diversity in all test cases and a better convergence in around two thirds of the test instances. Independent of the search strategy, the complexity of the problem can be identified as the major limitation of the proposed approach. The vast number of mapping and routing options, even for medium-sized problems prohibits the exploration of the complete search space in a reasonable time. A final evaluation shows that the quality of the acquired implementations benefits from a reduction of the complexity. This was shown on the example of the routing substep where a reduced number of routing options yields better results, although it may lead to congestions of the links. However, even with these restrictions, a complete exploration is still infeasible.

4.2 Archive Management

The optimization framework proposed in the previous section extends the ASPmT-based synthesis framework towards multi-objective optimization. Therefore, it benefits from the tight integration of various background theories and of the ability to process partial assignments. While the foreground ASP solver allows for an effective exploration of the design space, the subsequent steps of dominance checks and the evaluation of design points are subject to improvement. Induced by the increased number of dominance checks and evaluations, when compared to full assignment checking, both steps can be identified as the bottlenecks of the approach. To this end, the following two sections concern the improvement of the theory and optimization propagators. First, an archive management technique based on the Quad–Tree data structure and adapted for partial assignment checking is proposed in this section. In short, it offers an efficient implementation as it avoids unnecessary dominance checks compared to commonly used list-based archives. A technique based on safe approximations to speed up the evaluation process is subsequently proposed in the next section.

4.2.1 Quad–Tree data structure

When solving a MOOP, the true Pareto front is generally unknown. Therefore, newly found solutions are often inserted into a dominance free archive. This property is achieved by consequently deleting dominated solutions from the archive whenever a better solution is found during the search. To this end, the Quad–Tree data structure offers an efficient implementation as unnecessary comparisons can be avoided.

The basic structure (Figure 4.5) of a Quad–Tree for an m-dimensional multi-objective optimization problem consists of a single *root* node that holds up to $2^m - 2$ child nodes. Without loss of generality, only minimization problem are considered in the following. An m-bit index number, the *k-successor*, is associated with each child and is calculated as follows [105, 106]:

$$\forall i \in [0, m[: k_i = \begin{cases} 1, & \textit{if} \quad n[i] \geq r[i] \\ 0, & \textit{else.} \end{cases} \tag{4.6}$$

Here, n and r are m-dimensional vectors containing the fitness values for each objective and represent the child and the root, respectively and k_i represents the i-th bit of k. The *k-successor* expresses which objectives of a solution are better ($k_i = 0$) and which are worse or similar ($k_i = 1$) w.r.t a reference vector and determines on which position a new solution must be inserted. For instance, node B $\langle 20, 20, 21 \rangle$ in Figure 4.5(a) is a 110_b-*successor* of the root $\langle 15, 18, 35 \rangle$ as its first two objectives evaluate worse ($k_0 = k_1 = 1$) and the third objective better ($k_2 = 0$). Note that no children with a *k-successor* equal to 0 or $2^m - 1$ exist as they would dominate or be dominated by the reference point, respectively. Informally, an m-dimensional solution vector x represents the origin of 2^m adjoining hyperboxes numbered from 0 to $2^m - 1$ that are occupied by different solution vectors. All solutions located in hyperbox 0 ($2^m - 1$) are better (worse) than x for all objective values. Thus, solutions of these regions must not be saved as the archive would not be dominance free anymore. This results in a maximum of 2^m–2 children per solution. For example, assuming a two-dimensional solution vector x, hyperbox 0 located left below of x and hyperbox 3 located right above of x can not contain incomparable solutions (see also Figure 2.5 on page 22).

Using Quad–Trees, a new solution can be checked for non-dominance. Given above definitions, solution vectors in the Quad–Tree which may dominate a novel solution n are located in the subtrees of a root r whose indices contain zeros in the same locations as n. To this end, for each k, the *k-sets* $S_0(k)$ and $S_1(k)$ specify all positions of zeros and ones, respectively and are defined as follows:

$$S_0(k) = \{i \mid k_i = 0, i = 0, \ldots, m-1\} \tag{4.7}$$
$$S_1(k) = \{i \mid k_i = 1, i = 0, \ldots, m-1\} \tag{4.8}$$

Formally, only *l-successors* of r with $l < k$ and $S_0(k) \subset S_0(l)$ have to be traversed. Consider for example the novel solution vector A in Figure 4.5(a). First, the *k-successor* is determined according to Equation (4.6). The new solution may be dominated by children whose index l is smaller or equal to k and contains zeros at the same positions as k. Hence, with $k = 110_b$, the novel solution may be dominated by the children with indices $010_b, 100_b$, and 110_b.

Analogously, in order to check which solutions of the Quad–Tree are dominated by a new solution, only *l-successors* of r with $l > k$ and $S_1(k) \subset S_1(l)$ have to be traversed. For solution A in Figure 4.5(a) with $k = 110_b$, only the subtree with index $l = 110_b$ has to be checked. In

(a) Original Quad–Tree **(b)** Updated Quad–Tree

Figure 4.5: Example Quad–Tree with three objectives. The novel solution A (blue) dominates and thus replaces solution B (red).

the example, A is not dominated by any previously inserted solution, but it dominates vector $B = \langle 20, 20, 21 \rangle$.

Note that the calculation of the *k-successor* as described by [106] is insufficient to test if a new vector dominates a solution in the tree. That is, A would be wrongly declared as 001_b*-successor* of B and hence be inserted in the subtree of B at index 001_b although A clearly dominates B. Using this method, the archive would not be dominance free anymore. As a consequence, here, the use of an additional calculation, called k^**-successor*, is proposed to test whether a new vector dominates an already found solution. The difference between k and k^* will be explained and discussed in the following section.

After identifying a dominated solution in the archive, it has to be deleted (Figure 4.5(b)). However, deleting a solution from the Quad–Tree is not trivial in the general case. When deleting a node that itself contains child subtrees, those children have to be reinserted into the tree. The authors of [106] propose three strategies for updating the Quad–Tree and deleting dominated solutions. The first method reinserts all children of a deleted node immediately into the root of the Quad–Tree which may lead to a huge overhead if those children are also dominated. In the second method, solutions that have to be deleted are marked before only unmarked vectors are reinserted. Finally, in the third method, the reinsertion is done recursively while concurrently checking for dominated solutions. This way, the solutions do not have to be reinserted into the global root of the Quad–Tree but can be processed at lower levels. As the experiments in [106] show the general preference of third strategy, it serves as a starting point for the proposed approach described in the following.

4.2.1.1 Quad–Trees for partial assignment evaluation

Compared to multi-objective evolutionary algorithms (MOEAs), CDCL-based approaches can leverage partial assignment evaluation. That is, as detailed in the previous section, an incomplete solution can be discarded if it is already dominated by a solution in the archive. As each partial solution has to be checked whether it is dominated, the ratio of dominance checking to inserting (and hence deleting) grows with the number of decisions which are necessary to complete a solution. Consequently, the need for an efficient dominance check becomes apparent. However, as partial solutions are subject to deterioration throughout the solving, checking whether a novel solution dominates solutions from the archive *must* be delayed until the assignment is complete. As a consequence, the management strategy described in [106] cannot be used as it performs dominance checks for both directions (i.e., *dominates* and *is*

```
1:  function IsDominated(Solution candidate n, Archive root r)
2:      k ← KSucc(n, r)
3:      if k = 0 then return False                    ▷ n dominates r
4:      else if k = 2^m − 1 then return True          ▷ r dominates n
5:      else
6:          L ← ⟨l⟩ | l ∈ [1, k] ∧ l ⇒ k = 2^m − 1
7:          for all l ∈ L do
8:              if IsDominated(n, getChild(r, l)) then
9:                  return True                       ▷ Child dominates n
10:         return False                              ▷ n is not dominated
```

Figure 4.6: Recursive algorithm checking whether a partial assignment is dominated.

dominated) per step. Therefore, the algorithm is split into two separate steps, namely the dominance check (Figure 4.6) and the update (Figure 4.7) steps, which are described in the following.

4.2.1.2 Dominance Check

Figure 4.6 outlines the necessary steps to check if a partial assignment n is dominated by a vector in the archive. Corresponding to Equation (4.6), the *k-successor* of the current partial assignment, represented by its objective vector n is calculated with respect to the root node r (line 2). If k equals to 0, the partial assignment still dominates the root and the algorithm returns $False$ (line 3). This signals the solver that the solution is still feasible. Otherwise, if k equals to $2^m - 1$ (all bits are set to 1), the root of the tree already dominates the novel solution. Thus, the algorithm will return $True$ and the solver can exclude the partial solution and prune the search space accordingly. For every other value of k (lines 5 to 10), the children of r have to be tested. Therefore, first, the *l-successors* are calculated. The bitwise imply-operator ($\rightarrow$) in line 6 only returns $True$ if the condition $S_0(k) \subset S_0(l)$ is fulfilled. Finally, the corresponding children l of r are tested recursively if they dominate n (line 8). If any child dominates the novel solution, represented by n, the algorithm returns $True$ and a conflict clause can be generated. This process is repeated until all decisions have been made and the solution is complete.

4.2.1.3 Update

At this point, it is clear that the new solution n is not dominated by any vector of the archive. Hence, such tests can be safely pruned from the final update algorithm shown in Figure 4.7. Only the two cases where the novel solution n either dominates (lines 4–23) or is incomparable (lines 24 to 39) to the root r have to be considered. Besides n and r, two Boolean values *insert* and *parentAlive* complete the input parameters of the algorithm. While *insert* determines whether the new solution n is supposed to be inserted into the subtree of root r or not, *parentAlive* provides the information if one or more predecessor nodes of r were already dominated by n.

As indicated for the example in Figure 4.5(a), the original *k-successor* is unable to detect if a new solution dominates a vector from the archive in every case. Hence, for the update algorithm, the *k*-successor* (line 3) is used and defined as follows:

$$\forall i \in [0, m[: k_i^* = \begin{cases} 1, & \textit{if} \quad n[i] > r[i] \\ 0, & \textit{else.} \end{cases} \tag{4.9}$$

```
1: function UPDATE(Solution candidate n, Archive root r, insert, parentAlive)
2:     reinsert ← ⟨⟩
3:     k* ← K*SUCC(n, r)
4:     if k* = 0 then                                          ▷ n dominates r
5:         L ← ⟨l⟩ | l ∈ [1, 2^m − 2]
6:         for all l ∈ L do
7:             APPEND(reinsert, UPDATE(n, GETCHILD(r, l), False, False))
8:             REMOVECHILD(r, l)
9:         if parentAlive ∧ insert then
10:            r ← n
11:            for all t ∈ reinsert do
12:                INSERT(r, t)
13:            return ⟨⟩                                       ▷ nothing to reinsert
14:        else if parentAlive ∧ ¬insert then
15:            if |reinsert| ≥ 1 then
16:                r ← POP(reinsert)
17:                for all t ∈ reinsert do
18:                    INSERT(r, t)
19:            else
20:                DELETE(r)
21:            return ⟨⟩                                       ▷ nothing to reinsert
22:        else if ¬parentAlive ∧ ¬insert then
23:            return reinsert
24:    else                                                    ▷ n is incomparable to r
25:        if parentAlive then
26:            if HASCHILD(r, k*) then
27:                UPDATE(n, GETCHILD(r, k*), insert, True)
28:            else if insert then
29:                INSERT(r, n)
30:            L ← ⟨l⟩ | l ∈ ]k*, 2^m − 2] ∧ k* ⇒ l = 2^m − 1
31:            for all l ∈ L do
32:                UPDATE(n, GETCHILD(r, l), False, True)
33:        else                                                ▷ Parent is dominated
34:            L ← ⟨l⟩ | l ∈ [1, 2^m − 2]
35:            for all l ∈ L do                                ▷ Check all children
36:                APPEND(reinsert, UPDATE(n, GETCHILD(r, l), False, False))
37:                REMOVECHILD(r, l)
38:            APPEND(reinsert, r)
39:        return reinsert
```

Figure 4.7: Recursive algorithm to update the Quad–Tree after a complete assignment has been found.

Although the only difference between k and k^* are the "$\geq$" and "$>$" operators, respectively, the *k^*-successor* correctly determines whether a vector A dominates another vector B if one or more objectives of A are smaller than the corresponding objectives of B and at least one objective is indifferent (cf. A and B in Figure 4.5(a)). Formally, the *k^*-successor* is necessary if there exist two complementary proper subsets I, J of all indices in the interval from 0 to $m-1$ such that A is better in all $i \in I$ as well as A and B are indifferent in all $j \in J$ objectives:

$$\exists I, J \subsetneq [0, m[\mid J = [0, m[\setminus I :$$
$$\forall i \in I : A[i] < B[i] \wedge \forall j \in J : A[j] = B[j].$$

If k^* evaluates to 0, i.e., r is dominated by n, all children of r have to be checked recursively whether they are also dominated by n and otherwise have to be marked for reinsertion into

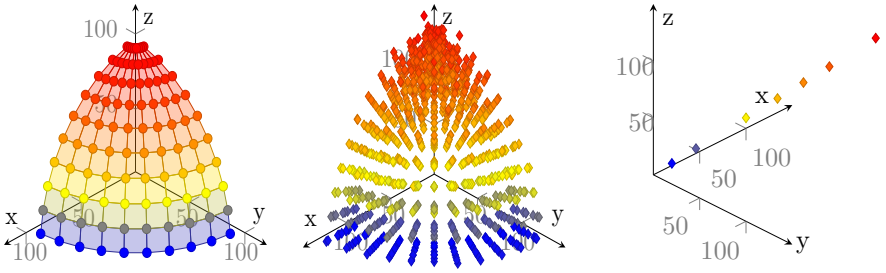
the Quad–Tree (lines 4–8). Independent of the truth value of *insert*, all children of r have to be traversed recursively with both *parentAlive* and *insert* set to $False$ (line 7). In case $insert = True$, the novel solution is inserted at the current layer of the Quad–Tree. Otherwise, the solution is inserted into another subtree with a root different to r. Afterwards, *insert* and *parentAlive* are analyzed. If both parameters equal to $True$ (lines 9–13), n replaces r and marked nodes are being reinserted into the subtree with the new root $r = n$. Otherwise, if $parentAlive = True$ and $insert = False$, the first vector to be reinserted becomes the new root (line 16) and the remaining nodes will be reinserted as children of it (lines 17–18). However, if there are no nodes to be reinserted, r is simply deleted (line 20). Finally, if both *parentAlive* and *insert* are $False$, the list of vectors to be reinserted is returned (lines 22–23) such that the nodes can be reinserted at a higher level. Note that the combination $parentAlive = False, insert = true$ cannot occur as the former indicates that the novel solution has already been inserted in a higher level of the Quad–Tree. In case n is incomparable to r, i.e., $k^* \neq 0$, n has to be inserted in the subtree k^* of r if $parentAlive = True$ (lines 25–32) or r and its subtrees have to marked for reinsertion if $parentAlive = False$ (lines 33–38). In the former case, UPDATE is called recursively with the parameters $parentAlive = True$ and *insert* derived from the current context if there is already a solution at position k^* (lines 26–27). Otherwise, n is simply added at position k^* if $insert = True$. Subsequently, all subtrees of r whose position indices are greater than k^* and contain ones at the same positions i as in k^* are checked if they are dominated by n (lines 30-32). In the latter case ($parentAlive = False$), all children of r have to be checked whether they are dominated by n and marked for reinsertion (including r itself). Finally, the nodes marked for reinsertion are returned to the parent of r.

4.2.1.4 Discussion on *k* and *k**

Note that the *k*-successor* in Figure 4.7 cannot detect whether A is dominated by B if all objectives are indifferent but one objective is worse in A. However, as it is already known that the new solution is not dominated by a vector in the archive, it is unnecessary to detect this anyway. In general, the *k*-successor* evaluates a solution better (regarding the number of zeros) than the *k-successor*. As the insertion of new solutions is based on k^*, this leads to a mismatch between check (Figure 4.6) and insertion (Figure 4.7) if one or more objectives are indifferent to each other. For example, assume the root node $r = \langle 5, 5, 5\rangle$ and a new solution $n = \langle 6, 5, 4\rangle$. With $k^* = 100_b$, both vectors are indifferent to each other and thus, n will be inserted as the child 100_b of r. Later, another solution $m = \langle 7, 5, 4\rangle$ is found that is dominated by n. Even though the check is based on the *k-successor* which evaluates m with respect to r as $k = 110_b$, it detects the dominance by also searching *l-successors* that include 100_b. That is, the difference between k and k^* does not influence the detection of dominated solutions but is necessary for their identification as shown in the example in Figure 4.5.

4.2.2 Experimental Evaluation

In this section, the proposed Quad–Tree implementation for partial assignment checking is evaluated and compared to a traditional list-based approach. To this end, two test series are executed. The aim of the first test is the evaluation of the scalability of the approach. Therefore, both approaches are fed with randomly generated Pareto sets of varying sizes. In the second test series the influence of the insertion is studied. Therefore, the randomly generated test cases are ordered by the quality of the respective solutions. Hence, this test serves as both a



(a) Pareto front **(b)** All solution vectors **(c)** Intermediate steps

Figure 4.8: Experimental setup for three objectives showing (a) the Pareto front, (b) all solutions and (c) the intermediate steps for one solution.

worst-case and a best-case scenario for the two approaches. When Pareto-optimal solutions are inserted first, it is expected that the Quad–Tree outperforms the list as the costly update can be skipped in the majority of the time. In contrast, if Pareto-optimal solutions are inserted in the end, solutions have to discarded regularly and the management overhead of the Quad–Tree may influence its performance beyond the list-based approach.

As the focus lies on the archive management, the actual DSE is abstracted and only the virtual partial solutions are given to the archive management. All tests were carried out on an Intel Core i7-4770 with 32 GiB RAM and implemented in *Python 2.7* running on Ubuntu 16.04.

4.2.2.1 Scalability

For testing the scalability of the proposed approach, a design space exploration is simulated for various complex problems by creating a set of different solutions that are evaluated by arbitrary objective functions. More precisely, first an m-dimensional spherical Pareto front is created that consists of a varying number of mutually non-dominated points (Figure 4.8(a)). Second, along the trajectory from the coordinate origin to each of the points, a specific number of dominated solutions x_i is randomly calculated (Figure 4.8(b)). Each solution, both Pareto-optimal and dominated, consists of a fixed length list (`hops`) of intermediate valuations (i.e $x_i = \langle x_{i1}, \ldots, x_{ihops} \rangle$) to simulate partial assignments (Figure 4.8(c)). Finally, each solution x_i is inserted into the archive. Hence, partial solutions are checked whether they are non-dominated with respect to all solutions in the archive. If x_{ij} is already dominated, x_i must not be inserted.

In the experiments, several instances with two to five objectives have been generated resulting in a varying number of non-dominated solutions that were inserted into a Quad–Tree and a list-based archive, respectively. Furthermore, the number of intermediate solutions has been varied from 50 to 200 to achieve a wide range of granularity for the partial assignment evaluation. For each test case, 20 independent runs have been executed. Figure 4.9 shows the results of the archiving runs. The x-axis represents the size of the Pareto fronts of the individual instances, while the y-axis shows the overall runtime of the run and the number of comparisons. While the former is depicted through the connected line chart, the latter is depicted by the bar chart. Note that both approaches have been fed with the same ordering of solutions and naturally result in the same Pareto front. One comparison corresponds to one calculation of k (or k^*, respectively) in the Quad–Tree and one dominance check for the list-based implementation.

With respect to the number of comparisons, the proposed archive management based on

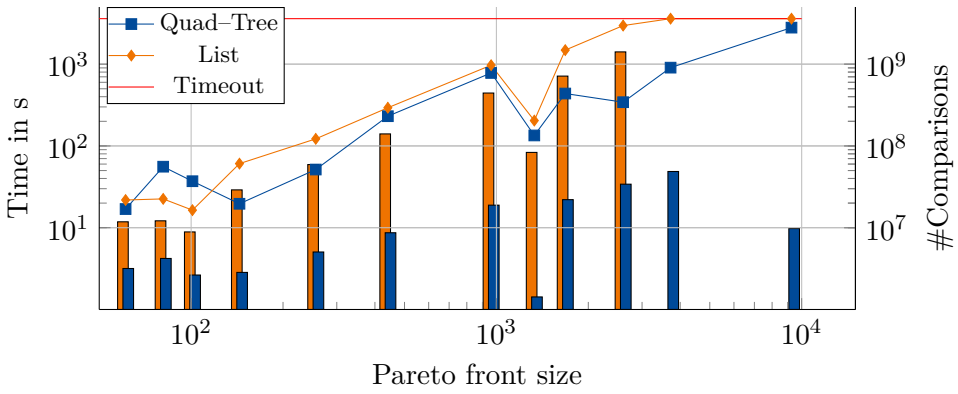


Figure 4.9: Runtimes (connected line chart) and number of comparisons (bars) for our experiments of various numbers of non-dominated solutions

the Quad–Tree outperforms the list-based approach significantly. Compared to the list-based implementation, the proposed approach requires approximately half the number of comparisons in the smaller instances and approximately one to two magnitudes fewer comparisons in the larger instances to filter the Pareto-optimal solutions. As the list-based archive is not able to filter the Pareto-optimal solutions within the time limit of 3,600 s, no number of comparisons can be obtained. The general trend, however, indicates that the gap between the list-based and Quad–Tree-based approach grows with the size of the Pareto front. The largest test instance as well as the instance with around 2,000 Pareto-optimal solutions do not follow the trend. The reason for this behavior is that the ordering of the instances was chosen randomly. If at the beginning many Pareto-optimal solutions are inserted into the front, much fewer comparisons have to be conducted as partial assignments can be disregarded.

