Linköping Studies in Science and Technology
Dissertation No. 2439

# Learning and Exploiting Subgoal Structures in Classical Planning

## Towards Reliable and Transparent Intelligent Agents that Learn to Plan on Multiple Levels

**Dominik Drexler**

LINKÖPING UNIVERSITY

Linköping Studies in Science and Technology
Dissertations, No. 2439

# Learning and Exploiting Subgoal Structures in Classical Planning

Towards Reliable and Transparent Intelligent Agents
that Learn to Plan on Multiple Levels

**Dominik Drexler**


Linköping University
Department of Computer and Information Science
Division of Artificial Intelligence and Integrated Computer Systems
SE-581 83 Linköping, Sweden

Linköping 2025

Typeset using LaTeX

Printed by LiU-Tryck, Linköping 2025

Edition 1:1


ISBN 978–91–8118–019–0 (print)
ISBN 978–91–8118–020–6 (PDF)
https://doi.org/10.3384/9789181180206
ISSN 0345-7524

# POPULÄRVETENSKAPLIG SAMMANFATTNING

Tänk tillbaka till första gången du satte dig bakom ratten på en bil. Dina händer greppade ratten, dina tankar rusade för att minnas varje trafikregel, och varje beslut kändes som en ödesdiger uppgift. Med tiden blev körningen en vana, enkel och intuitiv, även i hektisk trafik. Men när du ställs inför en oväntad omväg en komplicerad manöver måste du stanna upp och tänka efter, använda medvetet tänkande, för att navigera situationen. Denna skillnad mellan intuitivt och medvetet tänkande speglar två typer av intelligens som forskare försöker återskapa i artificiella system: system 1 och system 2.

Under det senaste decenniet har system 1-metoder gjort imponerande framsteg. Dessa system har slagit mänskliga mästare i spel som schack och Go – aktiviteter som kräver strategiskt tänkande och planering – och visat stor skicklighet i kooperativa flerspelarspel. System 1 bygger på erfarenhetsbaserat lärande, vilket gör det snabbt och intuitivt. Dock är beslutsprocessen inte transparent, vilket betyder att det är svårt att förstå eller förklara besluten. System 2, å andra sidan, resonerar steg för steg och analyserar varje möjlig handling för att uppnå ett mål. Detta gör beslutsprocessen transparent och tillförlitligt, men ofta för långsamt för realtidstillämpningar. Även om båda systemen är kraftfulla inom sina respektive områden, begränsar deras individuella svagheter att de används för mer komplexa, verkliga problem. Detta väcker en avgörande fråga inom artificiell intelligens: Hur kan vi kombinera system 1:s snabbhet och intuition med system 2:s pålitlighet och transparens för att skapa bättre och mer effektiva system?

Denna avhandling utforskar hur dessa två typer av intelligens kan integreras genom användning av fördefinierade målspråk. Till skillnad från många system 1-metoder, som hittar på vaga representationsspråk under inlärningen, har fördefinierade målspråk tydligt specificerad syntax och semantik. Syntaxen styr hur meningar byggs upp, medan semantiken bestämmer deras innebörd. Våra meningar beskriver delmål på högre nivå, vilket är avgörande för att lösa problem på ett effektivt, pålitligt och transparent sätt – en metod som även människor och många djur tillämpar. Vi planerar på en hög nivå först och löser detaljerna därefter. Forskare har föreslagit flera olika målspråk och jämfört deras uttryckskraft, det vill säga deras förmåga att beskriva objekten och deras relationer inom ett problem. I avhandlingen analyseras kraven för uttryckskraft hos olika målspråk, och resultaten visar att även enkla målspråk är tillräckliga för att möjliggöra en effektiv integration av system 1 och system 2. Detta demonstrerar att det är fullt genomförbart att kombinera snabbhet och intuition med transparens och pålitlighet genom målspråk.

ABSTRACT

Classical planning aims to find a plan that is a sequence of actions allowing an intelligent agent to move from its current situation to one that satisfies the goal. Finding a plan is computationally challenging. Agents in the real world often encounter structurally similar problems with varying objects but the same predicates, actions, and related goals. Generalized planning aims to find a general plan that compactly encodes efficiently obtainable plans for each problem in an infinitely large class of structurally similar problems. Thus, we can query a general plan to efficiently obtain a plan for any problem in the class. A general plan may encode behavior on different levels of abstraction. High-level abstractions include subgoal structures that encode stepping stones towards the goal. Subgoal structures play a central role in human problem-solving, enabling reasoning at a higher level before working out the details of a plan. Learning simple, compact, meaningful, and efficient subgoal structures and their hierarchies without human intervention is an open challenge.

This thesis introduces a method for learning subgoal structures with a crisp characterization; they decompose problems into subproblems of controllable polynomial complexity. We represent subgoal structures using the recently introduced policy sketches language, whose simple syntax and semantics build the theoretical foundation of our work. We extend our method to address the long-standing problem of learning hierarchical policies. Our extended method iteratively decomposes classes of problems into classes of subproblems with strictly smaller polynomial complexity, resulting in effective hierarchical decompositions. Our methods learn from small example problems using combinatorial optimization. They seek the syntactically simplest solution, enabling interpretability and allowing us often to establish their correctness for an entire problem class. When learning methods fail, it often results from limited scalability or a lack of language expressivity. We develop two methods to address these limitations. First, we develop symmetry-based abstractions to reduce redundancy in training data and improve learning efficiency. Second, we develop a method for testing the language expressivity requirement of benchmark sets using first-order logic. Moreover, we take steps toward developing a scalable planning framework that avoids an exponential preprocessing step known as grounding, which is often unnecessary in generalized planning. Our framework supports expressive language features such as conditional effects and derived predicates that cannot concisely be compiled away, enabling researchers to model and address more complex planning problems.

# Acknowledgments

Writing my thesis and the time spent during my doctoral studies have been an incredible journey. It gave me the chance to further explore my deepest passions and the opportunity of meeting amazing people worldwide. I want to use this space to express my gratitude to those who have contributed to this journey in one way or another.

First and foremost, I want to thank my advisor, Hector Geffner, for giving me the opportunity to pursue this path, making it possible to travel to international conferences, inviting me as a guest researcher to his group at RWTH Aachen University in Germany, giving me a lot of freedom in my work, providing many ideas, and working in close collaboration. Hector's vision of artificial intelligence and strict requirement for deep understanding and crisp solutions have truly inspired my research.

More often than not, a doctoral dissertation results from close collaboration with researchers worldwide. The success of this thesis and other publications has been made possible through the contributions of several co-authors: Blai Bonet, Hector Geffner, Daniel Gnad, Paul Höft, Jendrik Seipp, Javier Segovia-Aguas, David Speck, and Simon Ståhlberg. I am deeply grateful to all of you for your collaboration and insights.

Working in the Machine Reasoning Lab has been an incredible experience, surrounded by many wonderful people, including Karin Baardsen, Markus Fritzsche, Martin Funkquist, Hector Geffner, Elliot Gestrin, Daniel Gnad, Arash Haratian, Paul Höft, Oliver Joergensen, Kristina Levina, Farid Musayev, Ulf Nilsson, Mauricio Salerno, Jendrik Seipp, Mika Skjelnes, David Speck, Simon Ståhlberg, and Damien Van Meerbeeck. I would like to thank Jendrik Seipp, my co-advisor, for introducing me to this fantastic opportunity and providing invaluable guidance. Since the Machine Reasoning Lab was founded relatively recently, I had the unique experience of being its first and only student. Thankfully, the wonderful people from the Natural Language Processing Lab welcomed me for Fika and have continued to do so ever since. Thank you for being so inviting: Marcel Bollmann, Ehsan Doostmohammadi, Kevin Glocker,

Oskar Holmström, Jenny Kunz, Marco Kuhlmann, Kätriin Kukk, Noah-Manuel Michael, Romina Oji, and Olle Torstensson.

During my Bachelor's and Master's studies at the University of Freiburg, I met two amazing people whose passion first sparked my interest in artificial intelligence. First, Robert Mattmüller, an exceptional teacher who supervised several of my projects during my Bachelor's and Master's studies, including my theses. His guidance played a crucial role in shaping my academic journey. Second, David Speck, my co-advisor during my Masters, generously shared his office with me during that time. Ever since then, David has provided invaluable advice, both professionally and personally, and has become a truly great friend. I would also like to express my special thanks to Simon Ståhlberg, whom I first met at Linköping University. Together, we had countless fruitful discussions that led to impactful work, new ideas, and a shared passion for software development. Our conference visits were always paired with incredible travels, from a relaxed barbecue at a villa in Rhodes, Greece, to cruise shipping through Lan Ha Bay, Vietnam. Through all these experiences, Simon has become a true friend.

One of the reasons it felt easy to move such a long distance from Southern Germany to Sweden was my interest in powerlifting. In this regard, I would like to thank everyone at Linköpings Atletklubb for introducing me to the sport and creating such a great atmosphere in the gym. I am also grateful to the people at Kraftsport Aachen e.V., who welcomed me as a member during my research visit to Aachen. Your dedication has been truly inspiring, and the sport has played a significant role in my doctoral studies by keeping me healthy and balanced. Speaking about moving to Sweden, I would also like to thank my wonderful landlords Kristina and Bo Lennhammar for their support.

Doctoral studies can be intense and demanding, but they have also been an incredibly rewarding experience, thanks in no small part to the support of great friends. I would like to thank Oliver Blessing, Konstantin Jordan, Fabian Recktenwald, Natthakorn Saensaeng, and An Tran, who have been by my side for quite some time. They always had my back and provided a much-needed dose of distraction when I needed it most.

Last but not least, I would like to thank my family, my parents, Manfred and Monika Drexler, my sister Natalie Kotz and her husband, Daniel Kotz, my aunts Cornelia Faber, Eva Gach, Martina Stelter, my uncles Remko Faber, Robert Gach, Eberhard Roth, and Uwe Stelter, as well as my cousins Katharina Faber, Robert Faber, Elli Gach, and Lotta Gach. Your unwavering support, encouragement, and belief in me have been the foundation that made this journey possible. I am forever grateful to have such a wonderful family by my side.

# Contents

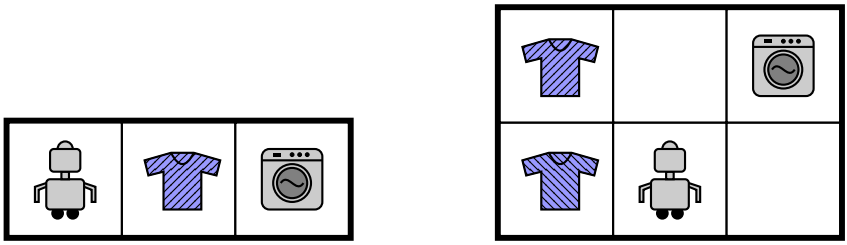
# List of Figures

# List of Tables

# 1 Introduction

Classical planning aims to find a plan that is a sequence of actions allowing an intelligent agent to move from its current situation to one that satisfies the goal. Finding a plan is computationally challenging as plan existence is already PSPACE-complete (Bylander 1994). Heuristic search (Hart et al. 1968) is one of the most effective methods for addressing classical planning problems, utilizing heuristics as goal distance estimators to guide the search toward the goal state. These heuristics derive from problem simplification, which may result in exponential worst-case search behavior (Bonet and Geffner 2001).

In the real world, agents often face structurally similar problems with varying objects but the same predicates used to describe the world, actions to act in the world, and related goals. For example, Figure 1.1 illustrates two problems in a discretized household environment where the goal is to do the laundry. Each problem has a robot with limited actions, such as moving between cells, grasping, and releasing objects. A plan for the first problem in Figure 1.1a is to move right, pick the laundry piece, move right, and drop the piece in the machine, while the second problem in Figure 1.1b requires to move both laundry pieces to the machine. A generalized planning problem considers a class of structurally similar problems (Jiménez et al. 2019). Unlike classical planning, which aims to find a plan for a single problem, generalized planning aims to find a general plan that compactly encodes efficiently obtainable plans for all problems of a class. Hence, compact general plans only exist for tractable



**(a)** First problem. **(b)** Second problem.

**Figure 1.1:** The illustration shows two discretized problems from the class of problems for doing the laundry. Each problem has a robot, a washing machine, a varying-sized grid, and various laundry pieces.

classes and exclude, for example, NP-hard classes unless P = NP (Garey and Johnson 1979). Unlike heuristics, general plans are often learned from small examples (e.g., Francès, Bonet, et al. 2021; Ståhlberg, Bonet, et al. 2023). For example, a general plan in natural language for the class of problems for doing the laundry might consist of the following steps: moving the robot to a laundry piece, picking it up, moving to the washing machine, placing the piece in the machine, repeating this process until all laundry pieces are in the machine, stopping if the machine is full, followed by launching the washing program.

Natural language is not ideal for representing general plans in computers because it is highly expressive and allows for ambiguities. Target languages are more suitable and come with well-defined syntax and semantics. Prominent examples from the literature are general policies (Francès, Bonet, et al. 2021), programs (Segovia-Aguas et al. 2019), linear temporal logics (LTL) (Bacchus and Kabanza 2000), reward machines (Icarte et al. 2022), and graph neural networks (Ståhlberg, Bonet, et al. 2022a). Deep learning and neural networks are among the most scalable approaches to artificial intelligence, demonstrated through numerous remarkable successes in the past decade. However, neural networks are difficult or impossible to understand. Formal target languages such as general policies, programs, LTL, or reward machines offer an interpretable but less scalable framework for automated learning.

Subgoal structures represent an important family of general plans, focusing on intermediate goals that an agent must achieve to progress toward its overall goal. Unlike general policies, which encode low-level behavior in terms of immediate actions, subgoals provide flexibility by allowing the behavior needed to achieve them to be tailored towards the specific environment (Zheng et al. 2020). For example, in Figure 1.1b, a subgoal for doing the laundry may be to place a single piece into the washing machine. It requires the robot to execute several actions that depend on its current location.

This thesis introduces a method that learns subgoal structures over the formal policy sketches language (Bonet and Geffner 2021; Bonet and Geffner 2024). Policy sketches define subgoals by constraining qualitative feature changes. Unlike LTL formulas in planning (Bacchus and Kabanza 2000), intrinsic reward functions in deep reinforcement learning (Zheng et al. 2020) or symbolic reward machines in reinforcement learning (Icarte et al. 2022), our method learns to split planning problems into subproblems with *strictly polynomial complexity* characterized by the notion of width (Lipovetzky and Geffner 2012). For example, the single sketch rule $\{u > 0\} \mapsto \{u\downarrow\}$ says that decreasing the number of laundry pieces that are not yet in the machine ($u\downarrow$) is good. Our method for learning policy sketches employs combinatorial optimization to find the simplest solution, measured by its syntactic complexity. Our syntactically optimized solutions are human-interpretable, enabling us to often manually show their correctness for an entire target class of problems.

While subgoal structures are reliable in solving planning problems, they often do not represent plans on multiple levels of abstraction. However, humans efficiently solve complex real-world problems by planning on multiple levels of abstraction (LeCun 2022). A long-standing open research question in artificial intelligence is how to learn an effective hierarchical representation of plans without supervision. Despite decades of interest in hierarchical planning, whether in model-based approaches (Sacerdoti 1974; Tate 1977; Erol et al. 1994) or model-free reinforcement learning (Parr and Russell 1997; Dietterich 2000; Barto and Mahadevan 2003), the problem of learning effective hierarchical structures without supervision remains an open challenge. A central issue is the absence of *precise characterizations for describing and uncovering effective hierarchical structures* (Drexler, Seipp, and Geffner 2023). In this thesis, we shed light on this problem by using policy sketches to iteratively split classes of problems into subclasses of problems whose *polynomial complexity strictly decreases* characterized by the width. For instance, placing all laundry pieces in the machine is a problem with unbounded complexity. Decomposing it into subproblems, such as placing one piece at a time, reduces the width to two.

When our learning methods fail, it typically results from limited scalability or a lack of language expressivity. We develop two methods to address these limitations. First, we develop equivalence-based abstractions based on state symmetries to reduce redundancy in training sets and improve learning efficiency. Second, we develop a method for testing the expressivity requirements of benchmark sets using first-order logic. Our method does not give strong guarantees for learning compact general plans for our training sets. However, our findings often show empirical alignment with previous work on learning general policies with graph neural networks (Ståhlberg, Bonet, et al. 2023). A general result is that a simple formal language called $k$-variable first-order logics with counting quantifiers restricted to at most three variables ($\mathrm{C}_3$) is

sufficient for all our considered benchmark sets, indicating that manageable expressive power often suffices.

This thesis also makes steps in building a unified framework for generalized planning by extending the Mimir planning library (Ståhlberg 2023). One of the key strengths of generalized planning is that general plans are highly informative. Hence, an exponential preprocessing step called grounding (Helmert 2009) that aims at improving runtime efficiency but limits the size of manageable problems is often unnecessary. Instead, our library works directly on the first-order problem representation, allowing general plans to be executed on problem sizes out of reach for search methods that require the ground problem representation (Drexler, Seipp, and Geffner 2024). Our novel contribution is the support for expressive language features, including conditional effects and derived predicates, that cannot be concisely compiled away (Nebel 2000; Thiébaux et al. 2005). Our experimental evaluation shows that our library is competitive with state-of-the-art systems.

Our findings contribute to the practical and theoretical foundation of generalized planning by providing characterizations and methods for learning subgoal structures, abstractions based on state symmetries for more efficient learning, tools to explain failures of learning general plans, and an expressive planning library focusing on generalized planning.

## 1.1 Outline

This thesis is structured as follows. In Chapter 2, we define the preliminaries of the thesis. In particular, we define classical planning, generalized planning, planning width, policy sketches, description logics, relational structures, and graphs. In Chapter 3, we analyze through examples whether policy sketches is a good language for capturing the subgoal structure in generalized planning. We conclude the chapter with an empirical evaluation, showing the computational value of policy sketches of bounded width on several planning domains. In Chapter 4, we present a combinatorial method for learning policy sketches of bounded width from small examples without supervision. In Chapter 5, we present a combinatorial method for learning hierarchical policies based on the method for learning sketches. In Chapter 6, we present a general method for generating equivalence-based abstractions to reduce redundancy in training sets. In Chapter 7, we present a method for testing the language expressivity requirements of benchmark sets. In Chapter 8, we present a planning library tailored towards generalized planning supporting expressive language features and an empirical evaluation against state-of-the-art systems. Chapter 9 summarizes and concludes this thesis.

## 1.2 Published Works

We published the core results at leading AI, AI planning, and knowledge representation conferences. The following publications form the foundation of this thesis. At the start of each chapter, we highlight the relevant publications that form its foundation. Each core paper is attached in its publication form at the end of the thesis.

- Dominik Drexler, Jendrik Seipp, and Hector Geffner (2024). "Expressing and Exploiting Subgoal Structure in Classical Planning Using Sketches". In: *Journal of Artificial Intelligence Research* 80, pp. 171–208.
  - Dominik Drexler, Jendrik Seipp, and Hector Geffner (2021). "Expressing and Exploiting the Common Subgoal Structure of Classical Planning Domains Using Sketches". In: *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*. Ed. by Esra Erdem, Meghyn Bienvenu, and Gerhard Lakemeyer. IJCAI Organization, pp. 258–268. **(superseded)**
- Dominik Drexler, Jendrik Seipp, and Hector Geffner (2022). "Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width". In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*. Ed. by Sylvie Thiébaux and William Yeoh. AAAI Press, pp. 62–70.
- Dominik Drexler, Jendrik Seipp, and Hector Geffner (2023). "Learning Hierarchical Policies by Iteratively Reducing the Width of Sketch Rules". In: *Proceedings of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*. Ed. by Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner. IJCAI Organization, pp. 208–218.
- Dominik Drexler, Simon Ståhlberg, Blai Bonet, and Hector Geffner (2024b). "Symmetries and Expressive Requirements for Learning General Policies". In: *Proceedings of the Twenty-First International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*. IJCAI Organization.
  - Dominik Drexler, Simon Ståhlberg, Blai Bonet, and Hector Geffner (2024a). "Equivalence-Based Abstractions for Learning General Policies". In: *ICAPS 2024 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*. **(superseded)**

The following publications also resulted from my doctoral research but are not central to this thesis. However, we will discuss some of their ideas and concepts in this thesis.

- Blai Bonet, Dominik Drexler, and Hector Geffner (2024). "On Policy Reuse: An Expressive Language for Representing and Executing General Policies that Call Other Policies". In: *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*. Ed. by Sara Bernardini and Christian Muise. AAAI Press, pp. 31–39.
  - Blai Bonet, Dominik Drexler, and Hector Geffner (2023). "General and Reusable Indexical Policies and Sketches". In: *NeurIPS 2023 Workshop on Generalization in Planning*. **(superseded)**
- Dominik Drexler, Javier Segovia-Aguas, and Jendrik Seipp (2022). "Learning General Policies and Helpful Action Classifiers from Partial State Spaces". In: *IJCAI 2022 Workshop on Generalization in Planning*.
- Dominik Drexler and Jendrik Seipp (2023). "DLPlan: Description Logics State Features for Planning". In: *ICAPS 2023 System Demonstrations and Exhibits*.
- Dominik Drexler, Daniel Gnad, Paul Höft, Jendrik Seipp, David Speck, and Simon Ståhlberg (2023). "Ragnarok". In: *Tenth International Planning Competition (IPC-10): Planner Abstracts*.
- Dominik Drexler, Jendrik Seipp, and David Speck (2023). "Odin: A Planner Based on Saturated Transition Cost Partitioning". In: *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

# 2 Preliminaries

This chapter defines classical planning, generalized planning, planning width, policy sketches, description logics, relational structures, and graphs. In the following, if not explicitly mentioned, we assume sets to be finite.

## 2.1 Classical Planning

Classical planning is the problem of finding an action sequence that allows an intelligent agent to move from its current situation to one that satisfies a given goal. The central assumptions are a fully observable environment and deterministic actions, where each action produces a predictable outcome. A planning domain defines predicate symbols for describing the environment and action schemas for describing interactions with the environment.

**Definition 1 (Planning Domain).** A (first-order) planning domain or simply domain is a pair $\mathcal{D} = \langle \mathcal{R}, \mathcal{A} \rangle$ where:

- $\mathcal{R}$ is a set of predicates (or relations) of the form $p/k$, where $p$ is the name, and $k$ is the arity, and
- $\mathcal{A}$ is a set of deterministic first-order action schemas.

**Example 1.** The laundry domain consists of the following predicate symbols and action schemas.[1] The predicate symbols are $robot/1$, $piece/1$, and $loc/1$ to describe the type of an object, $at/2$ to describe the location of a laundry piece, the robot, $carry/2$ to describe the laundry piece being carried by the robot, and $holding/1$ to describe whether the robot is holding a laundry piece, $adj/2$ to describe that two locations are adjacent. The action schemas are $pick(a, p, l)$ allowing the robot $a$ to pick up $p$ at $l$ when its hand is empty and both are at location $l$, $drop(a, p, l)$ for the inverse operation, and $move(a, l, l')$ to move the robot $a$ between adjacent locations $l$ and $l'$.

In a planning domain, predicates describe relationships among variables, and action schemas describe how the relationships among variables may change deterministically. **Grounding** replaces each variable in a predicate or action schema with an object. Grounding a predicate produces a **ground atom** describing the relationship between the objects. A **state** $s$ is a set of ground atoms where all ground atoms not in $s$ are assumed to be false. Similarly, grounding an action schema produces a **ground action** describing how object relationships change. We do not explicitly restrict the action schema structure but require that we can detect whether a ground action is **applicable** in a state. Moreover, we require that we can compute the unique **successor state** $s'$ when applying an applicable ground action $a$ in a state $s$. A state pair $(s, s')$ is in the relation $Succ$ if a ground action $a$ is applicable in $s$ and produces the successor state $s'$. A **trajectory** from a state $s_1$ to $s_n$ is a state sequence $s_1, s_2, \ldots, s_n$ such that $(s_i, s_{i+1})$ is in $Succ$, for all $1 \leq i < n$. A planning problem over a planning domain defines a set of objects, the initial and goal situations, and the objective of finding a plan, which is a goal-achieving trajectory.

**Definition 2 (Planning Problem).** A planning problem or simply problem over a planning domain is a pair $P = \langle \mathcal{D}, \mathcal{I} \rangle$ where $\mathcal{D}$ is the *planning domain* and $\mathcal{I} = \langle O, s_0, \gamma \rangle$ is problem-specific information consisting of

- $O$ is the set of objects,
- $s_0$ is the initial state describing the current situation,
- $\gamma$ is a set of ground atoms describing a goal situation, which induces a set of goal states $G = \{s \in S \mid s \supseteq \gamma\}$ where $S$ is the set of all states,

where the objective is to find a **plan** for $s_0$, i.e., a trajectory from $s_0$ to a state $s$ in $G$. A plan for $s$ with minimal length is **optimal** with length denoted by $V^*(s)$. A state $s$ is **solvable** if there is a plan for $s$, and otherwise, **unsolvable**. A state $s$ is **reachable** in $P$ if there is a trajectory from $s_0$ to $s$. We write $P[s, G]$ for a problem like $P$ but with initial state $s$ and *arbitrary* goal states $G$.

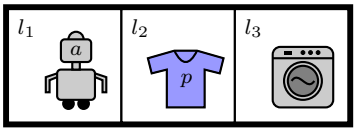



**Figure 2.1:** The illustration shows a problem over the laundry domain. A robot, a laundry piece, and a washing machine are in a grid with three locations.

**Example 2.** Figure 2.1 (same as Figure 1.1a) shows the initial state $s_0$ of a problem $P$ over the laundry domain. The set of objects consists of three locations $l_1, l_2$ and $l_3$, from left to right, a laundry piece $p$, and the robot $a$. The robot $a$ is at location $l_1$, i.e., $at(a, l_1)$ is true. The laundry piece is at location $l_2$. The washing machine is not an explicit object but acts as a marker for the goal location of the laundry pieces, i.e., $at(p, l_3)$ is in the goal $\gamma$ of $P$. We say that a laundry piece is delivered if and only if it is at the goal location; otherwise, we say it is undelivered. The action sequence $\langle move(a, l_1, l_2),\ pick(a, p, l_2),\ move(a, l_2, l_3),\ drop(a, p, l_3)\rangle$ achieves the goal, i.e., when sequentially applied from $s_0$, produces an optimal plan for $s_0$.

Notice that we define a plan for a state as a trajectory, i.e., state sequence, to a goal state and not as an action sequence. However, given a trajectory, we can easily reconstruct an action sequence with identical length. A problem induces a state space that fully captures the dynamics of a problem but where the actions are compiled away because we will never explicitly refer to them. We will use state spaces to learn solutions for a generalized planning problem.

**Definition 3 (Induced State Space).** The state space induced by a problem $P$ is a tuple $S_P = \langle S, s_0, G, Succ\rangle$ where $S$ is the set of all states, $s_0$ is the initial state, $G$ are the goal states of $P$, and $Succ$ is the successor relation.

## 2.2 Generalized Planning

Generalized planning extends classical planning to address classes of structurally similar problems within a common planning domain and requires scalable solution algorithms for any problem in the class (Jiménez et al. 2019).

**Definition 4 (Generalized Planning Problem).** A generalized planning problem is a possibly infinite set $\mathcal{Q}$ of problems over a planning domain $\mathcal{D}$. The objective is to find a **general plan** for $\mathcal{Q}$, i.e., an algorithm that solves any problem $P$ in $\mathcal{Q}$ in time that is polynomial in a reasonable representation size of $P$.