In the second data series, the average time in seconds needed to filter the Pareto front from the complete set of solutions is represented by the line chart. For any test with more than two objectives (all but the first three), the Quad–Tree implementation outperforms the list. Note that the list even timed out for the last two test cases (timeout set to 3,600 s). The good performance of the list-based approach for the instances with two objective functions is expected. Here, the list can be sorted, such that a much more efficient search is possible compared to three or more objectives. Hence, only a fraction of the list has to be traversed when checking the dominance of a new (partial) solution. Furthermore, the Quad–Tree archives are not able to keep the same advantage as indicated by the number of comparisons. The main reason for this is, that the update and deletion procedures are much more complex than for linear lists where deleting a particular solution always takes constant time. Hence, while the list-based outperforms the Quad-Tree archive management for two-objective problems, especially test cases with numerous mutually non-dominated solutions benefit from the Quad–Tree archive management as the dominance checks outnumber the deletions from the archive.

4.2.2.2 Ordering

The outliers in the previous set of experiments already indicate a high impact of the ordering on the performance of the filtering process. Now, the worst and best case scenarios are studied. The best case ordering corresponds to a lexicographic ordering, i.e., solutions near the Pareto

front are inserted first. In contrast, the worst case is an inversely ordered insertion strategy. Hence, solutions that are already in the archive are less or more often dominated by new solutions, in the best-case and the worst-case scenarios, respectively.

An ordered insertion of the solutions results in a significant decrease of delete operations as newly found solutions are already dominated by the archive. Thus, the Quad–Tree data structure loses its disadvantage over the list. While the relation with respect to the number of comparisons is nearly identical to the random test cases, the runtime for Pareto filtering decreases for the Quad–Trees especially for large archive sizes. Considering, for example, an archive size of approximately 2,500 solutions, the Quad–Tree implementation requires only 400 s (compared to 800 s for the random insertion) while the list-based method runs 3,250 s (compared to 3,400 s). For the largest considered test case (8,412 solutions), the list times out while the Quad–Tree finishes in under 550 s (compared to 3,500 s). In the worst case scenario, the inversely ordered insertion strategy, the execution times of both approaches are generally higher and also closer together as solutions from the archive must be deleted more regularly. However, the general trend as seen in the random insertions remains and the Quad–Tree outperforms the list in most test instances. Only small two-dimensional optimization problems are filtered by the list (19 s) faster than by the Quad–Tree (31 s). Similar to random test cases, the Quad–Tree is constantly more than two times faster than the list for large test cases with three and more objectives.

4.2.3 Section Summary

In this section, an archive management technique has been proposed that is based on Quad–Trees proposed by Habenicht and Mostaghim et al. [105, 106]. However, the previous work on this subject did not consider partial assignment checking and could lead to incorrect results due to the calculation of the child node positions (*k*-successor*). The additional requirements of partial assignments include a fast intermediate dominance check. As only one direction has to be checked regularly, the majority of the original algorithm can be skipped. It only has to be executed once the assignment is complete. In comparison with a conservative list-based approach, the overhead for deleting dominated solutions from the archive is significant. Hence, for small instances and a few objective functions, the list-based approach performs better than a Quad–Tree implementation. However, with an increasing number of solutions to be filtered, the Quad–Tree outperforms the list, as fewer solutions have to be compared with others and the list loses its sorting character of the two-dimensional case. Quad–Trees are shown to work best when few solutions have to be deleted from the archives. Hence, the performance not only depends on the size of the Pareto front but instead is mostly influenced by the ordering in which solutions are added to the archive.

4.3 Evaluation through Safe Approximations

The employment of an improved archive management, as detailed in the previous section, can help to diminish the time to filter Pareto-optimal solutions. However, the previous steps of the DSE are not enhanced by this approach. Particularly, while designing complex embedded systems, the evaluating step is typically the most time-consuming one. Furthermore, it is frequently executed as it has to be conducted independent of the validity and optimality of a feasible solution. While at the beginning of an exploration, many implementation candidates

are not dominated by previously found ones, with a progressing exploration, the probability of finding better, i.e., non-dominated, solutions decreases. The time spent on evaluating such design points is practically wasted as they will be discarded by subsequent steps. Hence, diminishing the evaluation time without deteriorating the evaluation quality would increase the overall exploration performance and significantly improve the applicability of the DSE.

One possibility to accelerate the evaluation is the use of approximations. Thus, instead of costly calculating the precise objective value of a design point, an estimation is performed. Due to its lower complexity, the estimation executes in a fraction of the time that is necessary for the exact calculation. However, the result of an optimization which only utilizes approximate evaluations might differ from the exploration result obtained from exact evaluations. To overcome this drawback, this section proposes to combine approximations and exact evaluations whenever possible. That is, the quality of a design point is only calculated exactly if the approximation already promises good results. This way, the performance improvement of approximations can be coupled with an accurate optimization process. The applicability of the approach is conditional to certain requirements that have to be fulfilled by the objective functions. These will be discussed in the following sections.

4.3.1 Safe Approximations

This section details the underlying methodology that allows for the reduction of the number of expensive objective evaluations in order to accelerate the DSE. Compared to previous works, the central goal of this approach is to obtain the real Pareto front when the entire search space has been explored. The final result must include every Pareto optimal solution (*completeness*) and must not contain dominated designs (*correctness*). In order to guarantee these properties, the combination of safe approximations and the subsequent exact calculation of potentially good design points are proposed in this section. Keeping in mind that an exhaustive exploration of the entire search space is in general not viable for complex design problems, completeness and correctness properties are relaxed in the following in a way that they must hold with respect to the explored region of the search space. In other words, if coincidently a Pareto optimal design point is found during DSE, the approach assures that it is not replaced by a suboptimal design point even if its approximated quality evaluates better. Otherwise, if an optimal design point is not explored by the underlying search (i.e., by the ASP solver in the present case), the obtained non-dominated front is not equal to the true Pareto front. This can happen when only a specific time budget is available for exploration and the search space is too large to be covered in its entirety.

4.3.1.1 Approximating Objective Functions

When calculating the objective vector of a design point, it has to be evaluated with respect to each objective function individually. Such evaluations may be costly, when, for example, time-consuming simulations have to be executed. To accelerate the evaluation, a more cost-efficient analytical approach can be applied, which results in an approximated objective vector. However, an approximation often cannot guarantee any or only very vague error bounds for the desired objective functions. This leads to erroneous results when using these values to obtain the Pareto front.

Moreover, a precisely bounded error of the estimation, as required, for example, in the work of Abraham et al. [13], is often not practical for real world objectives such as latency where

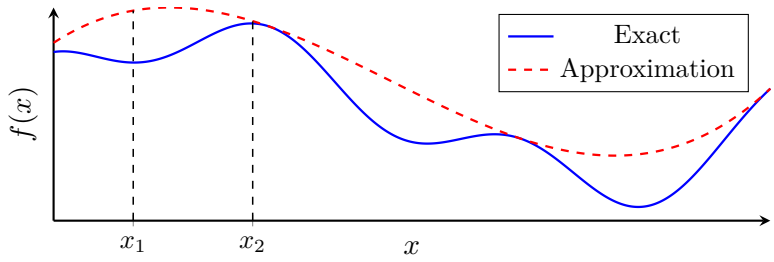


Figure 4.10: Safe approximation of an analytical function.

many factors (e.g., parallel execution, resource sharing) influence the calculation. Therefore, the proposed approach does not require strict error bounds of the approximated values. Instead, the approximation has to guarantee a consistent estimation, i.e., an approximation function either only exceeds the exact value or vice versa. To this end, a *safe approximation* is defined as follows that fulfills these requirements.

Definition 4.3.1 (Safe Approximation): Given an evaluation function $f : X_F \mapsto \mathbb{R}$, its approximation $f' : X_F \mapsto \mathbb{R}$ is a *safe approximation* if and only if

$$\forall x_i, x_j \in X_F : sgn(f(x_i) - f'(x_i)) \cdot sgn(f(x_j) - f'(x_j)) \geq 0,$$

where $sgn : \mathbb{R} \mapsto \{-1, 0, +1\}$ is the sign function.

Informally, for an approximation, applied to any feasible solution, to be considered safe, it must always be either larger than, smaller than, or equal to the corresponding exact evaluation of the same solution. Safe approximations can be further distinguished into *under-approximations* and *over-approximations*.

Definition 4.3.2 (Over- and Under-Approximation): A safe approximation is called *over-approximation* $f^{\uparrow}$ if and only if $\forall x \in X_F : f^{\uparrow}(x) \geq f(x)$. Analogously, a safe approximation is called *under-approximation* $f^{\downarrow}$ if and only if $\forall x \in X_F : f^{\downarrow}(x) \leq f(x)$.

In the domain of embedded systems design, a typical example for an over-approximation would be the calculation of the required execution time of a system by simply adding the WCET of each task without considering possible resource sharing and parallel execution. On the contrary, the calculation of the execution time would be an under-approximation if communication between dependent tasks is neglected.

A simple analytical example of safe approximations is visualized in Figure 4.10. The approximation, depicted as a dashed line, is always larger than or equal to the exact quality value. This satisfies the requirement of a safe approximation in general and, specifically, of an over-approximation. While a safe approximation can reduce the evaluation of an objective function, the further usage of the sole under- or over-approximated values can lead to misleading results. Note that, in the current example, the distances between the red and blue lines differ at varying locations. Hence, the utilization of a safe approximation does not guarantee particular error bounds. This can lead to a misinterpretation of the quality of a particular solution. Assuming a maximization problem, the approximations $f^{\uparrow}(x_1)$ and $f^{\uparrow}(x_2)$ in Figure 4.10 suggest that the solution x_1 is superior to solution x_2. However, when considering the exactly calculated objective values $f(x_1)$ and $f(x_2)$, solution x_2 evaluates better and should be favored over x_1.

Table 4.4: Pareto front of Approximations as of Figure 4.11

X_V	x_1	x_2	x_3	x_4
$f(x)$	$(2, 10)$	$(6, 6)$	$(7, 9)$	$(10, 4)$
$x_i \in X_P$	yes	yes	no	yes
$\dot{f}^{\downarrow}(x)$	$(1, 9)$	$(5, 5)$	$(3, 3)$	$(9, 1)$
$x_i \in X_P^{\downarrow}$	yes	no	yes	yes

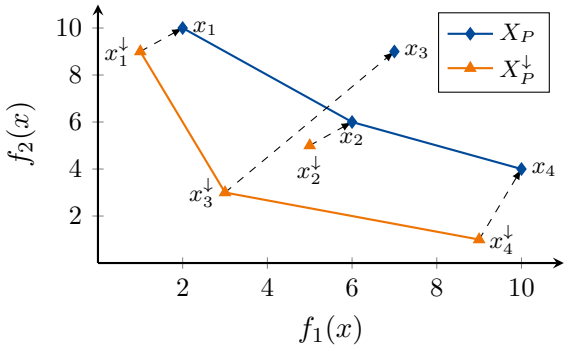


Figure 4.11: Exact (blue) and approximated (orange) Pareto front

The problem aggravates when determining the objective vectors of multi-objective optimization problems and subsequently filtering dominated solutions. When only safely approximated objective values are used to determine the objective vector, the resulting non-dominated front is not guaranteed to be correct nor complete. In the following, the notation of safe approximations (i.e., $f^{\uparrow}$ and $f^{\downarrow}$) is extended towards multi-dimensional functions. An n-dimensional safe approximation is denoted by $\dot{f}^{\updownarrow\cdots\updownarrow}(x) = (f_1^{\updownarrow}(x), \ldots, f_n^{\updownarrow}(x))$ with $\updownarrow \in \{\uparrow, \downarrow\}$. The order of the downwards and upwards arrows ($\updownarrow \cdots \updownarrow$) indicates which dimensions of the function are under-approximated and over-approximated, respectively. For example, the term $\dot{f}^{\uparrow\uparrow\downarrow}$ denotes a safe approximation of 3-dimensional function where the first two dimensions are over-approximated and the third dimension is under-approximated. If each dimension is approximated identically and the function dimensionality is clear from the context, the notation is shortened to a single upwards or downwards arrow, i.e., $\dot{f}^{\uparrow\cdots\uparrow} = \dot{f}^{\uparrow}$ and $\dot{f}^{\downarrow\cdots\downarrow} = \dot{f}^{\downarrow}$, respectively. For the sake of brevity, the term $\dot{f}'$ denotes an arbitrary safe approximation depending on the context.

Assume, for example, a two-dimensional minimization problem where a set of four feasible design points $X_F = \{x_1, x_2, x_3, x_4\}$ have been found. Subsequently, the individual solutions are evaluated and transformed into the objective space. When employing the exact (and costly) evaluation, the objective vectors of the four solutions equate to $f(x_1) = (2, 10), f(x_2) = (6, 6), f(x_3) = (7, 9)$, and $f(x_4) = (10, 4)$ as shown in the first row of Table 4.4 and visualized as the blue dots in Figure 4.11. When filtering the non-dominated solutions, x_1, x_2, and x_4 remain, while x_3 is dominated by x_2. In contrast, the evaluation using an under-approximation results in the objective vectors $\dot{f}^{\downarrow}(x_1) = (1, 9)$, $\dot{f}^{\downarrow}(x_2) = (5, 5)$, $\dot{f}^{\downarrow}(x_3) = (3, 3)$, and $\dot{f}^{\downarrow}(x_4) = (9, 1)$. This is shown by the third row in Table 4.4 and the orange dots in Figure 4.11. Hence, both objectives of each feasible solution are evaluated smaller than with the exact evaluation. However, filtering the non-dominated design points result in a disparate set of optimal solutions where x_3 replaces x_2 as $x_3 \succ x_2$. As a result, the latter non-dominated set is neither correct (x_3 is not optimal) nor complete (x_2 is missing).

Although a direct utilization of a safe approximation is therefore not viable, the next section proposes an improved approach where varying approximation accuracy does not impact the correctness of the result. While a specific error bound is not necessary for the correctness of the proposed approach, the impact of the accuracy on the performance is discussed later on.

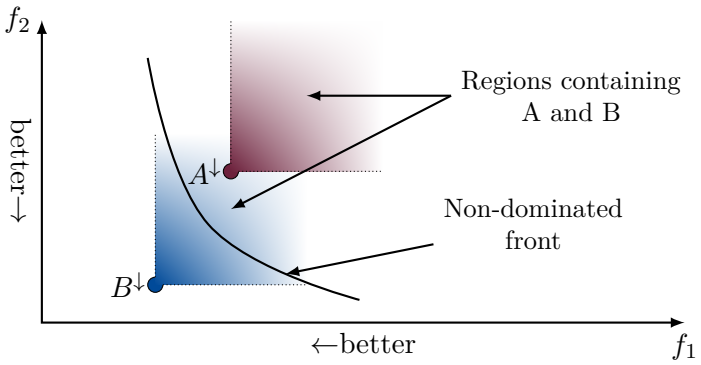


Figure 4.12: Under-approximations in minimization problems. The shaded red and blue regions are possible locations of the exact values A and B, respectively.

4.3.1.2 Pareto Optimality with Approximations

Despite the fact that the use of safe approximations alone does not result in the true Pareto front, they can be used to decrease the amount of necessary exact calculations. The idea is that the approximated quality vector of a newly found solution is compared against already found solutions that are currently present in the set of non-dominated designs (the set X_P). Only if the safely approximated quality vector is not dominated by any design point in the archive, the exact value is calculated. Otherwise, it is directly removed from the search and does not have to be investigated any further.

Theorem 4.3.1: Given a set of objective functions that are to be minimized (maximized) $f(x) = (f_1(x), \ldots, f_n(x))$, a design point x, its under-approximated (over-approximated) quality vector $\dot{f}^{\downarrow}(x)$ ($\dot{f}^{\uparrow}(x)$) and a set of exactly evaluated, mutually non-dominated design points X_P, the exactly evaluated quality vector $f(x)$ is dominated by X_P if $\dot{f}^{\downarrow}(x)$ ($\dot{f}^{\uparrow}(x)$) is dominated by a design in X_P.

Proof. Without loss of generality, only minimization problems are considered here. If the approximated quality vector $f^{\downarrow}(x)$ is dominated by X_P, at least one design point $y \in X_P$ evaluates better in each objective, i.e., $f_i(y) \leq f_i^{\downarrow}(x)|_{i=1,\ldots,n}$. According to Definition 4.3.2, the exact evaluation is always larger than the under-approximation $f_i^{\downarrow}(x) \leq f_i(x)$. Due to the transitivity of the inequality operator, it follows that $f_i(y) \leq f_i(x)$, i.e., $y \succ x$. □

A visualization of the idea is depicted in Figure 4.12 where the objective functions f_1 and f_2 are to be minimized. Two feasible design points A and B are found and are first evaluated approximately through $f_1^{\downarrow}$ and $f_2^{\downarrow}$. As the error bounds of the approximation is unknown, the real location of the objective vectors of A and B are unknown either. However, due to the properties of the under-approximation as of Definition 4.3.2 they are situated somewhere in the shaded red and blue areas, respectively. The objective vector $\dot{f}^{\downarrow}(A)$ is already dominated by the current archive of mutually non-dominated solutions (black line). Hence, it can be discarded directly. In contrast, the approximation $\dot{f}^{\downarrow}(B)$ dominates the front and might be Pareto-optimal. However, the real objective vector may be dominated by the current solutions in the archive. To get a definitive result, B has to be evaluated with the exact objective functions f_1 and f_2 and compared to the archive again.

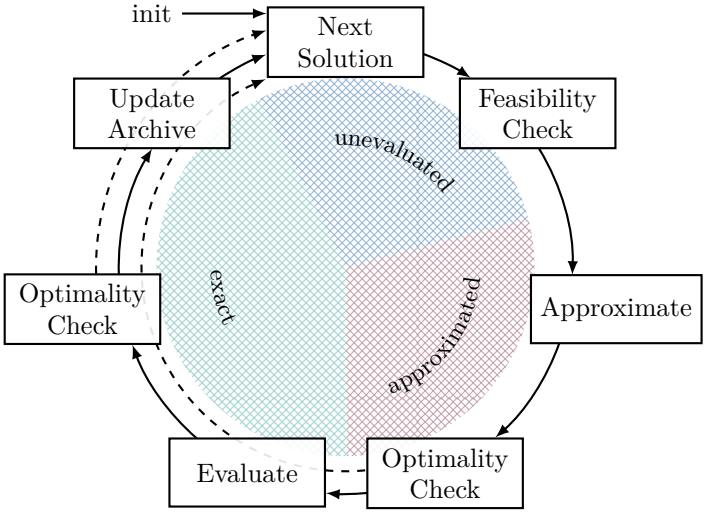


Figure 4.13: The design flow of the proposed iterative DSE methodology.

Note that the type of safe approximation that has to be used, depends on the optimization criterion of the corresponding objective. While minimization problems require under-approximations, maximization problems require over-approximations for the presented approach to work. For example, applying an over-approximation to a minimization problem, the approximated evaluation would not help in reducing the number of necessary exact evaluations. If $\dot{f}^{\uparrow}(x)$ was dominated by X_P, $f(x)$ is not guaranteed to be dominated by X_P as the exact evaluation is smaller (i.e., better) than the approximation. On the other hand, if the $\dot{f}^{\uparrow}(x)$ is not dominated by the front, the exact value must still be computed to save the solution and avoid the problems described in the previous subsection. Hence, the exact quality vector had to be calculated unconditionally for every found solution which led to an inevitable degradation of performance. This requirement is employed on a per-objective basis. Optimization problems with some objectives to be minimized and others to be maximized are not breaking the requirement when correct approximations are selected for respective objectives. That is, for a mixed 2-dimensional optimization problem where f_1 had to be minimized and f_2 had to be maximized, only a safe approximation of the form $\dot{f}^{\downarrow\uparrow}$ would be useful. Furthermore, note that the definition of safe approximations explicitly allows for approximation functions that are equal to the exact evaluations. If no safe approximation can be determined for an individual objective, the utilization of an exact evaluation only is still viable. In this case, the performance of the presented approach is equivalent to the conventional approach.