[1]The domain is like the Delivery domain (Bonet and Geffner 2021).

There is no general plan for classes of problems $\mathcal{Q}$ that are computationally intractable, e.g., NP-hard, unless P = NP, such as the traveling salesperson problem class (Garey and Johnson 1979). While avoidable, we assume for simplicity that $\mathcal{Q}$ is *closed*, i.e., if $P$ is in $\mathcal{Q}$, then all problems that are like $P$ but with initial state $s$ that are solvable and reachable in $P$ are in $\mathcal{Q}$, enforcing a general plan to solve those states $s$ too (Drexler, Seipp, and Geffner 2022).

**Example 3.** The class of problems over the laundry domain $\mathcal{Q}_L$ consists of infinitely many problems over the laundry domain from Example 1 restricted to a single robot with the common objective of moving all laundry pieces to the location of the washing machine. The class $\mathcal{Q}_L$ includes the problem in Example 2. A general plan for $\mathcal{Q}_L$ might outline a sequence of actions: moving towards an undelivered laundry piece, picking it up, moving to the machine, dropping it, and repeating until all pieces are in the machine.

## 2.3 Planning Width

The width (Lipovetzky and Geffner 2012) of a problem is an integer $k$ that measures the difficulty of finding an optimal plan. The iterative width algorithm (IW) can efficiently find a plan for problems with small width.

**Definition 5 (Width).** The width $w(P)$ of a problem $P$ with initial state $s_0$ and arbitrary goal states $G$ is the minimum $k$ for which there exists a sequence $t_0, t_1, \ldots, t_m$ of tuples $t_i$ each with at most $k$ ground atoms from $P$, such that:

1. $P[s_0, G_{t_0}]$ has an optimal plan of length $0$, i.e. $t_0$ holds in $s_0$,
2. any optimal plan for $P[s_0, G_{t_i}]$ can be extended into an optimal plan for $P[s_0, G_{t_{i+1}}]$ by adding a single action, for all $1 \leq i < m$, and
3. any optimal plan for $P[s_0, G_{t_m}]$ is an optimal plan for $P[s_0, G]$,

where, each tuple $t_i$ induces (sub-)goal states $G_{t_i} = \{s \in S \mid s \supseteq set(t_i)\}$ and $set(t_i)$ denotes the interpretation of $t_i$ as a set of ground atoms.

If $P$ has an optimal plan of length at most $1$, we set $w(P) = 0$. The width $w(\mathcal{Q})$ of a class of problem $\mathcal{Q}$ is the maximum width $w(P)$ of any problem $P$ in $\mathcal{Q}$. If the width of a problem $P$ is $w(P) = k$, then the iterative width algorithm IW($k$) finds an optimal plan for $P$ in time that is exponential in $k$. The IW($k$) algorithm is a breadth-first search that prunes a newly generated state if it does not make a set of at most $k$ state atoms true for the first time. The IW algorithm runs IW($k$) in sequence for $k = 0, 1, 2, \ldots$ until the problem is solved or found to be unsolvable. The effective width is the smallest $k$ for which IW($k$) solves $P$ and a plan may be suboptimal if $k$ is less than $w(P)$.

**Example 4.** The width of a problem over the laundry domain with $n$ laundry pieces is unbounded. The width of a problem with a laundry single piece is 2. Since the width is 2, IW efficiently finds a plan (Bonet and Geffner 2021).

## 2.4 Policy Sketches

Policy sketches is a language for temporal abstraction in classical planning (Bonet and Geffner 2021). The policy sketches language uses the same syntax as the language of general policies but with more general semantics (Francès, Bonet, et al. 2021). General policies aim to specify immediate single step subgoal states while policy sketches specify subgoal states possibly further away. A key component of policy sketches are features to define state abstractions.

A **feature** $f$ is a state function. There are two types of features. A Boolean feature $p$ maps a state $s$ in $P$ in $\mathcal{Q}$ into the Boolean domain, and a numerical feature $n$ maps a state into the non-negative integers.

A **Boolean feature condition** is an expression of the form $p$ or $\neg p$ for a Boolean feature $p$, and $n=0$ or $n>0$ for a numerical feature $n$. A state $s$ in $P$ in $\mathcal{Q}$ satisfies the condition $p$ (resp. $\neg p$) iff $p$ (resp. $\neg p$) is true in $s$, and the condition $n=0$ (resp. $n>0$) iff $n(s)=0$ (resp. $n(s)>0$).

A **Boolean feature effect** is an expression of the form $p$, $\neg p$, or $p?$ for a Boolean feature $p$, and $n\uparrow$, $n\downarrow$, or $n?$ for a numerical feature $n$. A state pair $[s,s']$ in $P$ in $\mathcal{Q}$ satisfies the effect $p$ (resp. $\neg p$) iff $p$ (resp. $\neg p$) is true in $s'$, and the effect $n\uparrow$ (resp. $n\downarrow$) iff $n(s)<n(s')$ (resp. $n(s)>n(s')$). The state pair $[s,s']$ always satisfies the effects $p?$ and $n?$.

A **sketch rule** $r_\Phi$ over features $\Phi$ has form $C \mapsto E$ where $C$ is a set of Boolean feature conditions and $E$ is a set of Boolean feature effects. We write $r$ for $r_\Phi$ if the set of features is clear from the context. A state pair $[s,s']$ is **compatible** with a sketch rule $r_\Phi = C \mapsto E$ iff $s$ satisfies all conditions in $C$, $[s,s']$ satisfies all effects in $E$, and all features $f$ in $\Phi$ that do not occur in $E$ must stay the same, i.e., $p(s)=p(s')$ and $n(s)=n(s')$. The set of **subgoal states** $G_{r_\Phi}(s)$ of a sketch rule $r$ is the set of states $\{s' \mid [s,s']$ is compatible with $r_\Phi\}$, and is extended to include the induced goal states $G_\gamma$ of a given problem $P$.

**Example 5.** Consider the laundry domain $\mathcal{Q}_L$ from Example 3 and the set of features $\Phi=\{u\}$, where $u$ is the number of undelivered laundry pieces. The sketch rule $r=\{u>0\} \mapsto \{u\downarrow\}$ over $\Phi$ specifies that the subgoal states $G_r(s)$ for a state $s$ in a problem $P$ contains the goal states $G$ of $P$ and the subgoal states where the number of undelivered laundry pieces decreases ($u\downarrow$). For simplicity, we describe such subgoal states by saying that delivering a laundry piece is "good". We will use similar intuitive phrasings throughout the thesis.

**Definition 6 (Policy sketch).** A policy sketch (or sketch) $R_\Phi$ over a set of features $\Phi$ for a class of problems $\mathcal{Q}$ is a set of sketch rules over $\Phi$. The set of **subgoal states** $G_{R_\Phi}(s)$ of a sketch $R_\Phi$ is the set of states $\cup_{r \in R_\Phi} G_r(s)$. We write $R$ for a sketch $R_\Phi$ if the set of features is clear from the context.

**Example 6.** We extend the Example 5 by adding another feature $H$ to $\Phi$ that is true if and only if the robot holds a laundry piece. The sketch $R$ consists of two rules $r_1$ and $r_2$ over $\Phi$ where rule $r_1 = \{u > 0, H\} \mapsto \{u\downarrow, \neg H\}$ and rule $r_2 = \{u > 0, \neg H\} \mapsto \{H\}$. Rule $r_1$ says that delivering a laundry piece to the goal location is good ($u\downarrow$) when holding one ($H$), and rule $r_2$ says that getting hold of an undelivered laundry piece is good ($H$) when not holding one ($\neg H$). Notice that $r_2$ does not affect $u$, enforcing that $u$ does not change.

For enabling efficient planning for problems of a target class of problem $\mathcal{Q}$, sketches must have *bounded width* and be *acyclic*. These properties, which we define next, lay the foundation for algorithms like $\text{SIW}_\text{R}$, which leverages them to decompose and solve problems iteratively (Bonet and Geffner 2021; Bonet and Geffner 2024).

The $\text{SIW}_\text{R}$ algorithm is a variant of the SIW algorithm (Lipovetzky and Geffner 2012) for solving any problem $P$ in $\mathcal{Q}$. The $\text{SIW}_\text{R}$ algorithm tracks the current state $s$, initially set to the initial state $s_0$, and iteratively solves the subproblem $P[s, G_R(s)]$ to find a subgoal state $s'$ in $G_R(s)$. After finding $s'$, the algorithm updates $s = s'$ and repeats this process until $s$ is a goal state of $P$. Bounded sketch width ensures that IW solves each subproblem $P[s, G_R(s)]$ efficiently (Bonet and Geffner 2021). Sketch acyclicity ensures that the iterative subproblem-solving process of $\text{SIW}_\text{R}$ never re-encounters subproblems, and hence, prohibits infinite execution (Drexler, Seipp, and Geffner 2022).

**Definition 7 (Sketch width).** The width $w_R(\mathcal{Q})$ of a sketch $R$ over a closed class of problems $\mathcal{Q}$ is $w_R(\mathcal{Q}) = \max_{P \in \mathcal{Q}} w(P[s_0, G_R(s_0)])$ where each $P[s_0, G_R(s_0)]$ is a problem with initial state $s_0$ and goal states $G_R(s_0)$.

**Definition 8 (Sketch acyclicity).** A sketch $R$ is acyclic in $\mathcal{Q}$ if there is no sequence of states $s_1, \ldots, s_n$ with $s_1 = s_n$ over the reachable states of any problem $P$ in $\mathcal{Q}$ such that $[s_i, s_{i+1}]$ is compatible with $R$, for all $0 \leq i < n$.

**Theorem 1.** *Consider a class of problems $\mathcal{Q}$ and an acyclic sketch $R$ over feature $\Phi$ whose width $w_R(\mathcal{Q})$ is bounded by $k$, i.e., $w_R(\mathcal{Q}) \leq k$. The $\text{SIW}_\text{R}$ algorithm solves any $P$ in $\mathcal{Q}$ in time $\mathcal{O}(N^{|\Phi|}(N^{k+1} + bN^{2k-1}))$ and space $\mathcal{O}(bN^k)$ producing a plan of length $\mathcal{O}(N^{|\Phi|+k})$ where $b$ bounds the branching factor in $P$, i.e., the maximum number of applicable ground actions in a state in $P$, $N$ is the number of ground atoms in $P$ (Bonet and Geffner 2021; Bonet and Geffner 2024).*

Theorem 1 assumes that all features $f$ in $\Phi$ are *linear*, i.e., $f$ can be evaluated in a state in time $\mathcal{O}(N)$, and take values from $\{f(s) \mid s \in S\}$ of size $\mathcal{O}(N)$. This thesis assumes that all features are *polynomial*, i.e., the time complexity and the number of possible feature values are polynomially upper-bounded in the size of a reasonable problem representation. These higher complexities introduce polynomial factors into the time complexity of $\text{SIW}_\text{R}$ but the runtime of $\text{SIW}_\text{R}$ remains exponential in $|\Phi|$ and $k$. Thus, given an acyclic policy sketch $R$ over a fixed set of polynomial features $\Phi$ whose width is bounded by $k$ for a class of problems $\mathcal{Q}$, $\text{SIW}_\text{R}$ solves any problem $P$ in $\mathcal{Q}$ in polynomial time.

## 2.5 Description Logics

Description logics (Baader et al. 2003) is a knowledge representation language where **concepts** represent unary relations and **roles** represent binary relations over the universe $\Delta$. In classical planning, for a state $s$ in a problem $P$, the universe $\Delta^s$ is the set of objects $O$ in $s$. The idea of using description logics in planning stems from work on learning general policies (Martín and Geffner 2000) and was used in several subsequent works (e.g., Fern et al. 2004; Bonet and Geffner 2018; Francès, Corrêa, et al. 2019; Ståhlberg, Francès, et al. 2021; Francès, Bonet, et al. 2021; Ferber et al. 2022). Similarly, we use a description logics to represent features for classes of problems $\mathcal{Q}$.

We follow the grammar definition from work on learning general policies (Francès, Bonet, et al. 2021). This thesis only shows the subset of rules we explicitly refer to, where $C, D$ are concepts and $R, S$ are roles.

- *atomic concept (resp. role)* $p$ for unary (resp. binary) predicate $p$ in predicates $\mathcal{R}$ of domain $\mathcal{D}$ with denotation $p^s = \{\bar{o} \mid p(\bar{o}) \in s\}$,
- *atomic goal concept (resp. role)* $p_g$ for unary (resp. binary) predicate $p$ in predicates $\mathcal{R}$ of domain $\mathcal{D}$ with denotation $p_g^s = \{\bar{o} \mid p(\bar{o}) \in \gamma\}$,
- *universal concept* $\top$ with denotation $\top^s \equiv \Delta^s$,
- *role-value mapping* $R = S$ with denotation $(R = S)^s \equiv \{a \in \Delta^s \mid (a, b) \in R^s \leftrightarrow (a, b) \in S^s\}$,
- *concept intersection* $C \sqcap D$ with denotation $(C \sqcap D)^s = C^s \cap D^s$,
- *concept negation* $\neg C$ with denotation $(\neg C)^s \equiv \Delta^s \setminus C^s$, and
- *existential abstraction* $\exists R.C$ with denotation $(\exists R.C)^s \equiv \{a \in \Delta^s \mid \exists b : (a, b) \in R^s \wedge b \in C^s\}$.

Observe that atomic goal concepts and roles strictly require that the goal $\gamma$ of a problem is a set of ground atoms and, hence, signals a language limitation. For a concept or role $X$, we write $\|X^s\|$ to denote the Boolean feature that evaluates to false if $X^s$ is empty and true otherwise and $|X^s|$ for the numerical feature that evaluates to the number of elements in $X^s$. The number of grammar rule applications defines the syntactic complexity of a feature.

**Example 7.** Consider the feature $H$ from Example 6. We can represent $H$ using description logics grammar rules as $H \equiv \|holding\|$, which has a syntactic complexity of two because the atomic concept $holding$ has a complexity of one and its composition to a Boolean feature results in a complexity of two.

## 2.6 Relational Structures and Graphs

In this section, we follow the notation of Drexler, Ståhlberg, et al. (2024b). Consider a planning problem $P$ over domain $\mathcal{D}$. Each state $s$ from $P$ induces a **relational structure** $\mathfrak{A}^s$ with *universe* $U^s = O$ for the set of objects $O$ in $s$, *signature* that is the set of domain predicate $\mathcal{R}$, and *interpretations* $p^s \subseteq (U^s)^k$ for each predicate $p/k$ in the planning domain $\mathcal{D}$, where $\langle \bar{o} \rangle$ in $p^s$ iff the ground atom $p(\bar{o})$ is in $s$. We assume fully relational structures that do not contain functions or constants, which are adequate for our assumed classical planning formalism.

Graphs can encode relational structures. We use isomorphisms between graph encodings of relational structures to define state equivalence.

A **directed graph**, or graph, is a pair $G = (V, E)$ where $V$ is a set of vertices and $E \subseteq V^2$ is a set of edges. Two graphs $G = (V, E)$ and $G' = (V', E')$ are **isomorphic**, denoted by $G \simeq_g G'$, if there is a bijection $f : V \to V'$ such that $(u, v)$ in $E$ iff $(f(u), f(v))$ in $E'$.

An **undirected graph** is a directed graph $G$ where $E$ is symmetric, i.e., $(v, w)$ in $E$ iff $(w, v)$ in $E$.

A **vertex-colored graph** is a tuple $G = (V, E, \lambda)$ where $(V, E)$ is a graph, and $\lambda : V \to \mathcal{C}$ maps vertices to the colors in $\mathcal{C}$. Two vertex-colored graphs $G = (V, E, \lambda)$ and $G' = (V', E', \lambda')$ are **isomorphic**, denoted as $G \simeq_g G'$, iff there is a *color preserving isomorphism* $f$ from $G$ to $G'$, i.e., $\lambda(v) = \lambda'(f(v))$ for $v$ in $V$.

We will use undirected vertex-colored graphs to represent the relational structure of a state. These graphs are designed so that applying the isomorphism to the ground atoms of a state results in a state where the objects are in the same relationship, just under a different object name.

# 3 Expressing the Subgoal Structure

## Core Publication of this Chapter

- Dominik Drexler, Jendrik Seipp, and Hector Geffner (2024). "Expressing and Exploiting Subgoal Structure in Classical Planning Using Sketches". In: *Journal of Artificial Intelligence Research* 80, pp. 171–208.

Researchers have introduced several languages for representing general plans, including general policies (Francès, Bonet, et al. 2021), programs (Segovia-Aguas et al. 2019), linear temporal logics (LTL) (Bacchus and Kabanza 2000), and reward machines (Icarte et al. 2022), each offering unique strengths and limitations. Unlike programs or policies that specify immediate low-level operations or actions to take and, hence, require no search, LTL and reward machines specify subgoals. These subgoals are higher-level abstractions that may require the agent to execute several actions. These higher-level abstractions extend beyond plan generation. They are more adaptable to changes in environmental dynamics and can accelerate policy learning, such as in environments with sparse rewards (Singh et al. 2010; Zheng et al. 2020).

We consider the recently introduced policy sketches language (Bonet and Geffner 2021; Bonet and Geffner 2024) a direct generalization of general policies that uses the same syntax but slightly different semantics. While

general policies specify immediate actions to take, policy sketches aim to *break problems into subproblems with controllable complexity through a single parameter*. Policy sketches incorporate planning width to characterize the polynomial complexity of subproblems (Lipovetzky and Geffner 2012). Unlike other approaches like LTL or reward machines, policy sketches incorporate an explicit complexity constraint to characterize an effective decomposition.

One of the most competitive approaches to address generalized planning is solving individual problems using heuristic search (Hart et al. 1968). These methods compute problem-specific goal distance estimators, called heuristics, to guide a search towards a goal state (e.g., Richter and Westphal 2010; Hoffmann and Nebel 2001). However, while widely effective, heuristic search struggles to exploit the structure in many problems. In the worst case, it scales exponentially in the problem size (Bonet and Geffner 2001).

Width-based search algorithms overcome exponential worst-case behavior at the cost of rendering the search incomplete. The iterative width (IW) algorithm efficiently solves arbitrary-sized problems with small, bounded widths, which applies to many classical planning benchmarks where the goal consists of a single atom (Lipovetzky and Geffner 2012). However, many real-world planning problems involve conjunctive goals involving multiple interacting subgoals. For these problems, the serialized iterative width (SIW) algorithm decomposes the problem into subproblems, achieving one goal atom one at a time, and stops when solving the problem (Lipovetzky and Geffner 2017). SIW is simple and effective for many problems. However, extensions that combine it with heuristic search *complicate understanding the resulting decomposition* and the interplay between heuristic guidance and width-based search.

Policy sketches build on the idea of problem decomposition and serialization to express them in an *interpretable* way. Motivated by work in qualitative numeric planning, a decidable fragment of numeric planning where Boolean and numerical features change qualitatively (Bonet and Geffner 2020), policy sketches provide a symbolic framework for defining temporal abstractions. Using sketch rules to constrain qualitative feature changes, an acyclic sketch $R$ of bounded width can effectively guide the search process via the $\text{SIW}_\text{R}$ algorithm. This approach is similar to SIW but focuses on identifying subgoal states that satisfy sketch-imposed constraints (Bonet and Geffner 2021).

In this chapter, we investigate whether policy sketches are a suitable language for representing common subgoal structures in classical planning domains. Our example-driven approach evaluates handcrafted policy sketches of bounded width across several tractable classical planning domains from the International Planning Competition, which are known to challenge heuristic search and width-based methods. Each policy sketch consists of only a few rules and

features designed to exploit the structural properties of these domains. We empirically compare the handcrafted policy sketches against state-of-the-art heuristic search methods to validate their theoretical advantages. To illustrate their intrinsics, it follows a presentation and discussion of the handcrafted policy sketch for the Childsnack domain.

# 3.1 Example Policy Sketch

This section presents the handcrafted policy sketch (Drexler, Seipp, and Geffner 2024) for the Childsnack domain, a benchmark from the International Planning Competition. Unlike the previous presentation of the sketch, we place additional emphasis on the feature representations, which are critical for understanding its computational value.

We begin by introducing the Childsnack domain and outlining its class of problems. Next, we define a set of features that capture the key domain aspects, focusing on their computational and syntactic complexity. Finally, we present the policy sketch and demonstrate that it is acyclic and has a bounded width, ensuring that it can be used with the $\text{SIW}_\text{R}$ algorithm to solve any problem in the domain in polynomial time.

## 3.1.1 Domain Description

In the Childsnack domain (Vallati et al. 2018), objects include contents, trays, bread, children, a kitchen, and tables. Contents and bread can be regular or gluten-free, and children may be allergic to gluten. The domain features actions for creating regular or gluten-free sandwiches using corresponding types of content and bread. Sandwiches can be placed on trays in the kitchen and transported to different places. If a tray is at a table with a child, the sandwich can be served, provided it meets the constraint that regular sandwiches cannot be served to gluten-allergic children. The domain defines the following predicates to define states with their intuitive meanings:

- $\mathit{waiting}(c : \mathit{child}, p : \mathit{place})$,
- $\mathit{served}(c : \mathit{child})$,
- $\mathit{notexists}(s : \mathit{sandwich})$,
- $\mathit{ontray}(s : \mathit{sandwich}, t : \mathit{tray})$,
- $\mathit{not\text{-}allergic\text{-}gluten}(c : \mathit{child})$,
- $\mathit{allergic\text{-}gluten}(c : \mathit{child})$,

- $at(t : tray, p : place)$,
- $at\text{-}kitchen\text{-}bread(b : bread\text{-}portion)$,
- $at\text{-}kitchen\text{-}content(c : content\text{-}portion)$,
- $at\text{-}kitchen\text{-}sandwich(s : sandwich)$,
- $no\text{-}gluten\text{-}bread(b : bread\text{-}portion)$,
- $no\text{-}gluten\text{-}content(c : content\text{-}portion)$,
- $no\text{-}gluten\text{-}sandwich(s : sandwich)$.

In the initial state, children are seated at their tables, and trays are either in the kitchen or at the tables. The problem is solvable with enough gluten-free content and bread available to serve all children. The goal is to serve a sandwich to each child. Goal serialization using SIW fails: serving gluten-free sandwiches to non-allergic children can lead to unsolvable states if too few remain for those who need them. Our sketch will fix this issue by ensuring that gluten-free sandwiches are only served to gluten-allergic children.

### 3.1.2 Features

In the Childsnack domain, we define a set $\Phi$ comprising six key features. It contains two numerical features $c_a$ and $c_r$ to track the number of unserved gluten-allergic, respectively, non-gluten-allergic children, two Boolean features $s_a^k$ and $s^k$ to track the availability of a gluten-free sandwich, respectively, any sandwich, in the kitchen, and two Boolean features $s_a^t$ and $s^t$ to track the availability of a gluten-free sandwich on a tray, respectively, any sandwich on a tray. We define the set of features $\Phi$ over the domain predicates as follows:

$$
\begin{aligned}
c_a &\equiv |allergic\text{-}gluten \sqcap served_g \sqcap \neg served| \\
c_r &\equiv |not\text{-}allergic\text{-}gluten \sqcap served_g \sqcap \neg served| \\
s_a^k &\equiv \|at\text{-}kitchen\text{-}sandwich \sqcap no\text{-}gluten\text{-}sandwich\| \\
s^k &\equiv \|at\text{-}kitchen\text{-}sandwich\| \\
s_a^t &\equiv \|\exists ontray.\top \sqcap no\text{-}gluten\text{-}sandwich\| \\
s^t &\equiv \|\exists ontray.\top\|
\end{aligned}
$$

Evaluating all features in $\Phi$ is efficient, with time linear in the number of state atoms and number of distinct feature values linear in the number of objects. Transitive closure, part of our complete grammar, generally has the highest runtime complexity, cubic in the number of objects in a problem. The features $c_a$ and $c_r$ have the largest syntactic complexity of seven, e.g., $c_a$ contains two atomic concepts, one atomic goal concept, one concept negation, two concept intersections, and one numerical composition.

### 3.1.3 Sketch Rules

The policy sketch for the Childsnack domain encodes an ordering of subgoals to prioritize gluten-free ingredients for gluten-allergic children, effectively managing resource constraints and avoiding unsolvable states. It consists of two triplets of rules, $\{r_1, r_2, r_3\}$ and $\{r_4, r_5, r_6\}$, which enforce this prioritization by serving gluten-free sandwiches to gluten-allergic children before addressing non-allergic children. The first triplet defines subgoals for serving gluten-allergic children ($c_a > 0$). In contrast, the second triplet defines subgoals for serving non-gluten-allergic children after all gluten-allergic children are served ($c_a = 0$). The policy sketch $R^C_\Phi$ over features $\Phi$ is defined as follows:

$$
\begin{aligned}
r_1 &= \{c_a > 0, \neg s^k_a, \neg s^t_a\} \mapsto \{s^k_a, s^k\} \\
r_2 &= \{c_a > 0, s^k_a, \neg s^t_a\} \mapsto \{s^k_a?\,, s^k?\,, s^t_a, s^t\} \\
r_3 &= \{c_a > 0, s^t_a\} \mapsto \{c_a{\downarrow}, s^t_a?\,, s^t?\,\} \\
r_4 &= \{c_a = 0, c_r > 0, \neg s^k, \neg s^t\} \mapsto \{s^k\} \\
r_5 &= \{c_a = 0, c_r > 0, s^k, \neg s^t\} \mapsto \{s^k_a?\,, s^k?\,, s^t_a?\,, s^t\} \\
r_6 &= \{c_a = 0, c_r > 0, s^t\} \mapsto \{c_r{\downarrow}, s^t_a?\,, s^t?\,\}
\end{aligned}
$$

The first triplet of rules splits the problem of serving a sandwich to gluten-allergic children into subproblems. When such a child exists, the respective feature condition in all rules of the first triplet is satisfied ($c_a > 0$), and the respective feature condition in all rules of the second triplet is unsatisfied ($c_a = 0$). Rule $r_1$ says that if there is no gluten-free sandwich in the kitchen ($\neg s^k_a$) or tray ($\neg s^t_a$), make such a sandwich ($s^k_a$). Rule $r_2$ says that if there is such a sandwich in the kitchen ($s^k_a$) but not on a tray ($\neg s^t_a$), put it from the kitchen on a tray ($s^t_a$). Rule $r_3$ says that if there is such a sandwich on a tray ($s^t_a$), serve a gluten-allergic child ($c_a{\downarrow}$). Notice that some of these rules have additional side effects on other features and that achieving a subgoal might require several steps. Suppose all gluten-allergic children were served or there are none ($c_a = 0$). In that case, the second triplet of rules splits the problem of serving a sandwich to non-gluten-allergic children similarly but uses less constrained features that do not require the sandwiches to be gluten-free.

In summary, the sketch successfully imposes an ordering on the subgoals, preventing it from leading to unsolvable states. Each triplet of rules decomposes a subproblem of serving a sandwich into making the sandwich, moving it on a tray, and serving sandwiches while abstracting details like the specific tray and places, effectively reducing the width of the subproblems for serving a sandwich to 1. Furthermore, the sketch exhibits acyclic behavior when executed with $\text{SIW}_\text{R}$. The following proposition summarizes its theoretical properties.