The workflow and the integration of the approach into the DSE is shown in Figure 4.13. During initialization, the archive (representing X_P), holding the set of mutually non-dominated design points, is empty. As can be seen in Figure 4.13, the approach is iterative, i.e., potential solutions are investigated one after the other. After the feasibility filter provides a feasible solution x, its quality is approximated by its corresponding safe approximation $\dot{f}'(x)$ and checked for validity[4-2] and optimality. As there are no solutions in the archive yet, the first found solution is not dominated by any other design point and therefore has to be evaluated exactly. The exact value $f(x)$ is again checked for validity and optimality and finally saved

[4-2]The validity check is not explicitly shown in Figure 4.13 but is assumed to be induced by the optimality check.

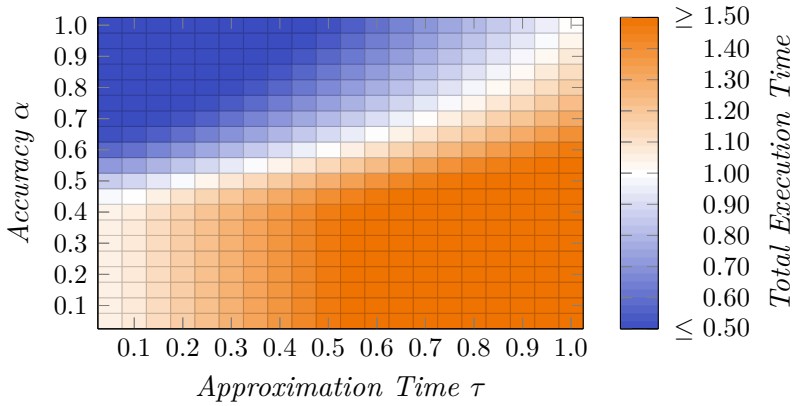


Figure 4.14: Impact of approximation accuracy and performance on overall runtime.

into the archive. In the next iteration, the next solution passing the feasibility filter undergoes the subsequent steps again. However, from now on, the newly found design points are not automatically non-dominated anymore and have to be checked for Pareto optimality twice if it is not dominated. Whenever a check fails, the design point x is discarded without the necessity to perform the remaining steps (dashed lines). This process is repeated until the whole design space has been explored or an abortion criterion (e.g., timeout) is fulfilled. At this point, the archive contains either the true Pareto front (if the whole design space was explored) or at least a complete and correct non-dominated front with respect to the explored region of the design space, i.e., no designs have been discarded or added wrongfully. Note that with this approach, the archive X_P, at any point in time, only contains exactly evaluated, mutually non-dominated design points.

4.3.1.3 Accuracy Impact

In comparison to an approach that only uses exact evaluations, the proposed approach needs one additional validity and Pareto check whenever a promising solution has been found. Thus, if many approximated solutions are assumed to be non-dominated, many solutions will be evaluated twice which will undoubtedly deteriorate the overall performance. Otherwise, if the majority of design points is already found to be worse than the current front, the time for the costly execution can be saved in many cases. As a consequence, the performance gain of the presented approach is primarily dependent on the quality, i.e., accuracy and performance, of the used approximation. Intuitively, the faster and more accurate (low absolute error with respect to the exact evaluation) the approximation for a given objective function is, the higher is the gain.

The impact of the approximation accuracy and timing properties on the performance of the entire exploration are studied in the following. Therefore, multiple optimization runs are simulated in which the performance and accuracy of the approximation functions are adjusted systematically. As shown in the graph depicted in Figure 4.14, the accuracy as well as performance vary from 5% to 100%. For example, an accuracy of 50% complies to an approximation function that consistently evaluates the objective 50% worse than exact function. Analogously, an approximation time of 50% indicates that the approximation takes only half the time of the exact evaluation. In this series, each exploration run consists of 100000 randomly generated

candidate solutions in the objective space with a convex-shaped Pareto-front. Afterwards each candidate, with its objective vector scaled by the accuracy factor, is checked for dominance. If it is considered to be non-dominated by the archive, another dominance is performed with its exact objective vector. Otherwise, the exact evaluation is skipped and the procedure continues with the next candidate as shown in Figure 4.13. The number of executed approximations n_A and exact evaluations n_E is saved throughout each run. After filtering is complete, the total execution time is calculated by adding the number of exact evaluations (each requiring one time unit) and the number of approximations weighted by the corresponding approximation time: $Time = n_E + accuracy \cdot n_A$. Finally, the overall performance, in relation to only using exact evaluations only is calculated by dividing the total execution time by the number of solution candidates, as an exact evaluation would be necessary for each of them.

As expected, the approach performs best when the accuracy is high and the execution time of the approximation is low (blue region). However, at an accuracy of 0.5 and below (and even higher for slower approximations), the overall execution time deteriorates and becomes worse compared to using a conservative approach (orange region). As indicated above, a low accuracy leads to fewer approximated solutions being dominated by the front. Hence, many candidates have to be evaluated twice. Additionally, even highly accurate approximation are not beneficial if the approximation time approaches that of the exact evaluation. Thus, it is imperative to find fast *and* accurate approximation functions to achieve an increase in performance. Note that in order to show a general trend, this test series made the overly simplified assumption that approximation accuracy and approximation performance are fixed. This assumption generally does not hold for real use cases and is not required for the proposed approach to work correctly. In DSE, the approximation accuracy and evaluation performance are highly dependent on the application structure and the decisions made during allocation, binding, and scheduling. For example, if the binding substep would map all tasks to the same resource (assuming no dependency conflicts), the latency evaluation would be a simple summation of execution times. On the other hand, if no resource is shared by different tasks, the evaluator had to account for additional communication delays. This would impact the performance and, at the same time, the approximation accuracy. Furthermore, the present series does not include the overhead, e.g., induced by the additional dominance checks, of the proposed approach. On that account, a more realistic use case is presented in the following sections.

4.3.2 Approximating Symbolic DSE

In the following, the approach, proposed in the previous section, is evaluated by applying it to the ASPmT-based system-level DSE, presented in Section 4.1. To this end, the latency evaluation is replaced by a SystemC-based simulation of each found implementation. While also started within the background theory of the framework, it offers the possibility to simulate the behavior in more detail and to acquire exact latency characteristics. The safe approximations are subsequently designed using the technology presented throughout Chapter 3, i.e., QF–IDL, but with further considerations to trade-off approximation accuracy and performance. The remainder of this section first presents the implementation of the simulation before the approximation functions are detailed.

Note that the utilization of the presented approach is, in general, applicable for any optimization (single- and multi-objective) problem where a safe approximation of the respective objectives can be designed. In particular, in the area of electronic systems design, it can be

applied at any abstraction level with few changes compared to the presented use case.

4.3.2.1 Evaluation Functions

While the area and energy consumption evaluations are easy to calculate in the considered model, the evaluation of the latency is more complex due to resource sharing, concurrency, and dependent task execution. To obtain an accurate result, the feasible solution is given to a NoC simulator implemented in SystemC using the transaction-level modeling (TLM) standard. Each router in the NoC has at most four independent external links as well as one dedicated home link connected to a processing unit. The processing unit implements the proposed execution scheme and is interfaced via TLM target and initiator sockets. Messages are transmitted through FIFO channels using a wormhole switching strategy with source routing. They are split into equally sized flits and prepended by a configuration dependent number of header flits, which contain information for the transmission through the NoC. The NoC model is designed to have a near to flit accurate granularity but keeping a high simulation performance. Therefore, the events for updating the actual state of all links are dynamically reduced. When a new message arrives at an input queue of a router, the routing decision is made and the current free space of the corresponding output queue is determined. Afterwards the simulation is continued until the time for transmitting all flits currently fitting into the current free space has passed. Then, either the following flits of the current message are put into the output queue, as with passed time the queue have new free slots, or the message is completely transmitted to the output queue and the transmission of a different message can start putting flits into the output queue. While a message is transmitted on an output link of a router, all other messages destined to the same output link are blocked until the last flit have been transmitted. Thus, if the amount traffic in the NoC is low, the simulation will only be activated by the events of a head flit entering an output queue and a tail flit leaving an input queue. If the NoC handles much of traffic, the amount of events can increase to the count of flits transmitted. As a credit per flit based reservation scheme is implemented, which reserves space in the home link of the destination router, the NoC is deadlock free (except for malicious source routes).

The prioritized self-timed execution paradigm of the tasks is realized by saving tasks according to their decided binding into priority queues of the corresponding processors and sorted through the partial order between them. Furthermore, each task is associated with its dependencies that are resolved whenever the appropriate message has been received, i.e., for each processor, only the first task can be executed if all of its dependencies have been received. After execution, the task is removed from the queue and sends its messages over the NoC towards the receiving task. If all dependencies have been resolved for the next task in the queue, it can be executed subsequently. Otherwise, the processor waits for new messages to arrive. Eventually, each task has been executed, and the queues are empty. The latest finishing equals to the latency of the system implementation and is returned to the caller.

Obtaining the latency by simulating the system requires numerous indicator variables to be submitted by the ASP solver. Besides the binding and routing decisions, also the partial order of tasks has to be communicated. Furthermore, SystemC does not allow restarting a simulation with different parameters (i.e., bindings, etc.) once a previous run has finished. That is why the simulation binary has to be executed anew for every implementation and variables must be exchanged via inter-process communication which is realized via a shared memory interface in this work. Consequently, the execution of the SystemC simulation is much more expensive

than the remaining two objectives, area and energy. In fact, evaluating an implementation with respect to its latency takes about three orders of magnitude more time than the evaluations of its area and energy requirements combined.

After obtaining the quality values for all three objective functions, the validity and Pareto filters are applied to the implementation. If no constraint violations are detected, and it is found to be non-dominated regarding already found solutions, the implementation is added to the archive. Simultaneously, each dominated, previously found implementation is removed from the archive to guarantee the mutual non-dominance. Finally, if the implementation is not valid or optimal, a conflict clause is generated that encodes the invalid solution which is then is returned to the ASP solver to prevent a reevaluation of this solution. The whole process is repeated until the whole design space is explored or an abortion criterion has been reached.

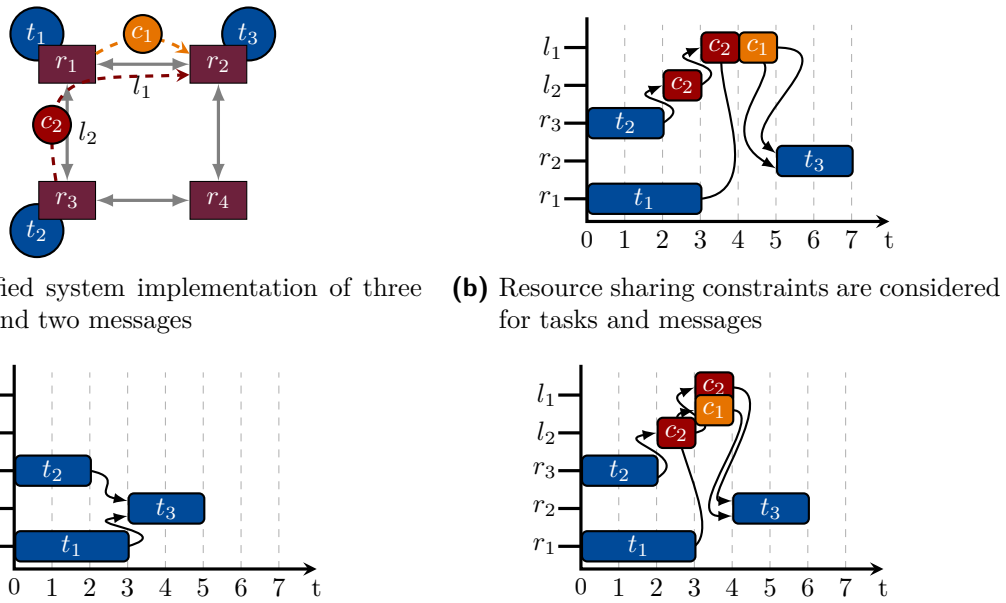
4.3.2.2 Approximation Functions

In order to employ the proposed approximation-based approach, safe approximations have to be found for each objective function. Here, an optimal implementation of a particular specification is one in which latency, area costs, and energy consumption are minimal. Hence, according to Theorem 4.3.1, under-approximations must be constructed for each of them.

The exact area and energy evaluations of an implementation are fairly inexpensive as they are calculated analytically. Safely approximating these objectives is not beneficial as no significant performance gain can be reached while attaining a high accuracy. Therefore, area and energy characteristics are exactly determined right away without using approximations at all. Due to the SystemC simulation, the corresponding latency evaluation of an implementation is much more time-consuming. The design of an accurate, yet high-performance approximation is imperative to achieve a significant improvement of the overall exploration time. To this end, the remainder of the section discusses three possible approximations for the latency evaluation.

As detailed in Section 3.2, the latency can be determined by applying QF–IDL in the background theory. Assume the example binding and routing of a simple application depicted in Figure 4.15(a). Here, the task t_3 depends on the results of its predecessor tasks t_1 and t_2. As all three tasks are bound to different computational resources, the results have to be transmitted via the messages c_1 and c_2. To reach t_3, that is bound to r_2, the route of message c_1 only consists of one hop, i.e., $\pi_{c_1} = \langle l_1 \rangle$, while the route of message c_2 consists of two hops, i.e., $\pi_{c_2} = \langle l_2, l_1 \rangle$. Note that for the sake of simplicity, communication resources (i.e., routers) are not shown in Figure 4.15(a). The Gantt chart in Figure 4.15(b) shows the schedule obtained with the approach discussed in Section 4.1. The accuracy is, with respect to the considered abstraction level, highly accurate as collisions are respected for both tasks and message hops. In particular, message hops $h_{c_1,1}$ and $h_{c_2,2}$ have to be sequenced as they occupy the same link at the same time. However, ensuring these constraints for large systems involves a huge number of additional decision variables that have to be explored during the search (viz. Section 4.1). Each possible overlap of independent tasks has to be considered and prevented through constraints. Especially, considering messages on flit level granularity, each possible hop may collide with other flits that are sent simultaneously over the same link. As a result, the increased evaluation performance is eventually nullified by the increased exploration complexity.

Therefore, three variations are investigated to increase the approximation performance. The first valid under-approximation for the latency evaluation of an implementation is the complete neglect of resource sharing constraints, i.e., for both tasks and messages. That is, tasks that are



(a) Simplified system implementation of three tasks and two messages

(b) Resource sharing constraints are considered for tasks and messages

(c) Resource sharing and dependency constraints are discarded for messages

(d) Resource sharing constraints are considered for tasks only

Figure 4.15: Different under-approximations for the calculation of the latency.

independent of each other are assumed to be executed concurrently even if they are mapped to the same device at the same time. Only dependency constraints are considered. This approach reflects the most aggressive approximation. On the one hand, as many theory variables can be neglected, the complexity of each individual approximation is accelerated. On the other hand, the accuracy deteriorates significantly for highly parallel applications. As shown in Figure 4.14, a high accuracy is imperative for the presented approach to be beneficial. As this problem is not represented by the running example, assume a duplication of the application and their corresponding binding and routing decisions. In this case, the exact schedule results in a latency of nine time units. For a schedule that does not respect resource sharing constraints, the latency equates to six time units. Hence, even with only two tasks overlapping[4-3], the error is already at 33 % (3/9).

A second variation is the omission of communication time but still respecting resource sharing constraints. This way, dependent tasks are assumed to be scheduled directly after each other, even if they are bound to different resources. While it resembles a valid under-approximation for the latency and saves many decision and indicator variables, the achieved accuracy is particularly low for highly parallel and communication intense applications. An example of this approximation is depicted in Figure 4.15(c). Even for this small application, the error of the approximated latency is already around 29 % (2/7). Similar to the first variant, utilizing this approximation is not expected to improve the overall exploration time as the approximation is assumed to be non-dominated with respect to the current front for nearly every found solution. Hence, the exact calculation is still executed each time. A third approximation function combines both previous variants to a certain extent. Resource sharing constraints are

[4-3]The task t_3 does not overlap with its copy as the resource sharing constraints shift the execution automatically.

assured for computational tasks only (i.e., independent tasks mapped to the same resource are executed sequentially), the links of the hardware architecture, however, are assumed to have an unlimited bandwidth and, thus, are not prone to congestion. This way, messages that are routed over the same link of the network do not interfere with each other. This approach has two advantages. First, as the number of possible message hops generally surpasses task mapping options, many additional decision variables can be saved. Second, although all possible message collisions have to be considered in the encoding, the number of actual collisions in the implementation are typically lower than within tasks. This is why the accuracy does not deteriorate as much as with the former two approaches. The schedule in Figure 4.15(d) shows this approximation approach. Even though the link l_1 is already occupied by message c_2, c_1 is transferred simultaneously. Note that the error has been decreased to around 14 % (1/7). In fact, if the ASP solver had selected another route for message c_2, i.e. via r_4, the latency approximation had the same value as the exact value.

4.3.3 Experiments

In this section, the proposed approach of safe approximations is evaluated by a set of experiments. Therefore, differently sized instances are generated by the instance generator presented in Section 3.4.1. The remainder of this section first elaborates on the experimental setup, followed by experiments regarding the approximation accuracy and the performance of the proposed approach.

4.3.3.1 Experimental Setup

Instead of comparing the proposed approach to the approaches discussed in Section 2.4.3, a reference DSE, as presented above, is implemented. The experiments are constructed in a way that they can assess the effect of the proposed contribution (i.e., the usage of safe approximations) only. This way, the reference DSE implies the very same scope and boundary conditions (i.e., models of computation, design time vs. runtime decisions, manual vs. automated DSE) as the presented approach. Hence, it can be assured that performance and quality differences stem from the contribution and not from a methodological mismatch. By strictly focusing on obtaining complete and correct Pareto sets by using approximations of the evaluation functions, a direct comparison to related approaches would be misleading as the premises of the other works are different. Even worse, as the boundary conditions would be varying, they could potentially mask the effect of the proposed solution.

The experiments are separated into two categories. First, a set of relatively small instances is created. Consisting of 11 tasks, 13 messages, and a total of 90 mapping options, the search space of these instances is small enough to be explored entirely. Thus, this series shows the absolute improvement of the proposed approach over a traditional one not using safe approximations. Furthermore, it underpins the claim of correctness and completeness of the methodology. To this end, it will be shown that Pareto-optimal solutions that are found by the proposed approach are identical to those found by the traditional DSE. That is, no optimal design points are missing and no non-optimal design points are present in the final result.

Second, a larger series of medium to large test instances is presented where a full coverage of the search space is not viable anymore. Therefore, a set of 120 specification instances are generated with varying properties and complexities as follows. The application, hardware template, and mapping options including their corresponding properties (i.e., area, WCET, etc.)

Table 4.5: Specification groups with its specific parameters

Group #	$\lvert A \rvert$	series	parallel	$\lvert T \rvert$	$\lvert C \rvert$
I	1	2	4	17	20
II	1	4	5	24	28
III	1	4	10	39	48
IV	2	4 (2;2)	9 (4;5)	37	44
V	2	7 (3;4)	10 (4;6)	46	54
VI	2	10 (5;5)	11 (4;7)	55	68
VII	3	7 (2;2;3)	12 (4;5;3)	53	62
VIII	3	10 (3;4;3)	13 (4;5;4)	62	72
IX	3	10 (5;4;3)	17 (4;8;5)	74	88
X	4	12 (3;4;2;3)	14 (4;3;3;4)	70	80
XI	4	15 (5;3;3;4)	20 (4;5;5;6)	94	110
XII	4	15 (6;3;4;2)	27 (6;6;6;9)	115	138

are created by the ASP-based specification generator (q.v. Section 3.4.1). For the application structure, series-parallel graphs are assumed as defined in Section 3.4.1. Hence, each application graph consists of a fixed number of series and parallel patterns that are connected with each other. In this experiment, the heterogeneous target architecture is formed by a regular 3×3 mesh implementing a NoC. For realizing the communication, on-chip routers are each connected to their neighbors and to one processor. Furthermore, each task is assumed to be composed of a specific mix of instruction types to model differently complex tasks. For example, a task may contain 60 integer, 30 floating point, and 10 special operations (e.g., AES) while a second task is only composed of 30 integer and 10 floating point operations. Each processor is characterized regarding its general capability, performance (cycles per instruction), and energy efficiency (energy per instruction) for each instruction type. With this information regarding tasks and processors, mapping options as well as their corresponding energy and timing properties are generated. Note that processors that are not capable of executing a specific instruction type cannot be chosen as a mapping option for a task that requires this type.

As shown in Table 4.5, the 120 problem instances are organized into groups of ten system specifications that share common properties in the form of the same number of series and parallel patterns. Hence, within a group, each application has the same amount of parallelism as well as number of tasks and messages. Note that the number of series and parallel patterns of specifications with more than one application is depicted in total and is also broken down by application in parentheses. For example, the graphs of group VII consist of three applications where the first one has two series and four parallel patterns, the second has two series and five parallel patterns, and the third has three series patterns and three parallel patterns. In order to further investigate the influence of the task execution time to routing delay ratio (ERR), three versions of each test instance are created totaling 360 individual medium to large instances plus three small instances. The medium ERR version is created as described above. In contrast, the high and low ERR versions have different ratios of the task execution times and routing delays of messages. In high ERR instances, the WCET of each task is increased by a factor of 10. Analogously, for low ERR, the execution time is decreased by a factor of 10. The goal of this approach is to investigate the accuracy impact of the approximation used. For higher

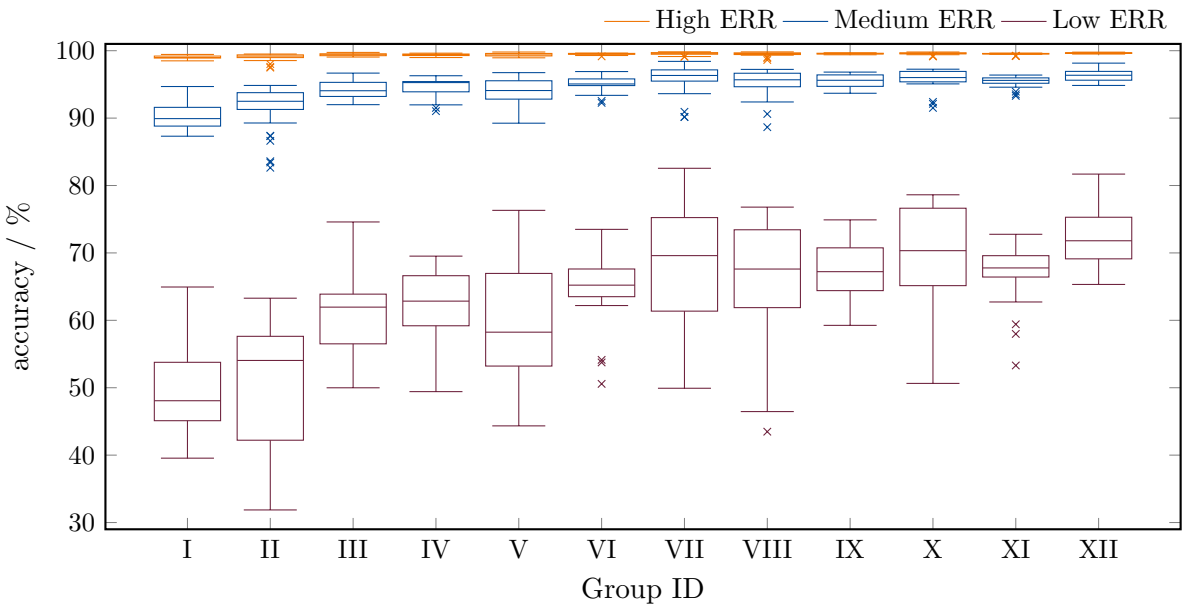


Figure 4.16: Latency approximation accuracy for different complexity groups and ERRs. Blue corresponds to the original, red to a low ERR, and orange to a high ERR.

ERRs, a higher accuracy is expected and vice versa.

For the different tests, as described in the following, each optimization instance is executed three times with a timeout set to one hour[4-4]. All optimization runs have been executed on an Intel Core i7 4470 with 32 GiB RAM and running Ubuntu 16.04.

4.3.3.2 Approximation Accuracy

In the first set of experiments, the accuracy of the proposed approximation approach is investigated with respect to the exact evaluation using the SystemC simulation. To this end, the ratio of approximated to exact values is determined for each explored implementation. Hence, the accuracy for one found solution equals the approximated value divided by the exact objective value. Subsequently, the mean accuracy and variance for each optimization run is calculated individually. The results for the medium to large instances are shown in the blue box plot series in Figure 4.16. The boxes represent the 0.25 and 0.75 quantiles of the input data with the indication of the 0.5 quantile (i.e., median). The lower whisker is the smallest data that is larger than the lower quartile minus one and a half times inter quantile range (IQR), i.e., the difference between upper and lower quartile. Analogously, the upper whisker represents the largest data that is smaller than the upper quartile plus one and a half times the IQR. In the default configuration, i.e., instances with a medium ERR, nearly all the optimization runs signify an accuracy higher than 90 %. However, it is apparent that the more complex test instances reach a higher accuracy. A higher number of tasks that are bound to the same number of processors leads to a higher average utilization of individual processors. Available computation time has to be shared and tasks have to postpone their execution even though the required messages have been received. Thus, the start time of tasks is less dependent on

[4-4]There is no timeout for the small instances.

the routing delay for messages such that an approximation of message transmissions does not influence the evaluation significantly.

The results of the low and high ERR optimization runs are shown as the red and orange box plot series in Figure 4.16, respectively. It has to be stated that the general trend is identical to the original optimization runs, i.e., less complex instances are less accurate and vice versa. However, the accuracy of the approximation approach differs significantly. While the red series (low ERR) achieves much lower accuracy, the orange series (high ERR) performs better. This behavior is expected and can be explained by the utilized latency approximation function. In high ERR problem instances, compared to the medium ERR instances, the task execution time has been increased. At the same time, the communication time has not been changed. Hence, the increase of the ERR. As a consequence, more free time slots are available on the links of the hardware platform to be used for communication messages. Thus, messages can be distributed according to more uniform patterns which leads to less congestion. Contrary, in low ERR instances, the time slots for communication are shorter and more collisions can happen on the links. As described in the previous section, congestions are not considered by the utilized approximation function, but instead are assumed to be irrelevant (cf., Fig. 4.15(d)). In turn, less congestion results in higher accuracy and vice versa.

A second observation is the wide range of achieved accuracies within one group of instances in the low ERR optimization runs. For instance, the accuracy for group two ranges from about 30 % to nearly 65 %. The reason for that behavior can be justified with the influence of the structure of a specific instance. Although the number of serial and parallel patterns is the same within each group, the execution times of tasks differ. This can lead to situations where multiple messages are sent simultaneously over the communication network, and especially, over identical links. If the availability of the time slots is simultaneously low (as in the low ERR instances), this influences the accuracy even more. Hence, the distribution follows a more uniform pattern for instances with a higher ERR.

Note that the accuracy results of the small test instances show the same trend as for the large ones. In detail, the accuracy of low, medium, and high ERR runs for the small instances are 0.49, 0.88, and 0.98, respectively.

4.3.3.3 Performance

While the approximation accuracy only indicates the performance of the utilized approximation, the performance assessment of the whole approach is presented in the following. Unlike assumed in Section 4.3.1.3, the accuracy of a utilized approximation is not constant. Different implementation candidates may induce variant approximation errors. Hence, the overall performance may present a different behavior as indicated by the median accuracy as presented above. For each instance, a reference DSE that only uses exact evaluation in the background theory and a DSE implementing the proposed approach are executed. Note that easy to calculate validity checks are carried out in both the reference and the approximation-based DSEs. In particular, this includes a check for schedulability that is based on the accumulated WCETs (q.v. Equation (3.2)). If, for any resource, the sum of execution times exceeds the specified periodicity, and thus, the maximum allowed latency of the application, the evaluation is not executed. Hence, the reference and the proposed approach do not differ in this aspect.

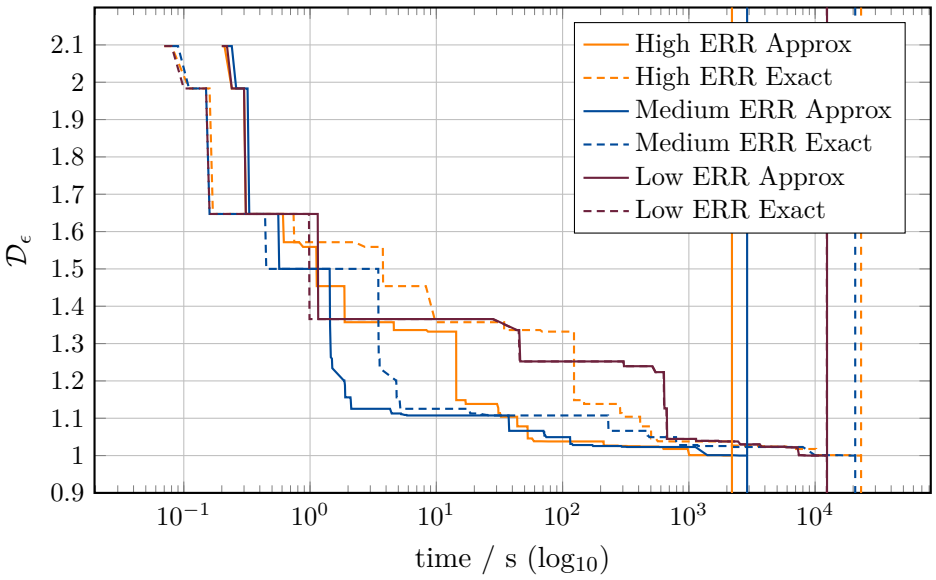


Figure 4.17: Epsilon dominance $\mathcal{D}_\epsilon$ over runtime for the small test instances. The vertical bars signify the time when the complete search space of the corresponding run has been explored. Note that the x-axis is logarithmic.

Small Instances The small instances are explored completely. Thus, absolute performance numbers can be presented for the proposed approach compared to the traditional methodology using exact evaluations only. For evaluating the performance of the approach, the ϵ-dominance [57] is calculated (q.v. Section 2.3.1). In short, the epsilon dominance measures the convergence of one front to a reference front. A value of $\mathcal{D}_\epsilon(A, B) > 1$ signifies that A is dominated by the reference front B and a value of 1 indicates that A lies directly on the reference front. As the true Pareto front is known for the small instances, a value smaller than 1 is not possible.

Figure 4.17 depicts the experimental results and shows the quality of the obtained non-dominated fronts over time. The solid lines represent the approximation-based runs while the dashed lines represent the quality of the traditional approach. The vertical lines at the end show the overall runtime of the corresponding optimization run. Essentially, there are three important observations. First, for medium and high ERR runs, the proposed approximation-based approach converges faster than the corresponding traditional approach towards the Pareto front. The optimum is reached approximately one order of magnitude earlier (i.e., 2,893 s vs. 20,771 s and 2,194 s vs. 23,117 s) with the high ERR run having the largest gap between approximation-based and exact run. In the high ERR run, only 7,183 (out of 2,103,799; 0.35 %) solution candidates had to be evaluated exactly while in the medium run, due to the lower accuracy, 98,443 (out of 1,936,887; 5.08 %) solutions had to be evaluated exactly. Second, in the low ERR run, the novel approach performs basically identical to the reference approach. This becomes more obvious when the high number of 929,674 (out of 931,351; 99.82 %) necessary exact evaluations are considered. Hence, both the approximated and the exact evaluations had to be executed for the low ERR instance. Thus, although a low accuracy deteriorates the performance significantly, it performs not worse than the traditional approach. Third, in the beginning of the optimization runs, up until about 10 s, the exact approach outperforms the approximation-based appraoch. The reason for this behavior is to be found in the solutions in the archive. In

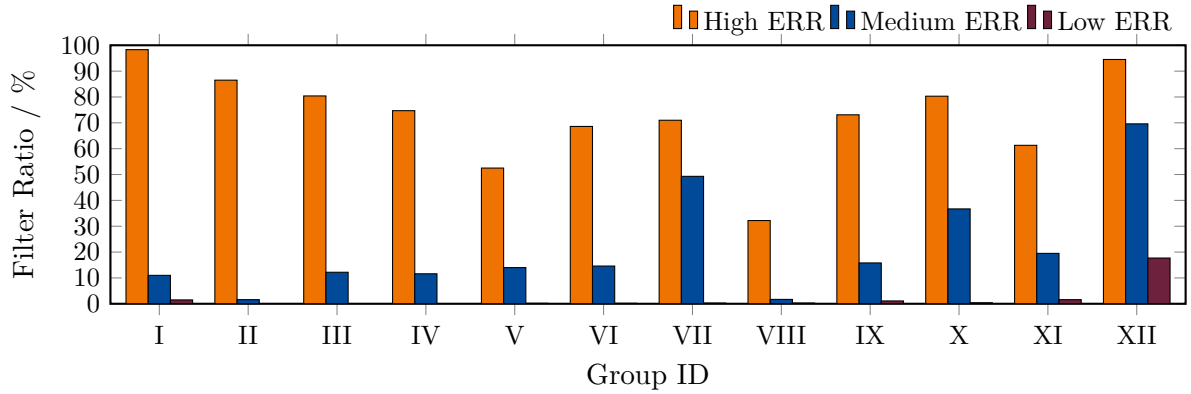


Figure 4.18: The average Filter Ratio of the instances in the individual specification groups.

the beginning, the search starts with an empty archive. Implementation candidates found early are thus non-dominated with respect to the current archive and the exact objective values have to be determined unconditionally. In the proposed approach, the approximated and exact evaluations are executed while the traditional approach only has to perform the latter step. That is, the traditional approach gains an advantage in the early search. After a few solutions have been found and the archive contains more optimal solutions, the number of necessary exact evaluations decreases and the approximation-based approach starts to outperform the traditional one.

Note that due to the altered instances, the number of solutions and necessary decisions are not comparable between low, medium, and high ERR runs. The foreground solver might prune the search space earlier before the background solver is even started in some cases.

Medium and Large Instances To evaluate the performance of the proposed approach for the medium and large instances, two measurement series were performed. The first series regards the filter ratio (FR) and its results are shown Figure 4.18. A detailed presentation of the results can be found in Table A.4 in the appendix. The FR is defined as the relative number of solutions that can be already removed safely from the search after the approximated evaluation has been executed in the background theory. That is, it corresponds to those implementations that would have been unnecessarily evaluated. Figure 4.18 depicts the average filter ratio of each specification group and each ERR type. Therefore, in every group, each instance has been analyzed regarding its filter ratio at the end of the one-hour exploration run. The value depicted in Figure 4.18 is then calculated as the geometric mean of all instance in the corresponding specification group. When compared to achieved accuracy (q.v. Figure 4.16), the FR is shown to roughly correlate with the accuracy of the corresponding group. Hence, a high accuracy results in a high FR. This is shown for the medium, and especially, high ERR instances. For nearly every considered specification group, the average filter ratio of the latter is 75 % percent or more, with two outliers in groups V and VIII. That is, the majority of expensive exact evaluations can be skipped with the proposed approach. However, the influence of the accuracy is not constant. Especially, in most cases of the low ERR specifications, the FR is extremely low although an average accuracy of more than 50 % is achieved. For example, the average FR of 0.021 for the low ERR specification group V indicates that only 2 % of all solutions can be removed from the search before the exact quality is determined. At the same

time, the approximation was able to achieve a median accuracy of 60 % for specification group V. In contrast, specification group II can be approximated less accurately, but the filter ratio is with 7.6 % more than three times higher. It can be concluded that the accuracy is only one factor for the applicability of the approach.

A second criterion is the *hardness* of the generated specification. Even if two specifications belong to the same specification group and achieve a similar accuracy, the FR may differ. As explained in Section 3.4.1, the hard constraints of individual specifications (especially the periodicity and execution deadline) are generated by first finding a set of unconstrained solutions which are subsequently analyzed for their achieved latency. The generated specification is then assigned with the achieved latency multiplied by a *hardness* factor smaller than one. Hence, if for a specific instance, a near-optimal solution was already found in the initial step, the resulting timing constraints are hard to fulfill. In these cases, the solver takes a long time to find any valid solutions at all. During this period, all solutions have to be evaluated exactly as the non-dominated archive is still empty. A good example is specification group III (q.v. Table A.4). In the high and medium ERR instances, the maximum FR by any of the instances is nearly one, i.e., practically all solutions could have been excluded by applying the approximation only. However, both contain instances (minimum FR approaches zero) where only a small fraction of solutions can be pruned with the help of safe approximations. The rest had to be evaluated exactly. Specification group XII shows the most significant correlation between accuracy and FR. Here, all three ERR types perform unusually well. The analysis of the individual exploration runs show however, that the relatively few solutions have been evaluated in the background theory. Thus, most assignments are already pruned by the ASP solver through the necessary but easily calculable schedulability check (q.v. Equation (3.2)). Most of the remaining solutions can be subsequently filtered by the safe approximation.

The second series of experiments regards the performance improvement of the proposed approach with respect to a DSE using exact evaluations only, called *reference* in the following. Note that ASP solving is deterministic when supplying the solver with the same random seed. Thus, executing the same instances leads to comparable results between the reference and the approximation based approach. As typical, complex DSE problem instances impose a huge number of decision variables. Therefore, it is impossible to cover the whole design space in reasonable time. Hence, the performance gain is measured by means of the found feasible solutions. The idea is, that if the evaluation process is faster, more time can be spent searching for new solutions in the ASP solver. Figure 4.19 depicts the average improvement of the twelve specification classes. A complete presentation of the results is shown in Table A.5 in the appendix. Two observations can be made from the results. First, none of the low ERR instances is able to significantly outperform the reference implementation. However, the performance is neither significantly worse. In the worst case, the proposed approach only deteriorates the DSE by 1 %. The root of this behavior lies again in the low accuracy and successively the low FR of the used approximation function for these instances. Therefore, for nearly all the found implementations, both the approximation and the simulation has to be performed. Even for specification group XII, where the filter ratio was substantially higher than for the rest, the proposed approach does not improve the overall performance. This indicates again that the majority of solutions is pruned by the ASP solver and does not reach the background theory. As a result, the reference and proposed DSEs behave similar.

The second observation is that the improvement decreases with increasing complexity of the problem instance. While, for the smallest specification group I, the improvements of the high

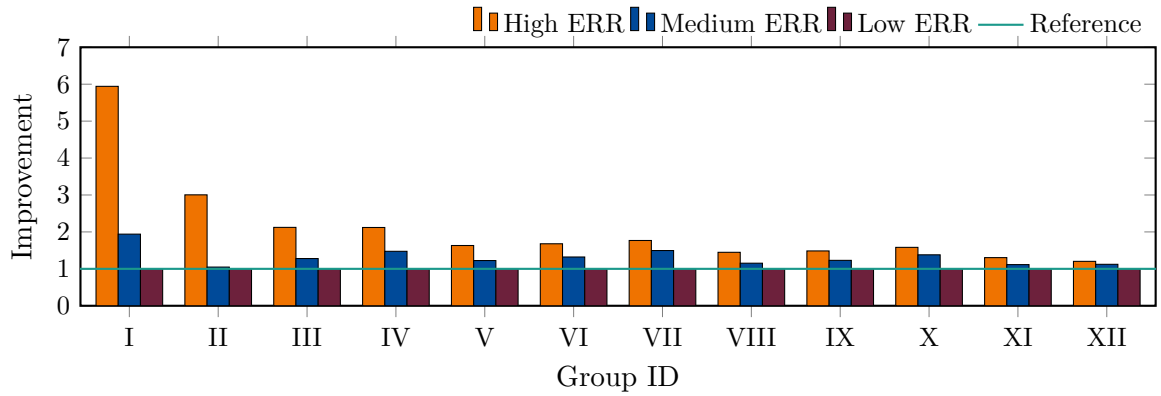


Figure 4.19: The average performance improvement of the instances in the individual specification groups.

and medium ERR instances are nearly six-fold and three-fold, respectively, the improvement deteriorates with an increasing number of independent applications, tasks, and messages. This can be explained by the exponential increase of decision variables that have to be explored by the underlying ASP solver. Thus, it simply takes continuously more time to find a feasible solution in the design space. However, the proposed approach only improves the evaluation step of the DSE while the search itself is not altered. That is, according to Amdahl's law [138], the overall improvement decreases as more time is spent in non-improvable steps.

A second factor to the poor performance of the more complex instances is again to be found in the hardness of the instances. Some instances even show a complete (q.v. Table A.4 in the appendix) failure of the approach where nearly all solution candidates are evaluated exactly even though the accuracy is virtually 100 %. An analysis of the respective instances shows that they resemble particularly hard instances where most solution candidates do not conform to specified timing constraints. Thus, the archive of non-dominated solutions does not contain any solutions for a very long time. In turn, each solution candidate has to be evaluated exactly. This deteriorates the performance accordingly and can be observed more frequently in the large instances. It is expected that performance increases with an exhaustive search space coverage. However, as the problem is exponentially complex in general, an exhaustive search is generally not possible in a reasonable time. To summarize, the utilization of safe approximations can be beneficial if their accuracy and performance are high. While the improvement deteriorates with larger instances, it still outperforms the reference approach that uses exact evaluations only.