**Proposition 2.** *The handcrafted sketch $R^C_\Phi$ for the Childsnack domain is acyclic and has width* 1.

## 3.2 Experiments

We ran experiments to analyze the computational value of policy sketches on a set of classical planning benchmarks from the International Planning Competition (IPC). We implemented $\text{SIW}_\text{R}$ in the LAPKT planning system (Ramirez et al. 2015). We compare our approach against the state-of-the-art domain-independent heuristic search planners LAMA (Richter and Westphal 2010) and Dual-BFWS (Lipovetzky and Geffner 2017), as well as standard goal serialization SIW (Lipovetzky and Geffner 2012). For each problem, we imposed a time limit of 30 minutes and a memory limit of 3 GiB.

Table 3.1 highlights the key results of our experiments. We report the total number of solved problems (S) and, for commonly solved problems, the maximum runtime in seconds (T), along with the average (A) and the maximum (M) effective width across all encountered subproblems.

Depending on the domain, $\text{SIW}_\text{R}$ performed significantly better in terms of either the number of solved problems, maximum runtime, or both. SIW fails on all domains, indicating that goal serialization is often insufficient. LAMA and Dual-BFWS solve only very few problems in Childsnack and Floortile and require significantly more time in other domains such as Barman with $760$ and $248$ seconds, respectively, compared to $\text{SIW}_\text{R}$ that requires at most $3$ seconds. The poor performance of Dual-BFWS results from subproblems that are still too difficult. A more extensive analysis with similar conclusions is available in Drexler, Seipp, and Geffner (2024). In summary, the experimental results demonstrate the computational strength of acyclic policy sketches with bounded width, confirming their advantages in performance and scalability.

## 3.3 Discussion

Through examples, we showed that policy sketches can capture the common subgoal structure in several classical planning domains from the International Planning Competition. Our sketches are *acyclic*, meaning that $\text{SIW}_\text{R}$ does not trap into cycles, have *bounded width*, meaning that the complexity of subproblems is polynomially upper bounded, are *compact*, meaning that they encode the common subgoal structure with a small number of syntactical elements. Related languages such as linear temporal logics (Bacchus and Kabanza 2000) or reward machines (Icarte et al. 2022) do not aim at characterizing the

| | LAMA | | BFWS | | SIW(2) | | | $SIW_R(2)$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Domain | S | T | S | T | S | A | M | S | T | A | M |
| Barman (40) | **40** | 760 | **40** | 248 | 0 | – | – | **40** | 3 | 0.8 | 2 |
| Childsnack (20) | 6 | 3 | 9 | 172 | 0 | – | – | **20** | 2 | 0.6 | 1 |
| Driverlog (20) | **20** | 39 | **20** | 3 | 7 | 1.5 | 2 | **20** | 4 | 0.4 | 1 |
| Floortile (40) | 9 | 202 | 6 | 865 | 0 | – | – | **40** | 1 | 1.2 | 2 |
| Grid (5) | **5** | 3 | **5** | 3 | 2 | 2.0 | 2 | **5** | 2 | 0.8 | 2 |
| Schedule (150) | **150** | 38 | **150** | 103 | 78 | 1.1 | 2 | **150** | 13 | 0.0 | 0 |
| TPP (30) | **30** | 12 | **30** | 1234 | 21 | 2.0 | 2 | **30** | 8 | 0.2 | 1 |

**Table 3.1:** Comparison of the search algorithms LAMA, Dual-BFWS (BFWS), SIW(2), and $SIW_R(2)$ on several planning domains, showing the number of solved problems (S) and, for commonly solved problems, the maximum runtime (T) in seconds, along with the average (A) and maximum (M) effective width across all encountered subproblems. We use boldface to denote the highest number of solved problems.

polynomial complexity of subproblems. However, this property yields practical advantages, as validated by experiments showing that $SIW_R$ outperforms state-of-the-art methods in challenging benchmark domains.

The main limitation of the policy sketches language, in its current form, is its limited ability to address the problem of *reuse*. Reuse is central to improving efficiency by leveraging structural similarities across different problem classes. Humans rarely solve problems entirely from scratch but instead restructure previously acquired knowledge while integrating newly acquired knowledge (Ellis et al. 2020; Dumancic et al. 2021).

To illustrate our concern, consider classes of problems such as doing the laundry, cleaning the floor, watering the plants, washing the dishes, or preparing meals. It is easy to see that general plans in all these classes involve getting hold of different kinds of objects. Notice that features *embedded directly* into the sketch rules specify the objects to be held based on their characteristics. Hence, the language of policy sketches is insufficient in the broader scope of artificial intelligence, where a central objective is to build agents capable of learning and representing general plans for numerous problem classes (LeCun 2022).

We addressed this limitation in a follow-up work (Bonet, Drexler, et al. 2024), where we wrapped policy sketches into so-called modules. Modules are essentially parameterized policy sketches that take as input concepts or roles. Modules allow for the injection of additional context that may depend on the problem class. A module for getting hold of objects in different classes might take a concept that specifies the object based on class-dependent char-

acteristics as input. The sketch might consist of two rules: one for moving to the location of the object and another one for picking it up. Modules can also call other modules, allowing for the composition of primitive into more complex behaviors. The injection of the concepts and roles via arguments allows modules to work for different problem classes.

The manual creation of policy sketches is time-intensive and domain-specific, limiting scalability and applicability. The next chapter presents an unsupervised method for learning policy sketches. While the problem of reuse is important, we have looked into it exclusively from the representational aspect. In the following, we return to the original form of the language, leaving extensions of our methods for future work.

# 4 Learning Policy Sketches

## Core Publication of this Chapter

- Dominik Drexler, Jendrik Seipp, and Hector Geffner (2024). "Expressing and Exploiting Subgoal Structure in Classical Planning Using Sketches". In: *Journal of Artificial Intelligence Research* 80, pp. 171–208.

Manual creation of policy sketches is time-intensive and domain-specific, limiting scalability and applicability. This chapter addresses these challenges by introducing an unsupervised method for learning policy sketches that decomposes planning problems into subproblems of polynomial complexity. Our method generalizes the method for learning general policies (Francès, Bonet, et al. 2021) by incorporating planning width that measures the polynomial complexity of a problem (Lipovetzky and Geffner 2012). We also employ combinatorial optimization and optimize for the syntactically simplest solution, i.e., policy sketch, increasing the likelihood of generalization to the target class of problems. The method for learning general policies relied on manually selected training problems, requiring significant human interaction and domain knowledge. Therefore, we incorporate curriculum learning (Bengio et al. 2009), tailored explicitly for learning from small problems, measured by the number of states. Our curriculum learning approach iteratively exposes the

learning procedure to a set of problem with a growing number of states only upon validation failures. Our approach simplifies applying learning approaches to unseen domains by incrementally exposing them to more data.

This chapter is organized as follows. First, we describe our learning method, which consists of three main steps. Second, we show the results of learning policy sketches of bounded width for several tractable classical planning domains from the International Planning Competition. Our evaluation includes an analysis of the learned sketch for the Childsnack and a comparison against the previously handcrafted sketch from Chapter 3. Last, we conclude and discuss the limitations of our method, recent advances in learning sketches with deep learning, and promising directions for future research.

# 4.1 Method

Our method is closely related to the method of learning general policies. It simultaneously learns the features $\Phi$ and policy sketch $R$ over the features $\Phi$ on a training set and an automatically generated pool of features using combinatorial optimization. We start by describing our curriculum learning approach to generate training sets of problems. Next, we describe how to automatically derive a pool of features from a training set. Last, we describe the combinatorial encoding for learning a sketch from a training set and its corresponding pool of features using answer set programming.

## 4.1.1 Data Generation

We sample a finite training set $\mathcal{Q}_\mathcal{T} \subseteq \mathcal{Q}$ using a problem generator. Directly learning a sketch from $\mathcal{Q}_\mathcal{T}$ is infeasible due to the large number of states it may contain. To address this limitation, we use a curriculum learning approach that iteratively trains on a subset of problems $\mathcal{P} \subseteq \mathcal{Q}_\mathcal{T}$, initially empty. If a learned policy sketch fails to generalize, the procedure identifies an increasingly more complex training set from $\mathcal{Q}_\mathcal{T}$. The procedure adds the problem with the fewest states where the sketch fails to set a bound on the sketch width or the sketch is not acyclic, and it may discard all problems first if this problem is larger than any of the problems in the training set.

Our curriculum learning approach is particularly advantageous, as it incrementally exposes the learning process to more complex problems. Ideally, the process converges before $\mathcal{P}$ grows to the size of $\mathcal{Q}_\mathcal{T}$, effectively reducing computational costs by considering small amounts of data.

### 4.1.2 Feature Construction

From a training set $\mathcal{P}$, we derive a pool of features $\mathcal{F}$ using the description logic grammar-based approach from the work on learning general policies.

The method takes as input an additional integer value that limits the syntactical complexity of the generated sentences. The lowest complexity sentences are primitive concepts and roles, representing the domain's unary and binary predicates or the universal concept, each with complexity one. Then, the procedure iteratively applies composite grammar constructors, such as concept intersection or existential abstraction, whose sentence complexity is the sum of the involved sub-sentences plus one. The procedure also composes Boolean and numerical features, each adding a complexity of one to the sub-sentence.

The procedure prunes features that do not add information, i.e., whose denotation is precisely the same on all states as a previously generated feature. The resulting Boolean and numerical features become members of the feature pool $\mathcal{F}$. The number of features grows exponentially in the limit imposed on the syntactic complexity. The number of features directly influences the size of our combinatorial encoding as it considers all features from the pool. Therefore, we are limited in the magnitude that we can set for the limit on the syntactical complexity. The curriculum learning approach also helps mitigate the combinatorial explosion, as problems with fewer objects result in numerical features with fewer possible valuations.

### 4.1.3 Answer Set Program

Our method learns acyclic sketches of bounded width using combinatorial optimization formulated as an answer set program (ASP) (Gelfond and Lifschitz 1988; Gebser et al. 2012). From the training set $\mathcal{P}$, the feature pool $\mathcal{F}$, and a bound $k$ on the sketch width, we generate an answer set program that includes constraints to ensure that a solution, i.e., policy sketch $R$, for $\mathcal{P}$ is acyclic and has width bounded by $k$. We optimize for the syntactically simplest solution by minimizing the sum of syntactic feature complexities over the selected features in $\Phi$. We refer the reader to the chapter's main contribution for complete encoding details.

Our encoding is *sound and complete* in the sense that it has a solution, i.e., answer set, if and only if there exists an acyclic policy sketch over a subset of features from $\mathcal{F}$ that has width bounded by $k$ on the training set $\mathcal{P}$. The resulting policy sketch can be read directly from an answer set of the program.

## 4.2 Experiments

We ran experiments to learn sketches for several tractable classical planning domains from the International Planning Competition (IPC). We generated a set of small training problems using PDDL generators (Seipp et al. 2022) and imposed a limit on the syntactic feature complexity of eight.

Table 4.1 shows the key results of learning sketches with width $k$ upper bounded by $0, 1$ and $2$. For each width, we report the peak memory usage in GiB (M), the total wall-clock time in hours for solving the answer set program in parallel on 32 CPU cores (T), the number of states in the last iteration of the curriculum learning procedure (#S), the number of selected features (#$\Phi$), and the number of learned sketch rules (#R).

The number of selected features and rules is always small, with at most $4$, respectively $5$, in Childsnack. As $k$ increases, less complex features and sketches are needed, but the search becomes more costly, albeit still polynomial. For $k$ equal to $0$, we observe failures to learn a sketch in Childsnack, Delivery, and Miconic for different reasons. In Childsnack, we suspect that the expressivity of the feature language is insufficient. Same for Delivery, where we know that a distance feature of complexity $15$ is necessary (Francès, Bonet, et al. 2021). In Miconic, we ran out of resources but succeeded in subsequent works (Drexler, Seipp, and Geffner 2023; Drexler, Ståhlberg, et al. 2024b). We solve all domains for $k$ equal to $1$, indicating a good balance between the feature expressivity and resources needed for learning a sketch. We empirically verified the correctness of our learned sketches on larger problems and formally showed the correctness of a subset of them.

We can also observe a large required time of $63.34$ hours and memory of $122$ GiB in Childsnack. This demonstrates a clear limitation of our approach when dealing with more complex domains, where memory and time consumption can become prohibitive. Complex domains often require many states that may induce large feature pools and, thus, even larger combinatorial encodings that are difficult to solve. We also tried domains such as Barman and Grid, where the number of required states was too large.

These results demonstrate the scalability limitations and a key strength of our approach, i.e., its ability to learn from small amounts of input data. As domain complexity increases, the memory and time requirements grow significantly in the number of states, the size of the feature pool, and the bound imposed on the sketch width. Reducing the size of the state space or the feature pool is crucial for improving scalability. In Chapter 6, we will present a method to reduce redundancy in the training data by removing symmetric states, which can result in exponentially smaller training data sets.

| | $k=0$ | | | | | $k=1$ | | | | | $k=2$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Domain | M | T | #S | #Φ | #R | M | T | #S | #Φ | #R | M | T | #S | #Φ | #R |
| Blocks-clear | 1 | 0.01 | 22 | 2 | 2 | 1 | 0.01 | 22 | 1 | 1 | 1 | 0.01 | 22 | 1 | 1 |
| Blocks-on | 26 | 3.89 | 22 | 3 | 3 | 9 | 0.03 | 22 | 2 | 2 | 13 | 0.05 | 22 | 1 | 1 |
| Childsnack | – | – | – | – | – | 122 | 63.34 | 792 | 4 | 5 | – | – | – | – | – |
| Delivery | – | – | – | – | – | 17 | 0.15 | 96 | 2 | 2 | 3 | 0.01 | 20 | 1 | 1 |
| Gripper | 2 | 0.01 | 28 | 2 | 3 | 3 | 0.02 | 28 | 2 | 2 | 7 | 0.02 | 28 | 1 | 1 |
| Miconic | – | – | – | – | – | 1 | 0.01 | 32 | 2 | 2 | 2 | 0.01 | 32 | 1 | 1 |
| Reward | 3 | 0.02 | 26 | 2 | 2 | 1 | 0.01 | 12 | 1 | 1 | 10 | 0.03 | 48 | 1 | 1 |
| Spanner | 12 | 0.56 | 227 | 2 | 2 | 3 | 0.01 | 74 | 1 | 1 | 6 | 0.01 | 74 | 1 | 1 |
| Visitall | 3 | 0.02 | 36 | 2 | 2 | 1 | 0.01 | 3 | 1 | 1 | 1 | 0.01 | 3 | 1 | 1 |

**Table 4.1:** Comparison of learning sketches for a class of problems $\mathcal{Q}$ with width $k$ upper bounded by $0, 1$ and $2$, showing the peak memory in GiB (M), the total wall-clock time in hours for solving the ASP on 32 CPU cores (T), the total number of states in the last iteration of the curriculum learning procedure (#S), the number of selected features (#Φ), and the number of sketch rules (#R). We denote failures by "–" and report the reasons in the text.