4.3.4 Section Summary

This section regarded the evaluation of solution candidates found during the search. Therefore, a novel approximation-based approach has been proposed that allows the acceleration of the evaluation without compromising the correctness and completeness of the DSE. The core idea is the utilization of safe approximations. With respect to an exact evaluation, a safe approximation guarantees the value to be consistently lower (viz. under-approximation) or higher (viz. over-approximation). Only if this approximation indicates a good solution, the exact evaluation is executed. Otherwise, the solution candidate can be directly rejected and the exact evaluation can be skipped entirely. Depending on the characteristics of the individual objec-

tive, either an under-approximation or over-approximation has to be defined for the proposed approach to work. However, if no such safe approximation can be obtained for one or more objectives, the proposed approach also allows the use of the exact evaluation only. In this case, the DSE is identical to a conventional approach. Both the theoretical and experimental analysis show that accuracy and performance of the approximations have to be high for the proposed approach to improve the DSE effectively. On the one hand, if the accuracy is too low, the approximation cannot distinguish good and bad solution and exact evaluations have to executed unconditionally. This leads to larger number of calculations than when the exact evaluation was executed right away. On the other hand, an approximation that does not perform significantly better than the exact evaluation is similarly ineffective as the time savings are not large enough to compensate for the two calculations. However, the DSE used as example shows that the proposed approach can achieve significant performance gains when the approximation is defined carefully. Here, the latency objective has been shown to be a good candidate for defining an under-approximation while the energy and area objectives where calculated exactly only. While the improvement of the proposed approach is shown to be significant for the small instances (up to one order of magnitude), the improvement in larger instances deteriorates, although the accuracy and filter ratio remain high. One possible reason is the exponentially increasing design space. That is, in larger instances, the search in the foreground solver (viz. ASP) for solution candidates overtakes the evaluation thereof. The former, however, is not influenced by the proposed approach. As a result, the overall performance gain is less significant than for smaller instances.

Finally, the application of the proposed approach is not limited to system-level DSE. In principle, it can be adopted for every optimization problem, where at least one objective can be safely approximated according to its optimization criterion. With little effort, it can be applied to every MOOP (and single-objective optimization problem (SOOP)) approach that compares newly found solutions with some kind of archive of currently known non-dominated solutions. This includes many population-based meta-heuristics such as the non-dominated sorting genetic algorithm (NSGA) and ant colony optimization (ACO).

4.4 Chapter Summary

In this chapter a novel approach to the DSE problem of embedded systems design has been proposed. It represents an extension to the ASPmT-based synthesis framework introduced in Chapter 3. Unlike most of the previous work in this area, the DSE is exclusively built on formal methods that allow for a systematic exploration of the design space. That is, besides the search for novel solutions, as presented in Chapter 3, the evaluation and optimization steps are tightly integrated into one framework. This allows a succinct and holistic formulation of the DSE. Therefore, the utilization of ASPmT offers the benefit of effective reasoning of the Boolean search as well as the capability of adding various domain specific background propagators and theories within one closed environment.

As both foreground and background theories are tightly coupled through common variables, the proposed approach is able to gather information on partial assignments instead of full assignments only. This already has been shown to be beneficial for the synthesis framework. However, also the optimization propagator can use information of partial assignments to prune non-optimal regions of the search earlier during solving. On the downside, the utilization of partial assignments imposes more work onto the downstream process, i.e., the evaluation and

the dominance checks of potential solutions. The latter has been addressed in Section 4.2. Here, the thesis proposed a methodology to increase the performance of dominance checks. Based on the Quad–Tree data structure, the section showed how the proposed approach can save unnecessary dominance checks for partial assignments while maintaining a correct archive in the end.

Finally, Section 4.3 proposes a novel approach of using approximations in the evaluation phase without breaking the correctness and the completeness of the DSE. To this end, the notion of safe approximations is introduced. These are special functions that approximate a property consistently lower (i.e., under-approximations) or higher (i.e. over-approximation). If such function can be defined for an objective of the DSE, it can be used to safe evaluation time. Only if the approximation indicates a good result, the exact evaluation has to be executed. If this is not possible for each objective of the DSE, the exact evaluation can be used instead in these cases. The performance of the proposed approach heavily depends on multiple factors. Although a high accuracy is not imperative for the approach to deliver correct results, it is essential for a good performance. Besides the accuracy, also the approximation speed-up has been shown to influence the overall gain. Finally, also the complexity of the problem instance itself influences the performance of the proposed approximation-based approach.

5
Conclusion

The thesis at hand proposed a systematic approach to the DSE of embedded computer systems. To this end, a novel ASPmT-based synthesis framework was first developed. It allows the transformation of deadline-constraint, periodic, and cyclic behavioral specifications into structural implementations that conform to various feasibility and validity constraints. The utilization of ASPmT provides two essential benefits. First, the problem instance and all synthesis constraints are formulated by a uniform input language resulting into one succinct and elaboration tolerant encoding. Furthermore, compared to other Boolean solving techniques (e.g., SAT), the encoding is split into a problem instance and a general problem encoding that can be reused for every individual instance. Second, it allows the tight integration of the foreground Boolean ASP and specialized background QF–IDL solvers. While the feasibility constraints can be effectively encoded with ASP rules, validity constraints are relayed to the background solver. This technique is imperative as non-linear problems, in particular the latency of the system, are not effectively computable with Boolean solvers. Foreground and background solvers communicate through common variables, allowing for the propagation of conflict clauses from both sources. In fact, the application of QF–IDL for performing the schedulability analysis yields a minimal reason if the set of assigned decision results in invalid implementations. Furthermore, the tight integration allows for the evaluation of partial assignment if the objectives are assignment monotonic. Hence, the background theory can already check the incomplete implementations for constraint violations. Invalid regions can thus be detected and pruned from the search space early during the search, which has been evaluated in Section 3.4.2.

Following, in Chapter 4, the synthesis framework has been extended towards a holistic DSE with multi-objective optimization. To this end, the background theory has been completed with a Pareto checker and additional specialized propagators evaluating the power consumption and area costs of implementations. The proposed methodology builds up a propagator tree, with the Pareto checker at the root that is dependent on its leaf propagators evaluating the individual objectives. Due to the increased amount of dominance checks required by the previously introduced partial assignment checking, the thesis proposes further improvements of the archive management and evaluation steps of the optimization. For the former, the application of a Quad–Tree data structure is proposed. Thereby, the dominance checks of partial assignments can be simplified as the archive must not be updated at this point. After a full assignment has been found, the new implementation is known to be non-dominated, and, at this point, only the now dominated design points have to removed from the Pareto archive. Furthermore, due to the tree-based data structure, only a subset of archive solutions have to checked which further improves the performance of the dominance checks.

Section 4.3 finally proposes an optimization for the objective evaluation through safe approximations. In short, an implementation candidate is first evaluated through a safe approxima-

tion before only potentially good solutions are evaluated exactly. A safe approximation is therefor defined to consistently evaluate the corresponding better or worse than the exact method. For a minimization problem, the approximation has to be consistently smaller (i.e., under-approximation) and a maximization problem, the approximation must always be larger (over-approximation) than the exact value would be. Although a high accuracy is not necessarily required for the proposed approach to work correctly, it has been identified that the accuracy has a high impact on the performance of the approach. However, it has been experimentally shown that even a low accuracy does not degrade the performance significantly when compared to a conventional approach that only uses exact evaluations. Furthermore, the approach is flexible in the sense, that it does not have to be applied to every objective function in the problem. If a safe approximation cannot be determined or would not be effective, the objective may be calculated through the conventional exact evaluation without interfering with other objectives.

5.1 Limitations

While the proposed ASPmT-based DSE addresses and improves drawbacks of current state-of-the-art approaches, the proposed solutions are still subject to a number of limitations. These shall be discussed in the following.

The proposed framework follows a strictly systematic approach. Although this guarantees the complete coverage of the design space in theory as no solution candidate is revisited during the search, in reality, most problem instances are too complex to be explored completely. Furthermore, as the search is conducted, in principle, by iterating through the solutions of the problem, the finding of diverse solutions is delayed. The difference between two subsequently found solutions tends to be marginal, and, thus, only local changes are applied to the solutions. In order to advance to more diverse regions in the design space, typically, a large amount of solutions has to be found first. This especially leads to a problem when the search starts in a region where no feasible design candidates are present. Then, even finding an initial solution can be nearly impossible in a reasonable amount of time. While this drawback can be diminished by the evaluation of partial assignments, it still poses a problem for highly complex systems. Furthermore, the evaluation of partial assignments is only possible when the considered objective are assignment monotonic. While applicable for the objective functions considered in the thesis at hand, it may become a problem if further objectives are to be evaluated. For example, when evaluating the reliability of a system, the addition of extra processing or communication resources renders the system potentially more fail-safe. Hence, the objective would not be assignment monotonic as additional decision could improve the quality of the system.

The proposed DSE framework makes purely static decisions. This basically reduces the applicability of the approach to streaming application without any dynamic behavior. Applications with a data-dependent task set can only be considered to a certain degree with annotated WCETs. These are typically pessimistic and cannot provide tight bounds. If many dynamic tasks are present, dynamic binding, routing, and scheduling decisions should be considered at runtime to be able to find optimal design points. This, however, requires different analysis models that have not been discussed in the thesis at hand.

Further limitations regard the specification model. First, the model can only represent strictly periodic tasks. Thus, the definition of sporadic tasks is not possible. Also, the schedulability analysis of the periodic tasks may be pessimistic even for periodic applications where indi-

vidual tasks are not required to start at a fixed frequency. Currently, the analysis assumes that a task restarts exactly after its periodicity. This, however, may not be necessary for all tasks as long as their dependencies are adhered. In turn, the schedulability analysis can falsely yield the invalidity of the taken binding decisions. This becomes apparent if independent applications with varying periodicity are specified. It may be helpful to shift tasks of subsequent iterations into free time slots instead of requiring exact distances of task instances of different iterations.

Second, multicast messages are not directly supported by the proposed model. Sending the same message to multi receivers must be split into multiple individual messages as a workaround. The drawback is a potentially pessimistic schedule. Ideally, both messages could share at least a few hops before they part and take individual routes to their respective destinations. This is not possible in the proposed framework. Hence, the shared hops are processed sequentially. In turn, this may yield pessimistic estimations of the acquired latency.

Third, in the hardware model, especially the on-chip routers are assumed to contain unlimited input buffers to store messages. Whenever multiple messages are routed over a common link at the same time, the connected router only forwards one message at a time while the rest has to be stored at the input buffers. In reality, the buffer size is limited. Thus, implementations containing highly trafficked links may be falsely evaluated in the proposed framework.

Fourth, the framework can exclusively model static behavior. Hence, dynamic behavior imposed by, for example, data dependent algorithms, user input, and the insertion of additional or removal of present applications cannot be modeled. Hence, all information must be available at design time. Furthermore, if at runtime, a component fails, an automatic reconfiguration of the system is not possible. In this case, a new synthesis had to be carried out.

Finally, the evaluation is executed on the basis of abstract properties that are annotated within the specification model. In reality, these properties have to be obtained through thorough investigation. The execution time of a task on different resources, for instance, can be obtained by compiling the task for different architectures. While realistic assumptions are aspired, the data source of these properties is not the primary scope in this thesis. It is assumed that the properties are known when starting the DSE. However, when executed for real applications, and hence, task mappings, it must be guaranteed that the annotations represent the worst-case. Annotating average or best-case properties could render the obtained validity of an implementation useless.

5.2 Future Work

Future work in the area of the thesis at hand should address the limitations discussed above. In particular, a combination of the proposed ASPmT-based synthesis framework and meta-heuristical approaches is expected to advance the field of DSE at the ESL. Similar, to SAT-based techniques used in MOEAs, the proposed synthesis can be used to diminish the problems of the huge design space as well as the problem of finding initial solutions. Compared to SAT, the proposed ASPmT encoding is capable of evaluating not only feasibility constraints but also validity constraints in succinct problem definitions. Furthermore, it has been shown in [129] that the closed-world assumption employed by ASP benefits the decision-making of routing decisions. Therefore, the author of the thesis at hand is convinced that an integration of the here proposed methodology into a state-of-the-art meta-heuristic approach can lead to significant improvements.

A major concern of future work should be put, however, on the support of dynamism. This

not only includes the support for data-dependent application but also the dynamic addition of new functionality at runtime as well as the automatic handling of hardware failures. This requires a paradigm shift where some decisions are postponed to the runtime of the system. In turn, it necessitates the adaption of both the specification and the exploration model. To choose the right model of computation for such design problems, future work must emphasize on the trade-off between expressiveness and analyzability of the respective models. That is, it must be evaluated which level of detail is appropriate at the ESL to gather meaningful results while maintaining a high exploration performance. As a result, the evaluation methods must fit the model of computation. Here, the (max,+)-algebra in combination with scenario-aware dataflow graphs [139] provide a promising starting point for further studies. In this context, it shall be clarified whether the (max,+)-algebra can be applied to partial assignments to accelerate exploration.

More recently, embedded systems are developed to support more general purpose tasks. This poses another interesting field of study for future work. In the thesis at hand, the hardware platform is synthesized with one particular application in mind. However, if the hardware platform shall support a wider range of applications, the synthesis step must be adapted. Defining, for example, not a single application but instead an entire class of applications provides interesting questions. How can the platform be optimized for a set of applications? What model of computation can be used to define application classes? Which communication infrastructure shall be derived to trade-off potential contention and implementation complexity? A further important aspect in the context of supporting more general purpose tasks is the handling of mixed-criticality task sets. That is, the execution of critical tasks has to be guaranteed while non-critical tasks may be delayed. The combination of predicable processors (i.e., without branch prediction, caches, etc.) with general purpose processors may be one possible solution to fulfill these requirements.

Considering the hardware platform, a further focus of future work may be the abandonment of hardware templates in favor of hardware libraries. Hence, the basic structure of the platform is not predefined but instead, the topology of individual hardware devices, i.e., computational and communication devices, is decided by the synthesis. This way, the author expects a better utilization of the hardware resources and less contention in the communication infrastructure. However, this results in a potentially unbounded design space. Hence, future work in this area must define constraints that make the problem manageable.

Finally, a tighter integration of expert knowledge in the design process may provide great opportunities. For example, a decision maker could steer the solving in an interactive process to study particular regions of the designs space more thoroughly or force global changes in the decision-making process. Hence, the designer could directly trade off the search regarding diversity and convergence. It is, however, unclear whether this methodology can still provide the guarantees of a systematic search, i.e., a complete and correct search where no solution is visited twice.

As a summary, the outlook of possible future work shows that there are still many open questions in the area of the design space exploration at the electronic system level. Although, the thesis at hand provided some optimizations, especially regarding the systematic exploration, the topic provides many more problems to be solved by future research.

A
Appendix

The appendix contains supplementary data to the experiments conducted in the individual evaluation sections.

A.1 Synthesis – Experimental Results

Compared to the results in Table 3.2, the following two Tables A.1 and A.2 not only consider solved but all synthesis runs. Therefore, the average runtime, number of choices, and number of encountered conflicts are calculated for the total number of runs and, in parentheses, only for the solved instances.

Table A.1: Detailed results of the synthesis experiments for the full assignment runs conducted in Section 3.4.2

Hardness	Group	Solved	Time (Solved) [s]		Choices (Solved)		Conflicts (Solved)	
			min	max	min	max	min	max
1.00	I	6	720.190 (0.319)		$6.94 \cdot 10^7$ ($7.83 \cdot 10^3$)		$1.03 \cdot 10^6$ ($1.38 \cdot 10^3$)	
			0.177	1,800.0	$1.26 \cdot 10^3$	$1.99 \cdot 10^8$	$4.30 \cdot 10^2$	$3.59 \cdot 10^6$
	II	9	596.806 (463.117)		$3.05 \cdot 10^7$ ($2.33 \cdot 10^7$)		$3.35 \cdot 10^5$ ($2.64 \cdot 10^5$)	
			0.881	1,800.0	$1.56 \cdot 10^3$	$9.52 \cdot 10^7$	$4.00 \cdot 10^0$	$9.71 \cdot 10^5$
	III	6	725.192 (8.654)		$1.80 \cdot 10^7$ ($6.35 \cdot 10^4$)		$1.85 \cdot 10^5$ ($5.67 \cdot 10^1$)	
			5.577	1,800.0	$3.49 \cdot 10^4$	$4.65 \cdot 10^7$	$2.30 \cdot 10^1$	$5.08 \cdot 10^5$
	IV	1	1,622.804 (28.017)		$2.59 \cdot 10^7$ ($9.24 \cdot 10^4$)		$3.17 \cdot 10^5$ ($7.30 \cdot 10^1$)	
			28.017	1,800.0	$9.24 \cdot 10^4$	$3.08 \cdot 10^7$	$7.30 \cdot 10^1$	$4.01 \cdot 10^5$
	V	0	1,800.0 (–)		$2.33 \cdot 10^7$ (–)		$2.91 \cdot 10^5$ (–)	
			1,800.0	1,800.0	$2.05 \cdot 10^7$	$2.88 \cdot 10^7$	$2.24 \cdot 10^5$	$3.56 \cdot 10^5$
0.94	I	5	900.450 (0.901)		$7.77 \cdot 10^7$ ($3.97 \cdot 10^4$)		$1.36 \cdot 10^6$ ($3.76 \cdot 10^3$)	
			0.221	1,800.0	$3.62 \cdot 10^3$	$1.86 \cdot 10^8$	$7.46 \cdot 10^2$	$3.59 \cdot 10^6$
	II	5	1,265.440 (730.881)		$6.38 \cdot 10^7$ ($3.55 \cdot 10^7$)		$6.83 \cdot 10^5$ ($4.16 \cdot 10^5$)	
			0.874	1,800.0	$1.49 \cdot 10^3$	$9.41 \cdot 10^7$	$5.00 \cdot 10^0$	$1.07 \cdot 10^6$
	III	3	1,262.033 (6.775)		$3.02 \cdot 10^7$ ($5.75 \cdot 10^4$)		$3.05 \cdot 10^5$ ($5.70 \cdot 10^1$)	
			5.314	1,800.0	$3.42 \cdot 10^4$	$4.59 \cdot 10^7$	$2.40 \cdot 10^1$	$4.69 \cdot 10^5$
	IV	1	1,621.663 (16.601)		$2.61 \cdot 10^7$ ($6.87 \cdot 10^4$)		$3.05 \cdot 10^5$ ($4.00 \cdot 10^1$)	
			16.601	1,800.0	$6.87 \cdot 10^4$	$3.06 \cdot 10^7$	$4.00 \cdot 10^1$	$3.95 \cdot 10^5$
	V	0	1,800.0 (–)		$2.31 \cdot 10^7$ (–)		$2.89 \cdot 10^5$ (–)	
			1,800.0	1,800.0	$1.94 \cdot 10^7$	$2.45 \cdot 10^7$	$2.13 \cdot 10^5$	$3.45 \cdot 10^5$
0.88	I	4	1,154.568 (186.428)		$1.17 \cdot 10^8$ ($1.57 \cdot 10^7$)		$1.74 \cdot 10^6$ ($1.80 \cdot 10^5$)	
			1.214	1,800.0	$5.26 \cdot 10^4$	$2.25 \cdot 10^8$	$5.21 \cdot 10^3$	$3.58 \cdot 10^6$
	II	3	1,367.634 (358.781)		$6.74 \cdot 10^7$ ($1.70 \cdot 10^7$)		$6.88 \cdot 10^5$ ($1.92 \cdot 10^5$)	
			0.906	1,800.0	$2.00 \cdot 10^3$	$9.29 \cdot 10^7$	$8.00 \cdot 10^0$	$1.04 \cdot 10^6$
	III	3	1,262.271 (7.570)		$3.05 \cdot 10^7$ ($5.23 \cdot 10^4$)		$3.17 \cdot 10^5$ ($5.37 \cdot 10^1$)	
			6.119	1,800.0	$4.36 \cdot 10^4$	$4.57 \cdot 10^7$	$3.70 \cdot 10^1$	$4.96 \cdot 10^5$

Table A.1: (continued)