## 4.3 Analysis

To illustrate the power of our method, we now present the learned sketch of width $1$ for the Childsnack domain while comparing it with the previously handcrafted sketch from Chapter 3. Our automated method simplifies the sketch by minimizing the sum of syntactic feature complexities and fully exploiting the structure of the training problems while maintaining generalization. The learned features $\Phi = \{c_a, c, s_a, s^k\}$ are:

$$
\begin{aligned}
c_a &\equiv |\mathit{allergic\text{-}gluten} \sqcap \mathit{served}_g \sqcap \neg \mathit{served}| \\
c &\equiv |\mathit{served}| \\
s_a &\equiv |\mathit{no\text{-}gluten\text{-}sandwich}| \\
s^k &\equiv |\mathit{at\text{-}kitchen\text{-}sandwich}|
\end{aligned}
$$

The feature $c_a$ is identical to its counterpart in the handcrafted sketch, while $c$ merges $c_r$ and $c_a$ into a simpler representation by counting served children. Similarly, $s_a$ abstracts the location of gluten-free sandwiches more simply than $s^k_a$, which specifically tracks their presence in the kitchen. The feature $s^k$ is now a numerical feature, unlike the Boolean feature in the handcrafted sketch, and the exclusion of features tracking sandwiches on trays indicates that the learned sketch leverages the domain's structural constraints, where sandwiches can only leave the kitchen by being placed on trays. The learned sketch demonstrates increased simplicity and efficiency with a total syntactic complexity of $13$, compared to $30$ for the handcrafted sketch. The learned sketch rules $R_\Phi$ over the features Φ are:

$$r_1 = \{\} \mapsto \{s^k?\,, c_a?\,, s_a\uparrow, c?\,\}$$
$$r_2 = \{\} \mapsto \{s^k\downarrow, c_a?\,, s_a?\,, c?\,\}$$
$$r_3 = \{\} \mapsto \{s^k?\,, c_a\downarrow, s_a?\,, c?\,\}$$
$$r_4 = \{c_a = 0\} \mapsto \{s^k\uparrow, c_a?\,, s_a?\,, c?\,\}$$
$$r_5 = \{c_a = 0\} \mapsto \{s^k?\,, c_a?\,, s_a?\,, c\uparrow\}$$

For example, the rule $r_1$ says that making a gluten-free sandwich is good, and the rule $r_2$ says that decreasing the number of sandwiches in the kitchen is good, effectively moving it on a tray. Notice that $r_2$ now works for both types of sandwiches because it can never result in an unsolvable state. Importantly, none of the rules explicitly track whether sandwiches are placed on a tray, exploiting the domain structure since decreasing the number of sandwiches in the kitchen is only possible by moving them on a tray.

The learned sketch is acyclic and has a width of 1, matching the theoretical result from the handcrafted sketch in Proposition 2.

**Proposition 3.** *The learned sketch $R_\Phi^C$ for the Childsnack domain is acyclic and has width* 1.

The analysis shows that our automated approach simplifies the handcrafted sketch by minimizing the sum of syntactic complexities while showing increased exploitation of the structure of the training problems.

## 4.4 Discussion

We introduced a method that automatically learns acyclic policy sketches of bounded width. It simultaneously learns state abstractions as Boolean and numerical features and temporal abstractions as sketch rules over the features in one shot. Our method is sound and complete for the training set and we can often show generalization towards the target class by hand. To the best of our knowledge, no other method *learns to split problems into subproblems with polynomial complexity that is controllable with a single integer parameter*, a step towards building reliable agents that learn subgoal structures to plan efficiently.

Our experimental evaluation shows that scalability is a primary concern of our method, preventing its application in complex real-world problems, including the household robot example. The main issue is the fast growth of the combinatorial encoding, which is polynomial in the number of states and the size of the feature pool and exponential in the bound imposed on the sketch width.

A promising direction for addressing the scalability issue is reuse. In this context, reuse means learning a policy sketch from previously acquired knowledge. We touched upon reuse in the previous chapter. We discussed so-called modules that are good candidates for reuse. Modules use the language of policy sketches and take additional concepts or roles as arguments to support reuse across different problem classes.

Instead of learning modules, one could also learn sketches from a pool of candidate sketch rules. That is, by first generating a pool of candidate sketch rules independently optimized for simplicity, followed by learning a sketch from the pool. This method effectively decouples the learning of state abstractions in step one from learning temporal abstractions in step two. Decoupling the learning of state and temporal abstractions allows for targeted optimizations, potentially reducing computational costs and improving scalability.

Recently, Aichmüller and Geffner (2024) showed that policy sketches can be learned with deep learning, a much more scalable framework. Their method uses graph neural networks (GNN) (Scarselli et al. 2009) that were first used for learning general policies (Ståhlberg, Bonet, et al. 2022a; Ståhlberg, Bonet, et al. 2022b; Ståhlberg, Bonet, et al. 2023). GNNs take arbitrary-sized graph structures that encode states as input. The features are implicit in the network weights, and the network predicts the subgoal states.

The expressivity of these GNNs is upper bounded by 2-variable first-order logic with counting quantifiers $C_2$ (Grohe 2021) and often sufficient for our training sets (Drexler, Ståhlberg, et al. 2024b). GNNs have a crisp characterization of expressivity compared to our feature pools, whose limit on the syntactic complexity complicates a precise characterization. Testing the correctness of GNNs on larger problems approximates generalization towards the target class of problems. However, GNNs lack interpretability, i.e., formally proving their correctness is challenging or impossible, which imposes risks in reliability and safety. An interesting and particularly challenging question for future work is how to tightly integrate the scalability of deep learning and the interpretability of simple target languages such as policy sketches.

Our contribution of learning policy sketches to split problems into subproblems of polynomial complexity controllable by a single integer parameter allows for an additional contribution. Learning hierarchical policies is a long-standing challenge in artificial intelligence that aims to find hierarchical decompositions of action plans without any prior structural information. Hierarchical policies are a representation of how humans tackle many complex real-world problems. The ability of policy sketches to control abstraction levels makes them a natural foundation for hierarchical policies. The following chapter presents a method for learning hierarchical policies based on our method for learning sketches.

# 5 Learning Hierarchical Policies

## Core Publication of this Chapter

- Dominik Drexler, Jendrik Seipp, and Hector Geffner (2023). "Learning Hierarchical Policies by Iteratively Reducing the Width of Sketch Rules". In: *Proceedings of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*. Ed. by Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner. IJCAI Organization, pp. 208–218.

Hierarchical policies are key to how humans and many animals efficiently solve complex, real-world problems. These agents prioritize higher-level planning before working out the details of a low-level action plan (LeCun 2022). The following example illustrates a multi-level plan for doing the laundry.

**Example 8.** Figure 5.1 shows a multi-level plan for doing the laundry. The highest-level plan consists of only two steps: placing all laundry pieces into the washing machine and launching the appropriate program. This plan involves putting individual pieces into the machine at a lower level, one at a time. A further breakdown might involve moving toward a laundry piece, picking it up, approaching the machine, and placing it inside. Notice that the multi-level plan applies to a large variety of households.

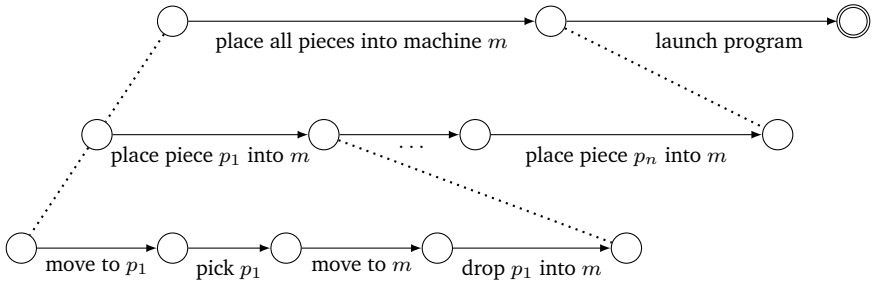



**Figure 5.1:** The illustration shows a multi-level plan for doing the laundry. The circle nodes represent states, and the double-circled node represents a goal state. The highest-level plan is at the top. The dotted lines indicate the respective decompositions into lower-level plans.

Humans excel at identifying appropriate levels of abstraction to enable efficient planning. However, automating this process for computational agents remains an important challenge (LeCun 2022). Despite decades of research in hierarchical planning, whether in model-based approaches (Sacerdoti 1974; Tate 1977; Erol et al. 1994) or model-free reinforcement learning (Parr and Russell 1997; Dietterich 2000; Barto and Mahadevan 2003), the problem of learning hierarchical plan representations *without supervision* remains an open challenge. A central issue is the absence of precise characterizations for describing and uncovering effective hierarchical structures. Researchers have explored strategies such as identifying "bottleneck states" (McGovern and Barto 2001), precondition relaxation (Sacerdoti 1974), and analyzing causal graphs (Knoblock 1994), but have restricted scopes, lacking either in learning compact representations to enable interpretability, a precise characterization of their effectiveness, or fully unsupervised methods for learning them.

For example, hierarchical policies expressed in hierarchical task networks (HTNs) (Erol et al. 1994) decompose classes of problems into classes of subproblems. Primitive tasks at the lowest level represent primitive actions or subproblems solvable in one step. Compound tasks at a higher level represent compositions of tasks or subproblems solvable in several steps. However, there is no characterization of what constitutes an effective decomposition.

Planning width is a well-suited to characterize an effective decomposition. Subproblems induced by a primitive task are solvable in a single step and, hence, have a width of one. Subproblems induced by compound tasks require several steps and potentially have greater width. Bounded width ensures that all subproblems have *polynomial complexity*, a crisp characterization, that can help in unsupervised uncovering of effective hierarchical structures.

In this chapter, we formalize this idea while using the policy sketches language and our method for learning them. Our method systematically uncovers a hierarchical structure by *iteratively splitting classes of problems into subclasses of problems whose polynomial complexity decreases*, measured by the planning width. Our method produces an interpretable hierarchical policy that is a single-rooted tree where each node contains a sketch rule and represents a class of problems. The width of classes of problems in the leaf nodes is zero and increases as we move up in the tree.

This chapter is organized as follows. First, we characterize our hierarchical policies and the validity property for effective hierarchical problem decomposition. We illustrate our characterization on a hierarchical policy for the class of problems over the laundry domain, which we learned for the equivalent Delivery domain (Bonet and Geffner 2021). Second, we present an empirical evaluation of learning hierarchical policies. Last, we summarize and conclude.

# 5.1 Characterization

In this section, we present our formal characterization of hierarchical policies. We illustrate our characterization using the class of problems $\mathcal{Q}_L$ over the laundry domain. Recall that in each problem, there is a robot, laundry pieces, and a washing machine that specifies the goal location. The objective is to move all laundry pieces to the goal location. There are actions to pick up and drop laundry pieces and move them between neighboring locations. A hierarchical policy is syntactically defined as follows.

**Definition 9.** A hierarchical policy $\Pi$ for a class of problems $\mathcal{Q}$ is a single-rooted tree where every node $n$ has a sketch rule $r(n)$ over features $\Phi$.

**Example 9.** Figure 5.2 illustrates a hierarchical policy for the class $\mathcal{Q}_L$ of laundry problems, which consists of eight nodes. Each node is associated with a rule defined over a subset of the following features: $D$ is true iff all laundry pieces are delivered, $H$ is true if and only a laundry piece is being held, $p$ is the distance to the nearest undelivered laundry piece, $t$ is the distance to the washing machine, and $u$ is the number of undelivered laundry pieces. For example, rule $r(n_0)$ over feature $D$ says that delivering all laundry pieces is good ($D$). Rule $r(n_1)$ over feature $u$ says that decreasing the number of undelivered laundry pieces is good ($u\downarrow$), rule $r(n_2)$ over features $H$ and $u$ says that getting hold of an undelivered laundry piece is good ($H$) if not holding one ($\neg H$), rule $r(n_4)$ over features $H$ and $p$ says that decreasing the distance to an undelivered laundry piece is good ($p\downarrow$) if not holding one ($\neg H$), and rule $r(n_5)$ over features $H$ and $p$ says that picking up an undelivered laundry piece is good ($H$) if not holding one ($\neg H$) and it is at distance zero ($p = 0$).