Hardness	Group	Solved	Time (Solved) [s]		Choices (Solved)		Conflicts (Solved)	
			min	max	min	max	min	max
	IV	0	1,800.0 (–)		$2.87 \cdot 10^7$ (–)		$3.44 \cdot 10^5$ (–)	
			1,800.0	1,800.0	$2.73 \cdot 10^7$	$3.08 \cdot 10^7$	$2.93 \cdot 10^5$	$3.99 \cdot 10^5$
	V	0	1,800.0 (–)		$2.32 \cdot 10^7$ (–)		$2.86 \cdot 10^5$ (–)	
			1,800.0	1,800.0	$2.06 \cdot 10^7$	$2.51 \cdot 10^7$	$2.30 \cdot 10^5$	$3.31 \cdot 10^5$
0.83	I	1	1,620.397 (4.027)		$1.59 \cdot 10^8$ ($2.13 \cdot 10^5$)		$2.49 \cdot 10^6$ ($1.32 \cdot 10^4$)	
			4.027	1,800.0	$2.13 \cdot 10^5$	$2.27 \cdot 10^8$	$1.32 \cdot 10^4$	$3.59 \cdot 10^6$
	II	0	1,800.0 (–)		$8.96 \cdot 10^7$ (–)		$9.63 \cdot 10^5$ (–)	
			1,800.0	1,800.0	$8.36 \cdot 10^7$	$9.77 \cdot 10^7$	$8.08 \cdot 10^5$	$1.09 \cdot 10^6$
	III	1	1,620.656 (6.560)		$3.85 \cdot 10^7$ ($5.04 \cdot 10^4$)		$4.01 \cdot 10^5$ ($8.60 \cdot 10^1$)	
			6.560	1,800.0	$5.04 \cdot 10^4$	$4.47 \cdot 10^7$	$8.60 \cdot 10^1$	$4.80 \cdot 10^5$
	IV	0	1,800.0 (–)		$2.91 \cdot 10^7$ (–)		$3.41 \cdot 10^5$ (–)	
			1,800.0	1,800.0	$2.73 \cdot 10^7$	$3.10 \cdot 10^7$	$2.79 \cdot 10^5$	$4.06 \cdot 10^5$
	V	0	1,800.0 (–)		$2.32 \cdot 10^7$ (–)		$2.89 \cdot 10^5$ (–)	
			1,800.0	1,800.0	$2.03 \cdot 10^7$	$2.53 \cdot 10^7$	$2.34 \cdot 10^5$	$3.34 \cdot 10^5$

Table A.2: Detailed results of the synthesis experiments for the partial assignment runs conducted in Section 3.4.2

Hardness	Group	Solved	Time (Solved) [s]		Choices (Solved)		Conflicts (Solved)	
			min	max	min	max	min	max
1.00	I	9	180.941 (1.045)		$2.47 \cdot 10^6$ ($1.82 \cdot 10^4$)		$2.21 \cdot 10^6$ ($9.87 \cdot 10^3$)	
			0.197	1,800.0	$3.71 \cdot 10^3$	$2.46 \cdot 10^7$	$6.88 \cdot 10^2$	$2.20 \cdot 10^7$
	II	10	135.955 (135.955)		$1.17 \cdot 10^6$ ($1.17 \cdot 10^6$)		$9.73 \cdot 10^5$ ($9.73 \cdot 10^5$)	
			0.708	697.321	$2.98 \cdot 10^2$	$6.45 \cdot 10^6$	$2.00 \cdot 10^0$	$5.60 \cdot 10^6$
	III	10	4.723 (4.723)		$1.25 \cdot 10^4$ ($1.25 \cdot 10^4$)		$4.81 \cdot 10^3$ ($4.81 \cdot 10^3$)	
			2.133	26.548	$4.01 \cdot 10^2$	$1.20 \cdot 10^5$	$2.00 \cdot 10^0$	$4.80 \cdot 10^4$
	IV	10	9.038 (9.038)		$1.28 \cdot 10^4$ ($1.28 \cdot 10^4$)		$1.40 \cdot 10^3$ ($1.40 \cdot 10^3$)	
			4.751	45.926	$5.82 \cdot 10^2$	$1.20 \cdot 10^5$	$1.00 \cdot 10^0$	$1.37 \cdot 10^4$
	V	10	20.350 (20.350)		$2.81 \cdot 10^4$ ($2.81 \cdot 10^4$)		$2.27 \cdot 10^3$ ($2.27 \cdot 10^3$)	
			6.732	135.588	$6.96 \cdot 10^2$	$2.61 \cdot 10^5$	$3.00 \cdot 10^0$	$2.19 \cdot 10^4$
0.94	I	9	239.049 (65.610)		$4.64 \cdot 10^6$ ($8.40 \cdot 10^5$)		$4.12 \cdot 10^6$ ($7.14 \cdot 10^5$)	
			0.237	1,800.000	$6.74 \cdot 10^3$	$3.88 \cdot 10^7$	$1.39 \cdot 10^3$	$3.47 \cdot 10^7$
	II	10	100.205 (100.205)		$8.23 \cdot 10^5$ ($8.23 \cdot 10^5$)		$6.60 \cdot 10^5$ ($6.60 \cdot 10^5$)	
			0.759	605.636	$3.17 \cdot 10^2$	$5.11 \cdot 10^6$	$2.00 \cdot 10^0$	$4.31 \cdot 10^6$
	III	10	5.211 (5.211)		$1.46 \cdot 10^4$ ($1.46 \cdot 10^4$)		$4.88 \cdot 10^3$ ($4.88 \cdot 10^3$)	
			2.138	31.526	$4.03 \cdot 10^2$	$1.42 \cdot 10^5$	$0.00 \cdot 10^0$	$4.88 \cdot 10^4$
	IV	10	4.939 (4.939)		$8.97 \cdot 10^2$ ($8.97 \cdot 10^2$)		$2.78 \cdot 10^1$ ($2.78 \cdot 10^1$)	
			4.757	5.436	$5.82 \cdot 10^2$	$3.02 \cdot 10^3$	$2.00 \cdot 10^0$	$1.91 \cdot 10^2$
	V	10	30.128 (30.128)		$4.88 \cdot 10^4$ ($4.88 \cdot 10^4$)		$5.25 \cdot 10^3$ ($5.25 \cdot 10^3$)	
			6.713	159.927	$6.79 \cdot 10^2$	$3.24 \cdot 10^5$	$5.00 \cdot 10^0$	$3.99 \cdot 10^4$
0.88	I	9	380.465 (222.738)		$6.19 \cdot 10^6$ ($4.18 \cdot 10^6$)		$5.39 \cdot 10^6$ ($3.58 \cdot 10^6$)	
			0.432	1,800.0	$1.89 \cdot 10^4$	$2.75 \cdot 10^7$	$4.48 \cdot 10^3$	$2.41 \cdot 10^7$
	II	8	554.573 (243.216)		$4.18 \cdot 10^6$ ($1.78 \cdot 10^6$)		$3.66 \cdot 10^6$ ($1.53 \cdot 10^6$)	
			0.769	1,800.0	$3.18 \cdot 10^2$	$1.39 \cdot 10^7$	$3.00 \cdot 10^0$	$1.22 \cdot 10^7$
	III	10	25.021 (25.021)		$1.06 \cdot 10^5$ ($1.06 \cdot 10^5$)		$5.24 \cdot 10^4$ ($5.24 \cdot 10^4$)	
			2.163	181.081	$4.01 \cdot 10^2$	$8.25 \cdot 10^5$	$1.00 \cdot 10^0$	$4.46 \cdot 10^5$
	IV	10	5.101 (5.101)		$1.31 \cdot 10^3$ ($1.31 \cdot 10^3$)		$6.90 \cdot 10^1$ ($6.90 \cdot 10^1$)	
			4.723	6.675	$6.01 \cdot 10^2$	$4.73 \cdot 10^3$	$1.00 \cdot 10^0$	$4.09 \cdot 10^2$
	V	10	32.800 (32.800)		$6.30 \cdot 10^4$ ($6.30 \cdot 10^4$)		$8.27 \cdot 10^3$ ($8.27 \cdot 10^3$)	
			6.836	255.519	$7.61 \cdot 10^2$	$6.01 \cdot 10^5$	$5.00 \cdot 10^0$	$8.04 \cdot 10^4$

Table A.2: (continued)

Hardness	Group	Solved	Time (Solved) [s] min	Time (Solved) [s] max	Choices (Solved) min	Choices (Solved) max	Conflicts (Solved) min	Conflicts (Solved) max
0.83	I	5	1,073.396 (346.815) 0.239	1,800.0	$1.75 \cdot 10^7$ ($4.58 \cdot 10^6$) $8.07 \cdot 10^3$	$3.55 \cdot 10^7$	$1.56 \cdot 10^7$ ($4.07 \cdot 10^6$) $1.38 \cdot 10^3$	$3.17 \cdot 10^7$
	II	5	1,055.025 (310.054) 26.969	1,800.0	$8.41 \cdot 10^6$ ($2.60 \cdot 10^6$) $2.36 \cdot 10^5$	$1.51 \cdot 10^7$	$7.18 \cdot 10^6$ ($2.12 \cdot 10^6$) $1.62 \cdot 10^5$	$1.30 \cdot 10^7$
	III	10	99.586 (99.586) 2.188	805.414	$4.80 \cdot 10^5$ ($4.80 \cdot 10^5$) $4.21 \cdot 10^2$	$4.01 \cdot 10^6$	$3.56 \cdot 10^5$ ($3.56 \cdot 10^5$) $0.00 \cdot 10^0$	$3.18 \cdot 10^6$
	IV	10	10.196 (10.196) 4.735	55.429	$1.54 \cdot 10^4$ ($1.54 \cdot 10^4$) $5.97 \cdot 10^2$	$1.42 \cdot 10^5$	$2.23 \cdot 10^3$ ($2.23 \cdot 10^3$) $2.00 \cdot 10^0$	$2.17 \cdot 10^4$
	V	10	73.721 (73.721) 6.875	202.209	$1.53 \cdot 10^5$ ($1.53 \cdot 10^5$) $7.06 \cdot 10^2$	$4.46 \cdot 10^5$	$1.81 \cdot 10^4$ ($1.81 \cdot 10^4$) $5.00 \cdot 10^0$	$6.72 \cdot 10^4$

A.2 Design Space Exploration – Experimental Results

The following Table A.3 contains the results of all exploration runs conducted in Section 4.1.3. It is an extended version of Table 4.1.

Table A.3: Extended results of the evaluation of the optimization framework regarding the quality of obtained solutions (Table 4.1).

$\|A\|$	$\|S\|$	$\|P\|$	Platform	I	$\mathcal{H}$ ALR	$\mathcal{H}$ SPR	$\mathcal{H}$ DOR	$\mathcal{T}_{breadth}$ ALR	$\mathcal{T}_{breadth}$ SPR	$\mathcal{T}_{breadth}$ DOR	$\mathcal{T}_{depth}$ ALR	$\mathcal{T}_{depth}$ SPR	$\mathcal{T}_{depth}$ DOR
1	8	6	$3 \times 3 \times 1$	ε	1.487	1.280	1.139	1.487	**1.131**	1.275	1.877	1.505	1.894
				H	2.472	2.473	2.998	2.989	3.531	**3.768**	2.473	2.471	2.473
1	4	10	$3 \times 3 \times 1$	ε	1.564	**1.018**	1.128	1.749	1.203	1.224	1.749	1.223	1.367
				H	2.473	2.473	2.473	2.473	3.452	**3.522**	2.473	2.473	2.471
1	6	6	$3 \times 3 \times 1$	ε	inf	1.419	1.324	1.590	**1.062**	1.097	1.784	1.687	1.387
				H	0.000	2.473	2.472	2.585	3.652	**3.927**	2.473	2.473	2.468
1	10	6	$3 \times 3 \times 1$	ε	inf	1.282	**1.074**	inf	1.156	1.109	inf	1.687	1.282
				H	0.000	2.473	2.473	0.000	3.524	**3.628**	0.000	2.473	2.473
1	10	10	$3 \times 3 \times 1$	ε	inf	1.248	1.288	inf	1.153	**1.085**	inf	1.437	1.391
				H	0.000	2.472	2.473	0.000	**3.528**	3.514	0.000	2.473	2.471
2	7	13	$3 \times 3 \times 1$	ε	1.479	1.421	1.879	1.340	**1.098**	1.215	1.624	1.879	1.640
				H	2.473	2.473	2.373	3.293	3.630	**3.708**	2.473	2.473	2.473
2	8	8	$3 \times 3 \times 1$	ε	inf	1.556	1.272	inf	1.174	**1.077**	inf	1.556	1.556
				H	0.000	2.470	2.473	0.000	3.539	**3.618**	0.000	2.472	2.466
2	9	12	$3 \times 3 \times 1$	ε	1.642	1.352	1.216	1.486	1.295	**1.196**	1.684	1.618	1.678
				H	2.472	2.469	2.473	2.619	3.256	**3.613**	2.473	2.472	2.473
2	11	9	$3 \times 3 \times 1$	ε	**1.099**	1.194	1.117	1.269	1.178	1.174	1.767	1.767	1.761
				H	2.473	2.473	2.473	3.091	**3.826**	3.741	2.473	2.472	2.472
2	12	13	$3 \times 3 \times 1$	ε	2.153	2.088	1.275	1.270	**1.055**	1.222	1.750	1.494	1.746
				H	2.472	2.450	2.473	3.096	**3.718**	3.430	2.473	2.473	2.473
2	13	16	$3 \times 3 \times 1$	ε	1.602	1.578	1.356	1.271	**1.153**	1.196	1.662	1.425	1.779
				H	2.467	2.473	2.473	3.130	**3.586**	3.415	2.473	2.473	2.466
3	7	16	$3 \times 3 \times 2$	ε	1.945	1.261	1.144	1.917	1.139	**1.032**	1.908	1.669	1.931
				H	2.473	2.473	2.473	2.566	3.182	**3.369**	2.473	2.473	2.469
3	10	16	$3 \times 3 \times 2$	ε	1.925	1.434	**1.000**	1.773	1.594	1.529	1.930	1.608	1.757
				H	2.473	2.473	2.473	2.745	**3.061**	2.949	2.473	2.473	2.473
3	7	24	$3 \times 3 \times 2$	ε	2.281	2.241	**1.098**	2.152	1.811	1.134	2.279	2.233	1.401
				H	2.473	2.392	2.466	2.512	**3.164**	2.753	2.473	2.473	2.469
3	11	22	$3 \times 3 \times 2$	ε	1.806	1.611	1.342	1.737	1.370	**1.155**	1.826	1.827	1.746
				H	2.473	2.472	2.473	2.825	**3.285**	3.203	2.473	2.473	2.473

Table A.3: (continued)

$\|A\|$	$\|S\|$	$\|P\|$	Platform	I	$\mathcal{H}$ ALR	$\mathcal{H}$ SPR	$\mathcal{H}$ DOR	$\mathcal{T}_{breadth}$ ALR	$\mathcal{T}_{breadth}$ SPR	$\mathcal{T}_{breadth}$ DOR	$\mathcal{T}_{depth}$ ALR	$\mathcal{T}_{depth}$ SPR	$\mathcal{T}_{depth}$ DOR
3	22	11	3 × 3 × 2	ε	1.785	1.592	1.306	1.686	**1.073**	1.142	1.786	1.786	1.732
				H	2.473	2.464	2.473	2.882	**3.522**	2.976	2.473	2.473	2.473
3	17	22	3 × 3 × 2	ε	1.534	1.441	**1.085**	1.469	1.109	1.152	1.534	1.532	1.152
				H	2.473	2.452	2.473	2.706	**3.254**	3.098	2.473	2.470	2.472
3	19	24	3 × 3 × 2	ε	2.283	2.185	1.538	2.274	1.433	**1.054**	2.106	1.950	1.792
				H	2.473	2.472	2.473	2.566	2.579	**3.188**	2.473	2.473	2.462
3	22	27	3 × 3 × 2	ε	1.855	1.650	1.477	1.835	**1.279**	1.309	1.855	1.637	1.594
				H	2.473	2.463	2.473	2.501	**3.166**	2.572	2.473	2.473	2.473
4	9	16	3 × 3 × 3	ε	1.919	1.703	**1.000**	1.906	1.272	1.272	1.913	1.919	1.272
				H	2.473	2.466	2.473	2.645	**3.155**	3.153	2.473	2.473	2.468
4	15	21	3 × 3 × 3	ε	1.289	1.283	1.168	1.283	**1.126**	1.191	1.289	1.284	1.243
				H	2.473	2.467	2.473	2.811	**3.422**	3.182	2.473	2.444	2.472
4	12	19	3 × 3 × 3	ε	2.074	1.617	**1.078**	2.054	1.130	1.172	2.071	2.071	1.225
				H	2.473	2.473	2.473	2.705	**3.467**	3.041	2.473	2.473	2.471
4	16	24	3 × 3 × 3	ε	1.915	1.899	**1.067**	1.905	1.071	1.073	1.918	1.795	1.733
				H	2.473	2.454	2.473	2.475	**3.322**	3.151	2.473	2.441	2.473
4	21	20	3 × 3 × 3	ε	1.610	1.614	**1.000**	1.510	1.301	1.151	1.615	1.579	1.435
				H	2.473	2.471	2.473	2.743	3.043	**3.110**	2.473	2.472	2.473
4	18	28	3 × 3 × 3	ε	2.177	2.134	1.546	2.177	1.265	**1.057**	2.170	1.686	1.687
				H	2.473	2.471	2.473	2.534	**3.214**	2.655	2.473	2.471	2.473
4	30	19	3 × 3 × 3	ε	1.927	1.868	**1.015**	1.900	1.172	1.070	1.932	1.693	1.930
				H	2.473	2.473	2.473	2.602	2.832	**2.861**	2.473	2.473	2.473
4	24	38	3 × 3 × 3	ε	inf	1.892	1.533	inf	1.515	**1.103**	1.895	1.894	1.837
				H	0.000	2.472	2.472	0.000	2.915	**2.976**	2.473	2.473	2.473

A.3 Approximation – Experimental Results

The Tables A.4 and A.5 contain the data of the Figures 4.18 and 4.19, respectively. The data was obtained through the DSE runs conducted to evaluate the proposed approximation-based approach in Section 4.3.3.

Table A.4: Detailed data regarding the Filter Ratio of Figure 4.18.

Group ID	FR (High ERR) $\bar{x}_{geom}$	Max	Min	σ^2	FR (Medium ERR) $\bar{x}_{geom}$	Max	Min	σ^2	FR (Low ERR) $\bar{x}_{geom}$	Max	Min	σ^2
I	0.983	1.000	0.887	0.001	0.110	1.000	0.000	0.183	0.015	0.200	0.000	0.005
II	0.865	1.000	0.550	0.019	0.016	0.676	0.000	0.041	0.001	0.752	0.000	0.051
III	0.804	1.000	0.267	0.045	0.122	0.955	0.000	0.086	0.001	0.018	0.000	0.000
IV	0.747	0.993	0.438	0.050	0.116	0.988	0.000	0.138	0.001	0.002	0.000	0.000
V	0.525	0.991	0.052	0.075	0.140	0.982	0.003	0.080	0.002	0.017	0.001	0.000
VI	0.686	0.982	0.294	0.073	0.146	0.973	0.002	0.112	0.002	0.003	0.001	0.000
VII	0.710	0.999	0.234	0.074	0.493	0.996	0.134	0.085	0.003	0.024	0.001	0.000
VIII	0.322	0.962	0.001	0.101	0.017	0.915	0.001	0.134	0.003	0.056	0.002	0.000
IX	0.731	0.977	0.201	0.049	0.158	0.854	0.004	0.122	0.011	0.082	0.004	0.001
X	0.803	0.997	0.387	0.042	0.367	0.970	0.003	0.084	0.004	0.027	0.002	0.000
XI	0.613	0.996	0.209	0.077	0.195	0.979	0.009	0.069	0.016	0.292	0.005	0.009
XII	0.945	1.000	0.865	0.002	0.696	1.000	0.161	0.056	0.177	1.000	0.007	0.116

Table A.5: Detailed data regarding the performance improvement shown in Figure 4.19.

Group ID	Improvement (High ERR)				Improvement (Medium ERR)				Improvement (Low ERR)			
	$\bar{x}_{geom}$	Max	Min	σ^2	$\bar{x}_{geom}$	Max	Min	σ^2	$\bar{x}_{geom}$	Max	Min	σ^2
I	5.942	6.847	4.060	0.738	1.940	6.849	0.997	5.459	0.999	1.000	0.997	0.000
II	3.004	4.374	1.046	1.041	1.048	1.238	0.998	0.007	0.994	1.000	0.988	0.000
III	2.123	2.564	1.184	0.177	1.279	2.375	0.999	0.164	0.996	0.999	0.991	0.000
IV	2.120	2.809	1.399	0.295	1.475	2.784	1.000	0.474	1.001	1.004	1.000	0.000
V	1.632	2.342	1.035	0.156	1.225	2.261	1.001	0.128	1.004	1.009	0.998	0.000
VI	1.679	2.032	1.176	0.116	1.321	2.020	1.000	0.111	1.000	1.001	0.997	0.000
VII	1.768	2.171	1.136	0.126	1.496	2.170	1.072	0.131	1.002	1.010	0.997	0.000
VIII	1.448	1.907	1.001	0.087	1.153	1.813	0.997	0.101	0.998	1.001	0.994	0.000
IX	1.485	1.679	1.084	0.030	1.230	1.548	0.998	0.048	0.998	1.004	0.994	0.000
X	1.582	1.776	1.203	0.041	1.380	1.722	1.006	0.047	1.003	1.007	0.995	0.000
XI	1.304	1.531	1.085	0.030	1.115	1.523	0.998	0.023	1.001	1.006	0.995	0.000
XII	1.203	1.426	0.998	0.024	1.123	1.380	1.000	0.018	0.997	1.007	0.990	0.000

Bibliography

All References

[1] Kai Neubauer, Christian Haubelt, and Michael Glaß. "Supporting Composition in Symbolic System Synthesis". In: *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. Samos, Greece, July 2016, pp. 132–139. DOI: 10.1109/SAMOS.2016.7818340.