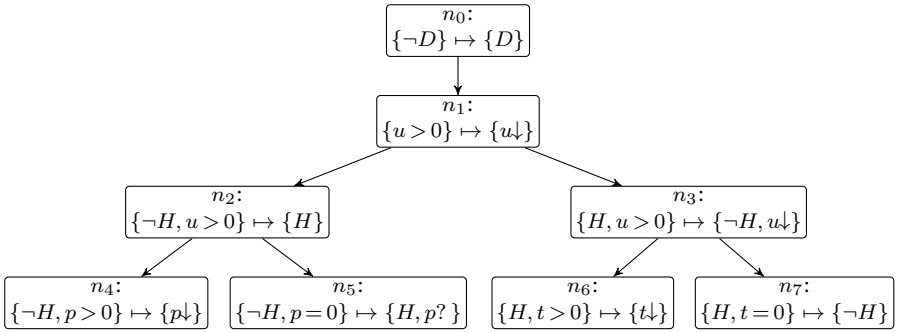



**Figure 5.2:** The illustration shows a valid hierarchical policy $\Pi^{\mathrm{L}}_2$ for class $\mathcal{Q}_L$.

We are interested in hierarchical policies with a computational characterization regarding polynomial time solvability of the class of problems $\mathcal{Q}$. Thus, we place additional structural restrictions based on the notion of width depending on three types of nodes: the root, inner, and leaf nodes. Each node captures a class of problems. The single root node captures the target class of problems that usually has unbounded width. Each inner node defines a problem decomposition into classes of subproblems with a strictly smaller width. Each leaf node captures a trivial class of subproblems with width zero, meaning that each subproblem is solvable by executing a single action. Our decomposition aims for an execution model that greedily picks child nodes to solve subproblems, which requires a redefinition of sketch width that upper bounds the width of each sketch rule independently from each other. Hierarchical policies that satisfy these restrictions are called *valid* (Drexler, Seipp, and Geffner 2023).

**Definition 10.** A hierarchical policy $\Pi$ for $\mathcal{Q}$ is *valid* if the rules $r(n)$ determine classes $\mathcal{Q}_n$ of subproblems from $\mathcal{Q}$ that together obey the constraints:

1. **Root node** $n$**:** The rule $r(n)$ is $\{\neg D\} \mapsto \{D\}$ where $D$ is a dummy Boolean feature that is true only in the goal of a problem $P$ in $\mathcal{Q}$, and $\mathcal{Q}_n = \mathcal{Q}$.
2. **Inner node** $n$**:** The rules $r(n')$ of the children $n'$ of $n$ encode an acyclic **sketch** $R(n)$ for $\mathcal{Q}_n$ with a **width** that is smaller than the width of $\mathcal{Q}_n$. The class of problems $\mathcal{Q}_{n'}$ at each child $n'$ with rule $r(n') = C \mapsto E$ is defined as $\mathcal{Q}_{n'} = \{P[s, G_{r(n')}(s)] \mid P[s, G] \in \mathcal{Q}_n, s \vDash C\}$.[2]
3. **Leaf node** $n$**:** The class of problems $\mathcal{Q}_n$ has **width** $0$, meaning that each problem $P$ in $\mathcal{Q}_n$ is solvable by executing a single action.

[2]We used that $\mathcal{Q}$ is closed to simplify the original definition and focus on the core idea of including each problem $P[s, G_{r(n')}(s)]$ that results from extending the goal states of problem $P[s, G]$ with initial state $s$ that satisfies $C$.

We write $\Pi_k$ to denote a valid hierarchical policy whose first decomposition has sketch width $k$. The following example illustrates the notion of validity.

**Example 10.** The hierarchical policy $\Pi_2^L$ for the class $\mathcal{Q}_L$ of laundry problems whose first decomposition has sketch width $2$ shown in Figure 5.2 is valid (Drexler, Seipp, and Geffner 2023):

- The root node $n_0$ contains the sketch rule over the dummy Boolean feature $D$ that holds only in the goal states of a problem from the class $\mathcal{Q}_0$, which is equal to $\mathcal{Q}$.
- The node $n_1$ encodes an acyclic sketch for $\mathcal{Q}_0$ over the feature $u$ with width $2$. Node $n_1$ represents the class of subproblems $\mathcal{Q}_1$ with width $2$ that aim to reduce the number of undelivered laundry pieces.
- The inner nodes $n_2$ and $n_3$ encode an acyclic sketch for $\mathcal{Q}_1$ over the features $u$ and $H$ with width $1$. Node $n_2$ represents the class $\mathcal{Q}_2$ of subproblems with width $1$ that aim to get hold of an undelivered laundry piece, while node $n_3$ represents the class of subproblems $\mathcal{Q}_3$ with width $1$, that aim to deliver a laundry piece that is already being held.
- The leaf nodes $n_4$ and $n_5$ encode an acyclic sketch for $\mathcal{Q}_2$ over the features $H$ and $p$ with width $0$. Node $n_4$ represents the class of subproblems $\mathcal{Q}_4$ with width $0$ that aim to move closer to an undelivered laundry piece, while node $n_5$ represents the class of subproblems $\mathcal{Q}_5$ with width $0$ that aim to pick a laundry piece up when its distance reaches zero.

We conclude the example by observing that the hierarchical policy $\Pi_2^L$ is also meaningful because it is simple, i.e., has few syntactic elements and reflects a hierarchical decomposition that a human might develop if given the task.

We define an execution model for valid hierarchical policies to compute a plan for a given problem. In a nutshell, the execution model uses a stack to track subproblems it currently solves. It traverses through the tree and makes decisions based on the local neighborhood, executing actions only in the leaf nodes. This execution model enables hierarchical policies to make decisions efficiently by focusing on the local neighborhood, reducing the computational overhead compared to flat general policies that evaluate all rules at each step. These local decisions introduce a computational advantage when executing hierarchical policies over flat general policies.

The following section builds on this formal characterization, presenting a method for automatically learning valid hierarchical policies. It leverages planning width and policy sketches to guide a refinement process that starts at the root node, which consists of the dummy sketch rule.

## 5.2 Method

Our method for learning a valid hierarchical policy $\Pi_k$ follows directly from Definition 10: starting from the given target class of problem $\mathcal{Q}$, we find a set of features that distinguishes goals from non-goals and instantiate the root node with the rule as shown in Definition 10.1. Then, we iteratively refine nodes whose class of problems has nonzero width by learning a policy sketch whose width is one less, or $k$ in the particular case of the root node. For each rule in the learned sketch, we instantiate a child node and compute the class of subproblems as shown in Definition 10.2. Notice that the size of problems in the class of subproblems in a child node can be smaller than the size of problems in the class of the parent node, i.e., by pruning states only reachable through goal states. The reduction in problem size translates into smaller combinatorial encodings for learning sketches and, hence, computationally more efficient learning compared to encodings based on problems from $\mathcal{Q}$.

## 5.3 Experiments

We implemented the method for learning hierarchical policies based on our method for learning policy sketches from Chapter 4. We generated numerical features until complexity $15$, increasing the feature pool size.

Table 5.1 shows the key results of learning hierarchical policies $\Pi_k$ for a class of problems $\mathcal{Q}$ for $k$ equal to $0, 1$ and $2$. We report the peak memory usage in GiB (M), the total CPU time in hours for solving the answer set program in parallel on 32 CPU cores (T), the highest number of states for the last iteration of the curriculum learning procedure (#S), the number of selected features (#$\Phi$), and the maximum branching factor (#R).

Learning hierarchical policies failed in the Childsnack domain, where we previously encountered issues in learning a sketch of width zero and where we suspect that the expressivity of the feature language is insufficient. In Delivery, we learned a sketch of width one after increasing the feature complexity to $15$. Still, we failed to learn a general policy directly, i.e., a sketch of width zero, demonstrating a computational gain of splitting classes of problems into simpler classes of subproblems. In Blocks-on, we failed due to the increase in the size of the feature pool. In Miconic, we also succeeded due to slightly improved encoding. Learning a hierarchical policy $\Pi_k$ starting with a larger $k$ often requires fewer states and resources because subclasses of problems contain smaller subproblems that represent a more specific problem aspect. We successfully tested the learned hierarchical policies on much larger problems. Our method was able to learn the hierarchical policy similar to $\Pi_2^{\mathrm{L}}$ from

Figure 5.2 for the equivalent Delivery domain. We also learned meaningful hierarchical policies for several other planning domains, demonstrating that the notion of width can effectively guide the discovery of effective hierarchical structures.

| | $\Pi_0$ | | | | | $\Pi_1$ | | | | | $\Pi_2$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Domain | M | T | #S | #Φ | #R | M | T | #S | #Φ | #R | M | T | #S | #Φ | #R |
| Blocks-clear | 2 | 0.26 | 22 | 2 | 2 | 1 | 0.04 | 5 | 3 | 2 | 2 | 0.10 | 5 | 2 | 2 |
| Blocks-on | – | – | – | – | – | 53 | 20.60 | 22 | 3 | 4 | 38 | 15.16 | 20 | 4 | 2 |
| Delivery | – | – | – | – | – | 12 | 39.18 | 48 | 4 | 2 | 8 | 10.42 | 28 | 4 | 2 |
| Gripper | 3 | 0.43 | 28 | 2 | 4 | 4 | 3.00 | 28 | 3 | 2 | 6 | 1.44 | 34 | 3 | 2 |
| Miconic | 14 | 12.48 | 36 | 3 | 4 | 4 | 0.40 | 18 | 3 | 2 | 6 | 0.31 | 18 | 4 | 2 |
| Reward | 3 | 0.47 | 14 | 2 | 2 | 1 | 0.13 | 8 | 2 | 2 | 3 | 0.84 | 12 | 2 | 2 |
| Spanner | 4 | 1.43 | 19 | 3 | 3 | 10 | 8.75 | 96 | 4 | 2 | 17 | 16.74 | 76 | 4 | 2 |
| Visitall | 11 | 10.22 | 22 | 2 | 2 | 3 | 1.69 | 9 | 2 | 2 | 5 | 4.15 | 11 | 2 | 2 |

**Table 5.1:** Comparison of learning hierarchical policies $\Pi_k$ for a class of problems $\mathcal{Q}$ for $k$ equal to $0, 1$ and $2$, showing the peak memory in GiB (M), the total CPU time in hours for solving the ASP on 32 CPU cores (T), the highest number of states for the last iteration of the curriculum learning procedure (#S), the number of selected features (#Φ), and the maximum branching factor (#R). We denote failures by "–" and report the reasons in the text.

## 5.4 Discussion

We introduced a crisp characterization of hierarchical policies based on planning width and an unsupervised method for learning them. A hierarchical policy is a single-rooted tree whose nodes consist of a single sketch rule. The sketch rules in a valid hierarchical policy induce a decomposition of classes of problems into classes of subproblems with smaller polynomial complexity. More precisely, the sketch rules in the children of each node represent an acyclic sketch whose width is smaller than the width of the class of problems at the parent node. These rules split the class of problems into subclasses of problems with smaller polynomial complexity. For every valid hierarchical policy whose sketch has mutually exclusive feature conditions at each node, a translation exists to a flat general policy, demonstrating a close connection in their expressive power (Drexler, Seipp, and Geffner 2023).

The unsupervised method for learning hierarchical policies iteratively *splits classes of problems into classes of subproblems of smaller bounded width* using our method for learning policy sketches. Our experimental results show that we can learn hierarchical decompositions with a few syntactical elements, allowing us to often prove their correctness for the entire problem class.

Hierarchical policies, similar to policy sketches, are also limited in their ability for reuse, as previously discussed in Chapter 3. The features are embedded directly into the rules, preventing the injection of context-dependent information. Modules that call other modules, as briefly introduced in Chapter 3, are an alternative language for defining hierarchical policies that allow reuse.

Our experimental evaluation shows that similar to learning policy sketches, scalability is a primary concern of our method, preventing its application in complex real-world problems, including the household robot scenarios. We presented several ways to address scalability in Chapter 4. The method of learning hierarchical policies has an advantage in terms of scalability over learning general policies directly, i.e., sketches of width zero. Classes of subproblems in the children of a node can be smaller, as a result of pruning states that are only reachable through the extended goal states, making learning of successive policy sketches increasingly cheaper.

In the next chapter, we present a general class of abstractions based on symmetry reduction to address scalability issues to some extent. These abstractions map symmetric states to the same abstract state. The number of abstract states can be exponentially smaller, effectively reducing the size of several combinatorial encodings in generalized classical planning, including ours.

# 6 Abstractions

## Core Publication of this Chapter

- Dominik Drexler, Simon Ståhlberg, Blai Bonet, and Hector Geffner (2024b). “Symmetries and Expressive Requirements for Learning General Policies”. In: *Proceedings of the Twenty-First International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*. IJCAI Organization.

A major challenge in classical planning is managing the explosion of generated states during the search. Symmetry detection helps to reduce the number of generated states by identifying structurally equivalent states that can be pruned, significantly improving search performance in problems with many structural symmetries (Fox and Long 1999).

Symmetry detection methods typically use graph encodings of the problem that combine information about the state, goal, and actions (Pochter et al. 2011; Sievers et al. 2019). The automorphism group of the graph captures all its symmetries by representing the set of all automorphism of the graph, which are structure-preserving mappings from the graph onto itself. These automorphisms translate into structural symmetries within the state space of a problem. Computing the automorphism group of a graph is harder than

solving the graph isomorphism problem, for which no known polynomial time algorithm exists (Babai 2016).

Research on symmetry detection in classical planning has focused on detecting symmetries derived from grounded problem representations (Pochter et al. 2011) and lifted problem representations (Fox and Long 1999; Sievers et al. 2019). The lifted representation contains first-order information, including predicates, objects, and action schemas, while the grounded representation abstracts first-order details and focuses on propositional facts and ground actions. Methods on the ground representation may result in a heavier symmetry reduction of the generated states (Sievers et al. 2019).

In generalized classical planning, where an objective is to learn general plans for infinite classes of problems, symmetry detection plays an additional role: general plans that solve a state can also solve any symmetric state under specific conditions. These conditions depend on the representation language that is used to represent general plans and the symmetry detection method. Symmetry reduction in generalized planning can reduce the amount of redundant information in the training data, enabling more efficient learning. The computational hardness of computing the automorphism group is less of a concern as the training sets usually contain small problems. Example 6.1 illustrates symmetry pruning on a small problem from the laundry domain.