[2] Kai Neubauer, Philipp Wanko, Torsten Schaub, and Christian Haubelt. "Enhancing Symbolic System Synthesis through ASPmT with Partial Assignment Evaluation". In: *Design, Automation and Test in Europe Conference (DATE)*. Lausanne, Switzerland, Mar. 2017, pp. 306–309. DOI: 10.23919/DATE.2017.7927005.

[3] Kai Neubauer, Christian Haubelt, Philipp Wanko, and Torsten Schaub. "Utilizing Quad-Trees for Efficient Design Space Exploration with Partial Assignment Evaluation". In: *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jeju, Korea, Jan. 2018, pp. 434–439. DOI: 10.1109/ASPDAC.2018.8297362.

[4] Kai Neubauer, Philipp Wanko, Torsten Schaub, and Christian Haubelt. "Exact Multi-Objective Design Space Exploration using ASPmT". In: *Design, Automation and Test in Europe Conference (DATE)*. Dresden, Germany, Mar. 2018, pp. 257–260. DOI: 10.23919/DATE.2018.8342014.

[5] Kai Neubauer, Christian Haubelt, Philipp Wanko, and Torsten Schaub. "Systematic Test Case Instance Generation for the Assessment of System-level Design Space Exploration Approaches". In: *21. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*. Tübingen, Germany, Mar. 2018. DOI: 10.15496/publikation-25685.

[6] Kai Neubauer, Christian Haubelt, Philipp Wanko, and Torsten Schaub. "Work-in-Progress: On Leveraging Approximations for Exact System-level Design Space Exploration". In: *International Conference on Hardware Software Codesign and System Synthesis (CODES/ISSS)*. Sept. 2018, pp. 1–2. DOI: 10.1109/CODESISSS.2018.8525974.

[7] Kai Neubauer, Benjamin Beichler, and Christian Haubelt. "Exact Design Space Exploration Based on Consistent Approximations". In: *Electronics* 9.7 (June 2020), p. 1057. ISSN: 2079-9292. DOI: 10.3390/electronics9071057.

[8] Christian Haubelt, Kai Neubauer, Torsten Schaub, and Philipp Wanko. "Design Space Exploration with Answer Set Programming". In: *KI - Künstliche Intelligenz*. Vol. 32. 2-3. Berlin Heidelberg: Springer Nature, May 2018, pp. 205–206. DOI: 10.1007/s13218-018-0530-3.

[9] Joachim Falk, Kai Neubauer, Christian Haubelt, Christian Zebelein, and Jürgen Teich. "Integrated Modeling Using Finite State Machines and Dataflow Graphs". In: *Handbook of Signal Processing Systems*. Ed. by S.S. Bhattacharyya, E.F. Deprettere, R. Leupers, and J. Takala. Third Ed. Springer International Publishing, Oct. 2019, pp. 825–864. DOI: 10.1007/978-3-319-91734-4_23.

[10] Luise Müller, Kai Neubauer, and Christian Haubelt. "Exploiting Similarity in Evolutionary Product Design for Improved Design Space Exploration". In: *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. July 2021.

[11] Jim Turley. *Embedded Processors by the Numbers*. Electronic Engineering Times. URL: https://www.eetimes.com/embedded-processors-by-the-numbers/. Jan. 1999. Last accessed: Sep. 15 2020.

[12] Michael Barr. *Real men program in C*. embedded.com. URL: https://www.embedded.com/real-men-program-in-c/. Aug. 2009. Last accessed: Jun. 06 2021.

[13] Santosh G. Abraham, B. Ramakrishna Rau, and Robert Schreiber. "Fast design space exploration through validity and quality filtering of subsystem designs". In: *HP Laboratories Technical Report* 98 (2000).

[14] Christian Haubelt and Torsten Schaub. *Skalierbare Entwurfsraumexploration mit Antwortmengenprogrammierung*. Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 269264143. URL: https://gepris.dfg.de/gepris/projekt/269264143. 2015. Last accessed: Jan. 19 2022.

[15] Heinz Zemanek. "Zuse, Konrad". In: *Encyclopedia of Computer Science*. Ed. by Anthony Ralston and Edwin D. Reilly. GBR: John Wiley and Sons Ltd., 2003. Chap. Encyclopedia, pp. 1877–1878. ISBN: 0470864125.

[16] Jürgen Teich. "Hardware/Software Codesign: The past, the present, and predicting the future". In: *Proceedings of the IEEE* 100.Special Centennial Issue (2012), pp. 1411–1430. DOI: 10.1109/JPROC.2011.2182009.

[17] *IEEE Standard VHDL Language Reference Manual*. IEEE Std 1076-1987. 1987, pp. 1–218. DOI: 10.1109/IEEESTD.1988.122645.

[18] *IEEE Standard for Verilog Hardware Description Language*. IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001). 2006, pp. 1–590. DOI: 10.1109/IEEESTD.2006.99495.

[19] *IEEE Standard for Standard SystemC Language Reference Manual*. IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005). 2012. DOI: 10.1109/IEEESTD.2012.6134619.

[20] Christian Zebelein, Christian Haubelt, Joachim Falk, and Jürgen Teich. "Model-Based Representation of Schedules for Dataflow Graphs". In: *Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*. Mar. 2013, pp. 105–116. ISBN: 978-3-86009-147-0.

[21] Sander Stuijk, Marc Geilen, and Twan Basten. "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs". In: *Design Automation Conference (DAC)*. July 2006, pp. 899–904. DOI: 10.1145/1146909.1147138.

[22] Weichen Liu, Zonghua Gu, Jiang Xu, Yu Wang, and Mingxuan Yuan. "An Efficient Technique for Analysis of Minimal Buffer Requirements of Synchronous Dataflow Graphs with Model Checking". In: *International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS)*. Oct. 2009, pp. 61–70. DOI: 10.1145/1629435.1629445.

[23] Simone Casale Brunet, Marco Mattavelli, and Jorn W. Janneck. "Buffer optimization based on critical path analysis of a dataflow program design". In: *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*. May 2013, pp. 1384–1387. DOI: 10.1109/ISCAS.2013.6572113.

[24] Andreas Gerstlauer, Christian Haubelt, Andy D. Pimentel, Todor P. Stefanov, Daniel D. Gajski, and Jürgen Teich. "Electronic System-Level Synthesis Methodologies". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.10 (Sept. 2009), pp. 1517–1530. DOI: 10.1109/TCAD.2009.2026356.

[25] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Third Annual ACM Symposium on Theory of Computing*. STOC '71. New York, NY, USA: Association for Computing Machinery, 1971, pp. 151–158. ISBN: 9781450374644. DOI: 10.1145/800157.805047.

[26] Martin Davis and Hilary Putnam. "A Computing Procedure for Quantification Theory". In: *Journal of the ACM* 7.3 (July 1960), pp. 201–215. DOI: 10.1145/321033.321034.

[27] Martin Davis, George Logemann, and Donald Loveland. "A Machine Program for Theorem-Proving". In: *Communications of the ACM* 5.7 (July 1962), pp. 394–397. DOI: 10.1145/368273.368557.

[28] Grigori Samuilowitsch Tseitin. "On the Complexity of Derivation in Propositional Calculus". In: *Automation of Reasoning: 2*. Ed. by Jörg H. Siekmann and Graham Wrightson. Springer Berlin Heidelberg, 1983. Chap. Classical Papers on Computational Logic 1967–1970, pp. 466–483. DOI: 10.1007/978-3-642-81955-1_28.

[29] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. "Chaff: Engineering an Efficient SAT Solver". In: *Design Automation Conference (DAC)*. Association for Computing Machinery, 2001, pp. 530–535. DOI: 10.1145/378239.379017.

[30] Niklas Eén and Niklas Sörensson. "An Extensible SAT-solver". In: *Theory and Applications of Satisfiability Testing*. Springer Berlin Heidelberg, 2004, pp. 502–518. DOI: 10.1007/978-3-540-24605-3_37.

[31] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. "Boolean Satisfiability Solvers and Their Applications in Model Checking". In: *Proceedings of the IEEE* 103.11 (2015), pp. 2021–2035. DOI: 10.1109/JPROC.2015.2455034.

[32] Lintao Zhang and Sharad Malik. "The Quest for Efficient Boolean Satisfiability Solvers". In: *Computer Aided Verification*. Springer Berlin Heidelberg, 2002, pp. 17–36. ISBN: 978-3-540-45657-5. DOI: 10.1007/3-540-45657-0_2.

[33] Eugene Goldberg and Yakov Novikov. "BerkMin: A fast and robust Sat-solver". In: *Discrete Applied Mathematics* 155.12 (2007), pp. 1549–1561. DOI: 10.1016/j.dam.2006.10.007.

[34] Christian Haubelt, Rainer Feldmann, Jürgen Teich, and Burkhard Monien. "SAT-based techniques in system synthesis". In: *Design, Automation and Test in Europe Conference (DATE)*. 2003, pp. 1168–1169. DOI: 10.1109/DATE.2003.1253784.

[35] Michael Gelfond and Vladimir Lifschitz. "The stable model semantics for logic programming". In: *Fifth International Conference on Logic Programming*. MIT Press, 1988, pp. 1070–1080.

[36] Vladimir Lifschitz. "What is Answer Set Programming?" In: *National Conference on Artificial Intelligence*. Vol. 3. AAAI'08. Chicago, Illinois, July 2008, pp. 1594–1597. ISBN: 9781577353683.

[37] Timo Soininen and Ilkka Niemelä. "Developing a Declarative Rule Language for Applications in Product Configuration". In: *Practical Aspects of Declarative Languages*. Springer Berlin Heidelberg, 1998, pp. 305–319. ISBN: 978-3-540-49201-6.

[38] Alain Colmerauer and Philippe Roussel. "The Birth of Prolog". In: *History of Programming Languages—II*. Association for Computing Machinery, 1996, pp. 331–367. DOI: 10.1145/234286.1057820.

[39] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. "Answer Set Programming at a Glance". In: *Communications of the ACM* 54.12 (Dec. 2011), pp. 92–103. DOI: 10.1145/2043174.2043195.

[40] Vladimir Lifschitz. "Answer set programming and plan generation". In: *Artificial Intelligence* 138.1 (2002). Knowledge Representation and Logic Programming, pp. 39–54. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(02)00186-8.

[41] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. "Clingo = ASP + Control: Preliminary Report". In: *CoRR* abs/1405.3694 (2014).

[42] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. "Theory Solving Made Easy with Clingo 5". In: *32nd International Conference on Logic Programming (ICLP)*. Dagstuhl, Germany: Leibniz-Zentrum für Informatik, 2016, 2:1–2:15. DOI: 10.4230/OASIcs.ICLP.2016.2.

[43] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. "Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T)". In: *Journal of the ACM* 53.6 (Nov. 2006), pp. 937–977. DOI: 10.1145/1217856.1217859.

[44] Leonardo de Moura and Nikolaj Bjørner. "Model-based Theory Combination". In: *Electronic Notes in Theoretical Computer Science* 198.2 (2008), pp. 37–49. DOI: 10.1016/j.entcs.2008.04.079.

[45] Michael Bartholomew and Joohyung Lee. "Stable Models of Formulas with Intensional Functions". In: *International Conference on Principles of Knowledge Representation and Reasoning*. Rome, Italy: AAAI Press, 2012, pp. 2–12. ISBN: 9781577355601.

[46] Michael Bartholomew and Joohyung Lee. "Functional stable model semantics and Answer Set Programming Modulo Theories". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Dec. 2013, pp. 718–724. ISBN: 9781577356332.

[47] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane G. da Fonseca. "Performance assessment of multiobjective optimizers: an analysis and review". In: *IEEE Transactions on Evolutionary Computation* 7.2 (2003), pp. 117–132. DOI: 10.1109/TEVC.2003.810758.

[48] Tobias Blickle, Jürgen Teich, and Lothar Thiele. "System-Level Synthesis Using Evolutionary Algorithms". In: *Design Automation for Embedded Systems* 3.1 (Jan. 1998), pp. 23–58. DOI: 10.1023/A:1008899229802.

[49] Miqing Li, Shengxiang Yang, and Xiaohui Liu. "Diversity Comparison of Pareto Front Approximations in Many-Objective Optimization". In: *IEEE Transactions on Cybernetics* 44.12 (Apr. 2014), pp. 2568–2584. DOI: 10.1109/TCYB.2014.2310651.

[50] Joshua Knowles, L. Thiele, and Eckart Zitzler. "A tutorial on tbe performance assessment of stochastic multiobjective optimizers". In: *TIK Report* 216 (Feb. 2006).

[51] David A. Van Veldhuizen and Gary B. Lamont. "Evolutionary Computation and Convergence to a Pareto Front". In: *Late Breaking Papers at the Genetic Programming 1998 Conference*. July 1998, pp. 221–228.

[52] Oliver Schütze, Xavier Esquivel, Adriana Lara, and Carlos A. Coello Coello. "Using the Averaged Hausdorff Distance as a Performance Measure in Evolutionary Multiobjective Optimization". In: *IEEE Transactions on Evolutionary Computation* 16.4 (Feb. 2012), pp. 504–522. DOI: 10.1109/TEVC.2011.2161872.

[53] Kalyanmoy Deb and Sachin K. Jain. *Running performance metrics for evolutionary multi-objective optimizations*. Tech. rep. KanGAL Report No. 2002004. Kanpur, India: Kanpur Genetic Algorithms Laboratory (KanGAL), 2002.

[54] Ali Farhang-Mehr and Shapour Azarm. "An Information-Theoretic Entropy Metric for Assessing Multi-Objective Optimization Solution Set Quality ". In: *Journal of Mechanical Design* 125.4 (Jan. 2004), pp. 655–663. ISSN: 1050-0472. DOI: 10.1115/1.1623186.

[55] Eckart Zitzler and Lothar Thiele. "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". In: *IEEE transactions on Evolutionary Computation* 3.4 (Nov. 1999), pp. 257–271. DOI: 10.1109/4235.797969.

[56] Carlos M. Fonseca, Luís Paquete, and Manuel López-Ibáñez. "An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator". In: *IEEE International Conference on Evolutionary Computation*. Sept. 2006, pp. 1157–1163. DOI: 10.1109/CEC.2006.1688440.

[57] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. "Combining Convergence and Diversity in Evolutionary Multiobjective Optimization". In: *Evolutionary Computation* 10.3 (Sept. 2002), pp. 263–282. DOI: 10.1162/106365602760234108.

[58] Yacov Y. Haimes, Leon S. Lasdon, and David A. Wismer. "On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-1.3 (1971), pp. 296–297. DOI: 10.1109/TSMC.1971.4308298.

[59] Leonardo Lai, Lorenzo Fiaschi, and Marco Cococcioni. "Solving mixed Pareto-Lexicographic multi-objective optimization problems: The case of priority chains". In: *Swarm and Evolutionary Computation* 55 (2020), p. 100687. ISSN: 2210-6502. DOI: 10.1016/j.swevo.2020.100687.

[60] Saul Gass and Thomas Saaty. "The computational algorithm for the parametric objective function". In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 39–45. DOI: 10.1002/nav.3800020106.

[61] Kaisa Miettinen. "Introduction to Multiobjective Optimization: Noninteractive Approaches". In: *Multiobjective optimization: Interactive and evolutionary approaches*. Ed. by Jürgen Branke, Jurgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowiński. Springer-Verlag Berlin Heidelberg, 2008. Chap. Basics on Multiobjective Optimization, pp. 1–27. DOI: 10.1007/978-3-540-88908-3.

[62] Kevin I. Smith, Richard M. Everson, Jonathan E. Fieldsend, Chris Murphy, and Rashmi Misra. "Dominance-Based Multiobjective Simulated Annealing". In: *IEEE Transactions on Evolutionary Computation* 12.3 (May 2008), pp. 323–342. DOI: 10.1109/TEVC.2007.904345.

[63] John A. Nelder and Roger Mead. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (Jan. 1965), pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

[64] Manuel Lopez-Ibanez and Thomas Stutzle. "The Automatic Design of Multiobjective Ant Colony Optimization Algorithms". In: *IEEE Transactions on Evolutionary Computation* 16.6 (Feb. 2012), pp. 861–875. DOI: 10.1109/TEVC.2011.2182651.

[65] Carlos A. Coello Coello and Maximino Salazar Lechuga. "MOPSO: a proposal for multiple objective particle swarm optimization". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*. Vol. 2. May 2002, pp. 1051–1056. DOI: 10.1109/CEC.2002.1004388.

[66] Eckart Zitzler and Lothar Thiele. *An evolutionary algorithm for multiobjective optimization: The strength pareto approach*. Tech. rep. 43. Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), 1998. DOI: 10.3929/ethz-a-004288833.

[67] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. *SPEA2: Improving the strength Pareto evolutionary algorithm*. Tech. rep. 103. Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), 2001. DOI: 10.3929/ethz-a-004284029.

[68] N. Srinivas and Kalyanmoy Deb. "Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms". In: *Evolutionary Computation* 2.3 (1994), pp. 221–248. DOI: 10.1162/evco.1994.2.3.221.

[69] Kalyanmoy Deb, Amrit Pratap, Samir Agarwal, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[70] Kalyanmoy Deb and Himanshu Jain. "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints". In: *IEEE Transactions on Evolutionary Computation* 18.4 (2014), pp. 577–601. DOI: 10.1109/TEVC.2013.2281535.

[71] R. J. Dakin. "A tree-search algorithm for mixed integer programming problems". In: *The Computer Journal* 8.3 (Jan. 1965), pp. 250–255. DOI: 10.1093/comjnl/8.3.250.

[72] Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. "Cutting planes in integer and mixed integer programming". In: *Discrete Applied Mathematics* 123.1 (Nov. 2002), pp. 397–446. DOI: https://doi.org/10.1016/S0166-218X(01)00348-1.

[73] Martin Lukasiewycz, Michael Glass, Christian Haubelt, and Jürgen Teich. "Efficient symbolic multi-objective design space exploration". In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2008, pp. 691–696. DOI: 10.1109/ASPDAC.2008.4484040.

[74] Gerhard Brewka, James P. Delgrande, Javier Romero, and Torsten Schaub. "asprin: Customizing Answer Set Preferences without a Headache". In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 1467–1474. ISBN: 0262511290.

[75] Martin Lukasiewycz, Michael Glass, Christian Haubelt, and Jürgen Teich. "SAT-decoding in evolutionary algorithms for discrete constrained optimization problems". In: *IEEE Congress on Evolutionary Computation*. 2007, pp. 935–942. DOI: 10.1109/CEC.2007.4424570.

[76] Mario R. Barbacci, Gary E. Barnes, Roderic Geoffrey Galton. Cattell, and Daniel P. Siewiorek. *The ISPS computer description language : the symbolic manipulation of computer descriptions*. Tech. rep. Carnegie Mellon University, June 1978. DOI: 10.1184/R1/6610637.v1.

[77] Hugo J. De Man, Jan M. Rabaey, Jan Vanhoof, Gert Goossens, Paul Six, and Luc J. M. Claesen. "CATHEDRAL-II – A computer-aided synthesis system for digital signal processing VLSI systems". In: *Computer-Aided Engineering Journal* 5 (May 1988), pp. 55–66. DOI: 10.1049/cae.1988.0015.

[78] Giovanni De Micheli, David Ku, Frederic Mailhot, and Thomas Truong. "The Olympus Synthesis System". In: *IEEE Design & Test* 7.5 (Sept. 1990), pp. 37–53. DOI: 10.1109/54.60605.

[79] Rajesh K. Gupta and Giovanni De Micheli. "System-level synthesis using re-programmable components". In: *European Conference on Design Automation (EDAC)*. Mar. 1992, pp. 2–7. DOI: 10.1109/EDAC.1992.205881.

[80] Jean-Marc Daveau, Tarek Ben Ismail, and Amine Ahmed Jerraya. "Synthesis of system-level communication by an allocation-based approach". In: *Proceedings of the Eighth International Symposium on System Synthesis*. Sept. 1995, pp. 150–155. DOI: 10.1109/ISSS.1995.520627.

[81] Felice Balarin, Yosinori Watanabe, Harry Hsieh, Luciano Lavagno, Claudio Passerone, and Alberto Sangiovanni-Vincentelli. "Metropolis: an integrated electronic system design environment". In: *Computer* 36.4 (Apr. 2003), pp. 45–52. DOI: 10.1109/MC.2003.1193228.

[82] Jason Cong, Yiping Fan, Guoling Han, Wei Jiang, and Zhiru Zhang. "Platform-Based Behavior-Level and System-Level Synthesis". In: *2006 IEEE International SOC Conference*. Jan. 2006, pp. 199–202. DOI: 10.1109/SOCC.2006.283880.

[83] Rainer Dömer, Andreas Gerstlauer, Junyu Peng, Dongwan Shin, Lukai Cai, Haobo Yu, Samar Abdi, and Daniel D. Gajski. "System-on-Chip Environment: A SpecC-Based Framework for Heterogeneous MPSoC Design". In: *EURASIP Journal on Embedded Systems* 2008 (Jan. 2008). ISSN: 1687-3955. DOI: 10.1155/2008/647953.

[84] Tero Kangas, Petri Kukkala, Heikki Orsila, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen, Jouni Riihimäki, and Kimmo Kuusilinna. "UML-Based Multiprocessor SoC Design Framework". In: *ACM Transactions on Embedded Computing Systems* 5.2 (May 2006), pp. 281–320. DOI: 10.1145/1151074.1151077.

[85] Hristo Nikolov, Mark Thompson, Todor Stefanov, Andy Pimentel, Simon Polstra, R. Bose, Claudiu Zissulescu, and Ed Deprettere. "Daedalus: toward composable multimedia MP-SoC design". In: *Design Automation Conference (DAC)*. Jan. 2008, pp. 574–579. DOI: 10.1145/1391469.1391615.

[86] Joachim Keinert, Martin Streubūhr, Thomas Schlichter, Joachim Falk, Jens Gladigau, Christian Haubelt, Jūrgen Teich, and Michael Meredith. "SystemCoDesigner – an Automatic ESL Synthesis Approach by Design Space Exploration and Behavioral Synthesis for Streaming Applications". In: *ACM Transactions on Design Automation of Electronic Systems* 14.1 (Jan. 2009). DOI: 10.1145/1455229.1455230.

[87] Soonhoi Ha, Sungchan Kim, Choonseung Lee, Youngmin Yi, Seongnam Kwon, and Young-Pyo Joo. "PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems". In: *ACM Transactions on Design Automation of Electronic Systems* 12.3 (May 2008). DOI: 10.1145/1255456.1255461.

[88] Seongnam Kwon, Yongjoo Kim, Woo-Chul Jeun, Soonhoi Ha, and Yunheung Paek. "A Retargetable Parallel-Programming Framework for MPSoC". In: *ACM Transactions on Design Automation of Electronic Systems* 13.3 (July 2008). DOI: 10.1145/1367045.1367048.

[89] Martin Lukasiewycz, Felix Reimann, Fedor Smirnov, and Falko Hoefte. *OpenDSE - Open Design Space Exploration Framework*. URL: https://github.com/SDARG/opendse. June 2014. last accessed 2021-05-05.

[90] Mark Thompson, Hristo Nikolov, Todor Stefanov, Andy D. Pimentel, Cagkan Erbas, Simon Polstra, and Ed F. Deprettere. "A Framework for Rapid System-Level Exploration, Synthesis, and Programming of Multimedia MP-SoCs". In: *International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS)*. Salzburg, Austria, Sept. 2007, pp. 9–14. ISBN: 9781595938244. DOI: 10.1145/1289816.1289823.

[91] Andy D. Pimentel. "Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration". In: *IEEE Design Test* 34.1 (Feb. 2017), pp. 77–90. DOI: 10.1109/MDAT.2016.2626445.

[92] Tobias Schlichter, Martin Lukasiewycz, Christian Haubelt, and Jürgen Teich. "Improving system level design space exploration by incorporating SAT-solvers into multi-objective evolutionary algorithms". In: *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI)*. 2006. DOI: 10.1109/ISVLSI.2006.57.

[93] Mark Thompson and Andy D. Pimentel. "Exploiting Domain Knowledge in System-level MPSoC Design Space Exploration". In: *Journal of Systems Architecture* 59.7 (Aug. 2013), pp. 351–360. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2013.05.023.

[94] Fabrizio Ferrandi, Pier Luca Lanzi, Christian Pilato, Donatella Sciuto, and Antonino Tumeo. "Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.6 (2010), pp. 911–924. DOI: 10.1109/TCAD.2010.2048354.

[95] Hung-Yi Liu, Ilias Diakonikolas, Michele Petracca, and Luca Carloni. "Supervised design space exploration by compositional approximation of Pareto sets". In: *Design Automation Conference (DAC)*. June 2011, pp. 399–404. DOI: 10.1145/2024724.2024818.

[96] Nima Khalilzad, Kathrin Rosvall, and Ingo Sander. "A modular design space exploration framework for multiprocessor real-time systems". In: *Forum on Specification and Design Languages (FDL)*. Sept. 2016, pp. 1–7. DOI: 10.1109/FDL.2016.7880377.

[97] Martin Lukasiewycz, Martin Streubühr, Michael Glaß, Christian Haubelt, and Jürgen Teich. "Combined system synthesis and communication architecture exploration for MPSoCs". In: *Design, Automation and Test in Europe Conference (DATE)*. 2009, pp. 472–477. DOI: 10.1109/DATE.2009.5090711.

[98] Martin Lukasiewycz, Reinhard Schneider, Dip Goswami, and Samarjit Chakraborty. "Modular scheduling of distributed heterogeneous time-triggered automotive systems". In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2012, pp. 665–670. DOI: 10.1109/ASPDAC.2012.6165039.

[99] Nadathur Satish, Kaushik Ravindran, and Kurt Keutzer. "A Decomposition-based Constraint Optimization Approach for Statically Scheduling Task Graphs with Communication Delays to Multiprocessors". In: *Design, Automation and Test in Europe Conference (DATE)*. 2007, pp. 1–6. DOI: 10.1109/DATE.2007.364567.

[100] Felix Reimann, Michael Glaß, Christian Haubelt, Michael Eberl, and Jürgen Teich. "Improving platform-based system synthesis by satisfiability modulo theories solving". In: *International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS)*. 2010, pp. 135–144. DOI: 10.1145/1878961.1878986.

[101] Benjamin Andres, Alexander Biewer, Javier Romero, Christian Haubelt, and Torsten Schaub. "Improving Coordinated SMT-Based System Synthesis by Utilizing Domain-Specific Heuristics". In: *International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. 2015, pp. 55–68. DOI: 10.1007/978-3-319-23264-5_6.

[102] Alexander Biewer, Benjamin Andres, Jens Gladigau, Torsten Schaub, and Christian Haubelt. "A symbolic system synthesis approach for hard real-time systems based on coordinated SMT-solving". In: *Design, Automation and Test in Europe Conference (DATE)*. 2015, pp. 357–362. DOI: 10.7873/DATE.2015.0606.

[103] Felix Reimann, Martin Lukasiewycz, Michael Glass, Christian Haubelt, and Jürgen Teich. "Symbolic system synthesis in the presence of stringent real-time constraints". In: *Design Automation Conference (DAC)*. 2011, pp. 393–398. DOI: 10.1145/2024724.2024817.

[104] Roberto Sebastiani. "Lazy Satisfiability Modulo Theories". In: *Journal on Satisfiability Boolean Modeling and Computation* 3 (2007), pp. 141–224. DOI: 10.3233/SAT190034.

[105] Walter Habenicht. "Quad Trees, a Datastructure for Discrete Vector Optimization Problems". In: *Essays and Surveys on Multiple Criteria Decision Making: Proceedings of the Fifth International Conference on Multiple Criteria Decision Making*. Ed. by Pierre Hansen. Springer Berlin Heidelberg, Aug. 1983, pp. 136–145. DOI: 10.1007/978-3-642-46473-7_12.

[106] Sanaz Mostaghim and Jürgen Teich. "Quad-trees: A Data Structure for Storing Pareto Sets in Multiobjective Evolutionary Algorithms with Elitism". In: *Evolutionary Multi-objective Optimization: Theoretical Advances and Applications*. Ed. by Ajith Abraham, Lakhmi Jain, and Robert Goldberg. Springer London, 2005. Chap. Evolutionary Multiobjective Optimization, pp. 81–104. DOI: 10.1007/1-84628-137-7_5.

[107] M. Drozdík, Y. Akimoto, H. Aguirre, and K. Tanaka. "Computational Cost Reduction of Nondominated Sorting Using the M-Front". In: *IEEE Transactions on Evolutionary Computation* 19.5 (Oct. 2015), pp. 659–678. DOI: 10.1109/TEVC.2014.2366498.

[108] Andrzej Jaszkiewicz and Thibaut Lust. "ND-Tree: a Fast Online Algorithm for Updating a Pareto Archive and its Application in Many-objective Pareto Local Search". In: *arXiv preprint arXiv:1603.04798* (2016).

[109] Carlos A. Coello Coello, Clarisse Dhaenens, and Laetitia Jourdan. "Multi-objective combinatorial optimization: Problematic and context". In: *Advances in multi-objective nature inspired computing*. Springer, 2010, pp. 1–21. DOI: 10.1007/978-3-642-11218-8_1.

[110] Radu Marinescu. "Exploiting Problem Decomposition in Multi-objective Constraint Optimization". In: *Principles and Practice of Constraint Programming*. Springer Berlin Heidelberg, 2009, pp. 592–607. DOI: 10.1007/978-3-642-04244-7_47.

[111] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. "ReSPIR: A Response Surface Based Pareto Iterative Refinement for Application Specific Design Space Exploration". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.12 (2009), pp. 1816–1829. DOI: 10.1109/TCAD.2009.2028681.

[112] Cristina Silvano, William Fornaciari, Gianluca Palermo, Vittorio Zaccaria, Fabrizio Castro, Marcos Martinez, Sara Bocchio, Roberto Zafalon, Prabhat Avasare, Geert Vanmeerbeeck, Chantal Ykman-Couvreur, Maryse Wouters, Carlos Kavka, Luka Onesti, Alessandro Turco, Umberto Bondi, Giovanni Mariani, Hector Posadas, Eugenio Villar, Chris Wu, Fan Dongrui, Zhang Hao, and Tang Shibin. "MULTICUBE: Multi-objective Design Space Exploration of Multi-core Architectures". In: *IEEE Computer Society Annual Symposium on VLSI*. 2010, pp. 488–493. DOI: 10.1109/ISVLSI.2010.67.

[113] Sebastian Rojas-Gonzalez, Hamed Jalali, and Inneke Van Nieuwenhuyse. "A stochastic-kriging-based multiobjective simulation optimization algorithm". In: *Winter Simulation Conference (WCS)*. Dec. 2018, pp. 2155–2166. DOI: 10.1109/WSC.2018.8632322.

[114] Jianxia Zhang, Yizhong Ma, TahoYang, and Lijun Liu. "Estimation of the Pareto front in stochastic simulation through stochastic Kriging". In: *Simulation Modelling Practice and Theory* 79 (2017), pp. 69–86. DOI: 10.1016/j.simpat.2017.09.006.

[115] Gereon Onnebrink, Ahmed Hallawa, Rainer Leupers, Gerd Ascheid, and Awaid-Ud-Din Shaheen. "A Heuristic for Multi Objective Software Application Mappings on Heterogeneous MPSoCs". In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. Tokyo, Japan: ACM, 2019, pp. 609–614. DOI: 10.1145/3287624.3287651.

[116] Anirban Sengupta, Reza Sedaghat, and Zhipeng Zeng. "Rapid design space exploration by hybrid fuzzy search approach for optimal architecture determination of multi objective computing systems". In: *Microelectronics Reliability* 51.2 (2011), pp. 502–512. DOI: 10.1016/j.microrel.2010.08.003.

[117] Tobias Schwarzer, Joachim Falk, Simone Müller, Martin Letras, Christian Heidorn, Stefan Wildermann, and Jürgen Teich. "Compilation of Dataflow Applications for Multi-Cores Using Adaptive Multi-Objective Optimization". In: *ACM Transactions on Design Automation of Electronic Systems* 24.3 (Mar. 2019), 29:1–29:23. DOI: 10.1145/3310249.

[118] Miguel Angel Aguilar, Abhishek Aggarwal, Awaid Shaheen, Rainer Leupers, Gerd Ascheid, Jeronimo Castrillon, and Liam Fitzpatrick. "Work-in-progress: multi-grained performance estimation for MPSoC compilers". In: *International Conference on Compilers, Architectures and Synthesis For Embedded Systems*. Oct. 2017. DOI: 10.1145/3125501.3125521.

[119] Stefan Schürmans, Gereon Onnebrink, Rainer Leupers, Gerd Ascheid, and Xiaotao Chen. "Frequency-Aware ESL Power Estimation for ARM Cortex-A9 Using a Black Box Processor Model". In: *ACM Transactions on Embedded Computing Systems* 16.1 (Oct. 2016), 26:1–26:26. ISSN: 1539-9087. DOI: 10.1145/2987375.

[120] Sumit Mohanty, V. K. Prasanna, Sandeep Neema, and James Davis. "Rapid Design Space Exploration of Heterogeneous Embedded Systems Using Symbolic Search and Multi-granular Simulation". In: *Joint Conference on Languages, Compilers, and Tools for Embedded Systems & Software and Compilers for Embedded Systems*. Berlin, Germany, 2002, pp. 18–27. DOI: 10.1145/513829.513835.

[121] Zai Jian Jia, Antonio Núñez, Tomás Bautista, and Andy D. Pimentel. "A two-phase design space exploration strategy for system-level real-time application mapping onto MPSoC". In: *Microprocessors and Microsystems* 38.1 (2014), pp. 9–21. DOI: 10.1016/j.micpro.2013.10.005.

[122] Giuseppe Ascia, Vincenzo Catania, Alessandro G. Di Nuovo, Maurizio Palesi, and Davide Patti. "Efficient design space exploration for application specific systems-on-a-chip". In: *Journal of Systems Architecture* 53.10 (2007), pp. 733–750. DOI: 10.1016/j.sysarc.2007.01.004.

[123] Roberta Piscitelli and Andy D. Pimentel. "Design space pruning through hybrid analysis in system-level design space exploration". In: *Design, Automation and Test in Europe Conference (DATE)*. 2012, pp. 781–786. DOI: 10.1109/DATE.2012.6176600.

[124] Fernando Herrera and Ingo Sander. "Combining analytical and simulation-based design space exploration for time-critical systems". In: *Forum on Specification and Design Languages (FDL)*. 2013. DOI: 10.1007/978-3-319-06317-1_9.

[125] Amit Kumar Singh, Anup Das, and Akash Kumar. "RAPIDITAS: RAPId Design-Space-Exploration Incorporating Trace-Based Analysis and Simulation". In: *Euromicro Conference on Digital System Design*. IEEE Computer Society, 2013, pp. 836–843. DOI: 10.1109/DSD.2013.93.

[126] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. "Resource and Throughput Aware Execution Trace Analysis for Efficient Run-Time Mapping on MPSoCs". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.1 (2016), pp. 72–85. DOI: 10.1109/TCAD.2015.2446938.

[127] Robert P. Dick, David L. Rhodes, and Wayne Wolf. "TGFF: task graphs for free". In: *International Workshop on Hardware/Software Codesign*. Mar. 1998, pp. 97–101. DOI: 10.1109/HSC.1998.666245.

[128] Sander Stuijk, Marc C. W. Geilen, and Twan Basten. "SDF^3: SDF For Free". In: *Proceeding of 6th International Conference Application of Concurrency to System Design*. Turku, Finland: IEEE Computer Society Press, June 2006, pp. 276–278. DOI: 10.1109/ACSD.2006.23.

[129] Benjamin Andres, Martin Gebser, Torsten Schaub, Christian Haubelt, Felix Reimann, and Michael Glaß. "Symbolic System Synthesis Using Answer Set Programming". In: *International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. Ed. by Pedro Cabalar and Tran Cao Son. Springer Berlin Heidelberg, 2013, pp. 79–91. DOI: 10.1007/978-3-642-40564-8_9.

[130] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. 3rd Edition. MIT press, July 2009. ISBN: 978-0262033848.

[131] Richard Bellman. "On A Routing Problem". In: *Quarterly of Applied Mathematics* 16.1 (1958), pp. 87–90. DOI: 10.1090/qam/102435.

[132] Lester R. Ford. *Network Flow Theory*. Tech. rep. RAND Corporation, 1956.

[133] Jori Bomanson, Martin Gebser, Tomi Janhunen, Benjamin Kaufmann, and Torsten Schaub. "Answer set programming modulo acyclicity". In: *Fundamenta Informaticae* 147.1 (2016), pp. 63–91. DOI: 10.3233/FI-2016-1398.

[134] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. "Fully dynamic shortest paths in digraphs with arbitrary arc weights". In: *Journal of Algorithms* 49.1 (Oct. 2003), pp. 86–113. DOI: 10.1016/S0196-6774(03)00082-8.

[135] Scott Cotton and Oded Maler. "Fast and Flexible Difference Constraint Propagation for DPLL(T)". In: *International Conference on Theory and Applications of Satisfiability Testing*. Vol. 4121. Aug. 2006, pp. 170–183. DOI: 10.1007/11814948_19.

[136] Martin Gebser, Benjamin Kaufmann, Ramon Otero, Javier Romero, Torsten Schaub, and Philipp Wanko. "Domain-specific heuristics in answer set programming". In: *Conference on Artificial Intelligence* (Jan. 2013), pp. 350–356.

[137] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Javier Romero, and Torsten Schaub. "Progress in clasp Series 3". In: *International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. 2015, pp. 368–383. DOI: 10.1007/978-3-319-23264-5_31.

[138] Gene Myron Amdahl. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *American Federation of Information Processing Societies (AFIPS) Spring Joint Computer Conference*. Apr. 1967, pp. 483–485. DOI: 10.1145/1465482.1465560.

[139] Gustavo Patino Alvarez and Wang Jiang Chau. "Scenario-Aware Workload Characterization Based on a Max-Plus Linear Representation". In: *Formal Modeling and Analysis of Timed Systems*. Ed. by Martin Fränzle and Nicolas Markey. Cham: Springer International Publishing, Aug. 2016, pp. 177–194. DOI: 10.1007/978-3-319-44878-7_11.

Acronyms

ACO . . . . . Ant Colony Optimization
ADC . . . . . Analog to Digital Converter
AES . . . . . Advanced Encryption Standard
ALR . . . . . Arbitrary Length Routing
ALU . . . . . Arithmetic Logical Unit
ASIC . . . . . Application-specific Integrated Circuit
ASP . . . . . Answer Set Programming
ASPmT . . . . . ASP modulo Theories
CAD . . . . . Computer Aided Design
CDCL . . . . . Conflic-driven Clause Learning
CNF . . . . . Conjunctive Normal Form
CPI . . . . . Cycles per Instruction
CPU . . . . . Central Processing Unit
DCI . . . . . Diversity Comparison Indicator
DM . . . . . Decision Maker
DOR . . . . . Dimension Order Routing
DPLL . . . . . Davis-Putnam-Logemann-Loveland Algorithm
DSE . . . . . Design Space Exploration
DSP . . . . . Digital Signal Processor
EDA . . . . . Electronic Design Automation
ERR . . . . . Task Execution Time to Routing Delay Ratio
ESL . . . . . Electronic System Level
EUF . . . . . Equality of Uninterpreted Functions
FR . . . . . Filter Ratio
FSM . . . . . Finite State Machine
GD . . . . . Generational Distance

HDL . . . . . Hardware Description Language
HLS . . . . . High Level Synthesis
IDL . . . . . Integer Difference Logic
IIR . . . . . Infinite Impulse Response
ILP . . . . . Integer Linear Programming
IP . . . . . Intellectual Property
IQR . . . . . Inter Quantile Range
LP . . . . . Linear Programming
MOCOP . . . . . Multi-objective Combinational Problem
MOEA . . . . . Multi-objective Evolutionary Algorithm
MOOP . . . . . Multi-objective Optimization Problem
NoC . . . . . Network on Chip
NP . . . . . Non-deterministic Polynomial Time
NSGA . . . . . Non-dominated Sorting Genetic Algorithm
PSO . . . . . Particle Swarm Optimization
QF–IDL . . . . . Quantifier-free Integer Difference Logic
RAM . . . . . Random Access Memory
RTL . . . . . Register Transfer Level
SAT . . . . . Boolean Satisfiability Problem
SDFG . . . . . Synchronous Dataflow Graph
SLDL . . . . . System-level Design Language
SMT . . . . . Satisfiability modulo Theories
SoC . . . . . System-on-chip
SOOP . . . . . Single-objective Optimization Problem
SPEA . . . . . Strength Pareto Evolutionary Algorithm
SPG . . . . . Series Parallel Graph
SPR . . . . . Shortest Path Routing
TLM . . . . . Transaction-level Modeling
VHDL . . . . . Very High Speed Integrated Circuit Hardware Description Language
VLSI . . . . . Very-large-scale Integration
VSIDS . . . . . Variable State Independent Decaying Sum
WCET . . . . . Worst Case Execution Time

Declaration of Authorship

I hereby certify that the thesis I am submitting is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those regulations concerning disciplinary actions that may result from plagiarism. Any use of the works of any other author, in any form, is properly acknowledged at their point of use. The present work has not been submitted to any other examination committee in the same or similar form, neither abroad nor in Germany.

Rostock, 17.08.2021

M.Sc. Kai Neubauer