**Example 11.** Figure 6.1 shows the fully expanded initial state $s_0$ of a problem from the laundry domain. In both, the lifted and grounded problem representation, one can identify that the two laundry pieces are symmetrical. In the lifted problem representation, an automorphism explicitly maps these objects onto each other. In the grounded problem representation, it is implicit in the mapping of grounded propositions onto each other. In both cases, a forward search must only expand either successor $s_2$ or $s_3$, effectively pruning a large portion of the state space.

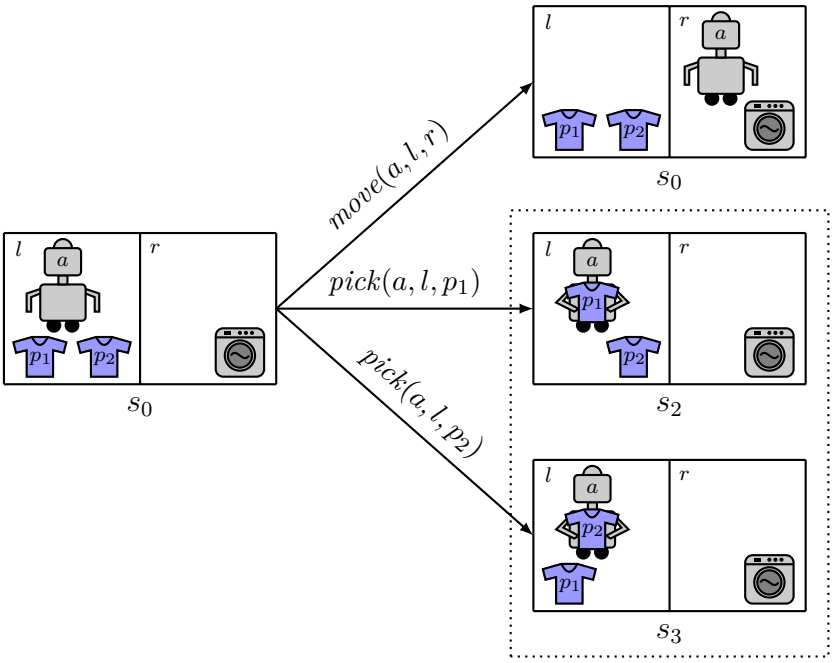



**Figure 6.1:** The illustration shows the fully expanded initial state $s_0$ of a laundry problem that contains a robot $a$, two laundry pieces $p_1$ and $p_2$, and two locations $l$ and $r$. In $s_0$, both laundry pieces are at location $l$ with goal location $r$, and the robot $a$ is at location $l$. The laundry pieces are symmetric; hence, the states $s_2$ and $s_3$ are symmetric, denoted by a dotted rectangle.

In this chapter, we apply symmetry detection to generalized classical planning. We define two states as symmetric if their relational structures are isomorphic. Relational structures are simply an alternative view of states. Isomorphisms between relational structures are functions that rename the object identities but preserve their relationships. We find isomorphisms by encoding states as graphs and running state-of-the-art graph isomorphism solvers. A central contribution of our approach is the exploitation of a well-known result that two isomorphic relational structures satisfy the same first-order logic sentences, allowing us to ignore lifted action schemas completely. Consequently, we bypass the complexity of considering the actions in the graph encodings. Hence, our graph encodings depend purely on the atoms in a state, resulting in smaller graph encodings. Using our method, we can generate abstractions of the state spaces. Experimental results show a reduction in training data size by more than two orders of magnitude across several domains, along with improvements in learning general policies from these abstractions compared to the state spaces as done previously (Francès, Bonet, et al. 2021).

## 6.1 Theoretical Framework

Planning states are relational structures. A finite relational structure consists of a finite set of objects (the universe) and the predicate symbols of the planning domain (the signature). The interpretation captures the relationships between the objects in the state. We define two planning states as equivalent if and only if their corresponding relational structures are isomorphic. Intuitively speaking, two equivalent states only differ in the naming of the objects, not the relationships between the objects. We cast the problem of finding an isomorphism between relational structures as graph isomorphism on undirected vertex-colored graphs. Although graph isomorphism is not known to be solvable in polynomial time, the best-known algorithm runs in quasi-polynomial time (Babai 2016). This section is organized as follows. First, we describe the graph encoding used for detecting equivalent states. Second, we describe how to obtain abstractions for generalized planning based on state equivalence. Last, we lift the notion of solvability for general policies in the context of our abstractions by introducing a structural restriction called uniformity.

### 6.1.1 Graph Encoding

Our graph encoding builds on top of existing graph encodings (Ståhlberg, Bonet, et al. 2022a; Chen, Trevizan, et al. 2024; Chen, Thiébaux, et al. 2024; Horcík and Sír 2024). Those graph encodings aim not to test state equivalence but to use graph neural networks. Each of the existing encodings has one of the following two limitations, preventing them from being used for our purpose of state equivalence testing: 1) there is a loss of information, in the sense that the state cannot be decoded from the graph, or 2) the graph encodings are not undirected vertex-colored graphs, which modern graph isomorphism libraries such as `nauty` (McKay and Piperno 2014) requires as input.

Our graph encoding addresses these limitations. For a given planning state, our undirected vertex-colored graph has an object vertex for each object in the state and a positional argument vertex for each occurrence of an object in a ground atom that is true in the state. Object vertices share the same unique color, allowing them to be mapped onto one another. Positional argument vertices have a unique color, given by the predicate and the position, to ensure that only semantically matching vertices are mapped onto one another. Edges connect object vertices to their respective positional argument vertices and neighboring positional argument vertices. Our encoding works for predicates of arbitrary arity and guarantees that two relational structures are isomorphic if and only if their object graphs are isomorphic. We say that two states are equivalent if and only if their corresponding relational structures are isomorphic. The equivalences ensure that there is no loss of information.

**Example 12.** Figure 6.2 shows the graph of the state $s_2$ or $s_3$ from Figure 6.1, depending on the labeling of the laundry pieces. It contains five object vertices, one for every object in the state. The sub-graph consisting of the object vertex $\langle l \rangle$, positional argument vertex $\langle loc, 1, \langle l \rangle \rangle$, and their connecting edge uniquely represent the unary ground atom $loc(l)$. For higher arity relations, the structure above is generalized. The sub-graph consisting of the object vertices $\langle a \rangle$ and $\langle l \rangle$, the positional argument vertices $\langle at, 1, \langle a, l \rangle \rangle$ and $\langle at, 2, \langle a, l \rangle \rangle$, and their connecting edges uniquely represent the binary atom $at(a, l)$. In other words, the ordering of the arguments can be uniquely decoded from the colors that depend on the position of the argument itself.

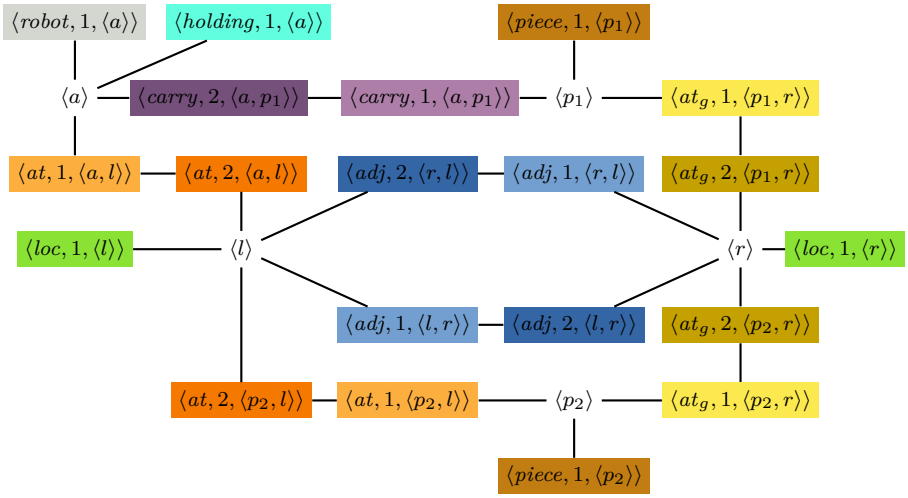



**Figure 6.2:** The illustration shows the graph of the state $s_2$ or $s_3$ from Figure 6.1. The object vertices are uncolored. The positional argument vertices are colored depending on their position and predicate. The static goal predicate $at_g$ represents the goal location of the pieces, e.g., $at_g(p_1, r)$.

### 6.1.2 Equivalence-based Abstractions

Equivalence-based abstractions are abstractions of the state space induced by a problem where two equivalent states are mapped to the same abstract state and two transitions with equivalent sources, respectively, target states, are mapped to the same abstract transition. A strength of our approach is that it is easily applicable across different problems, detecting completely equivalent problems or equivalent states across different problems. The following example illustrates the theoretical gain, as the number of states in the abstraction can be exponentially smaller.

**Example 13.** Figure 6.3 shows an abstraction for a laundry problem with $n$ laundry pieces and two locations $l$ and $r$. There are $4n + 2 = 2[(n + 1) + n]$ non-isomorphic states: for each of the two possible positions of the robot, there are $n + 1$ states where no laundry piece is being held and $n$ states where a laundry piece is being held. In comparison, the state space contains an exponential amount of states, e.g., each laundry piece can be in either room for a total of $2^{n+1}$ states.

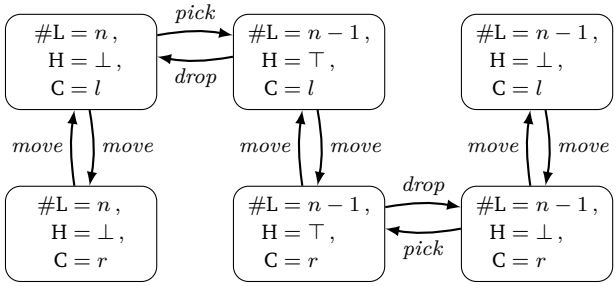



**Figure 6.3:** The illustration shows a fragment of the abstraction for a problem from $\mathcal{Q}_L$ with $n$ laundry pieces and two locations $l$ and $r$. The number of pieces at $l$ (#L), whether one is held (H), and the robot's location (C) identifies each equivalence class. The abstraction contains $4n + 2$ states (see text).

While the solvability of general policies is typically defined on state spaces (Francès, Bonet, et al. 2021), applying solvability to our abstractions requires introducing the concept of uniformity of general policies.

### 6.1.3 Uniform General Policies

Uniformity imposes a structural restriction on general policies,[3] requiring that they cannot distinguish transitions between equivalent states. In other words, uniform general policies treat structurally equivalent state transitions similarly, ensuring consistency in decision-making across equivalent states. The uniformity constraint allows the notion of solvability to be lifted from state spaces to equivalence-based abstractions. In this context, a uniform general policy is considered solvable if all maximal state trajectories for an equivalence class lead to a goal state. This raises the question of whether previous notions of general policies are uniform for our abstractions.

Formal languages that are isomorphism-invariant over finite relational structures cannot distinguish between isomorphic relational structures. Descriptive

[3]The restriction can be placed other types of general plans, e.g., sketches.

languages are usually isomorphism-invariant by design. Description logics (Baader et al. 2003), graph neural networks (Scarselli et al. 2009), and first-order logic with counting quantifiers (Immerman 1998) are examples of isomorphism invariant languages. Hence, general policies defined over such isomorphism-invariant feature languages are uniform over our abstractions.

Our work can be seen as a specific instantiation of the abstract structures framework for lifted problem representations while ignoring action schemas by Sievers et al. (2019). Their specific instantiation and work on the grounded problem representation (Pochter et al. 2011) allowed two truly non-isomorphic states to be considered equivalent. This can occur when an automorphism maps two distinct predicates onto each other. As a result, general policies based on isomorphism invariant languages, such as the ones mentioned above, may distinguish states and transitions in abstractions based on these stronger notions of symmetry, effectively breaking uniformity over such abstractions.

## 6.2 Experiments

We ran experiments to see the reduction in training data when using abstractions and how they affect the efficiency of learning general policies. We use the method for learning sketches from Chapter 4 to learn general policies, i.e., sketches of width zero. Since general policies are uniform for our equivalence-based abstractions, it suffices to pick a single representative, effectively reducing the size of the combinatorial encoding. The training set consists of planning domains from the International Planning Competition. We sampled sets of small problems using PDDL generators (Seipp et al. 2022).

Table 6.1 shows the results for learning general policies with and without equivalence-based reductions. We report the peak memory usage (M) in GiB, the wall-clock time in seconds for learning (T), the total number of states in the training set (#S) and the reduced training set (#S$/\!\sim_{iso}$), and ratios for the speedup in time and the reduction in the number of states.

The table shows a drastic reduction in the number of states in the training set, ranging from factor $1.18$ in Visitall to $355.12$ in Blocks4ops-clear. In principle, this should result in a significant decrease in peak memory. However, we are using the implementation for learning sketches, which requires the entire state space to be held in memory because it uses the notion of width that depends on the structure of the state space. We expect a decrease in more specialized implementations. There is a slowdown in some simple domains, such as Ferry or Blocks4ops-on, where the absolute time is still relatively small. In Blocks3ops, the speedup of $2.57$ is significant; after closer inspection, the curriculum learning procedure selects the same problem but with fewer states.

In summary, equivalence-based abstraction can reduce the necessary training data for learning general policies and improve learning efficiency.

| | without reduction | | | with reduction | | | | |
|---|---|---|---|---|---|---|---|---|
| Domain | M | T | #S | M | T | Speedup | $\#S/\sim_{iso}$ | Factor |
| Blocks3ops | **9** | 28,884 | 145,680 | 11 | **11,233** | **2.57** | **4,901** | 29.72 |
| Blocks4ops-clear | 1 | 8 | 30,540 | 1 | **6** | **1.33** | **86** | 355.12 |
| Blocks4ops-on | 3 | 265 | 30,540 | **2** | **228** | 0.47 | **249** | 122.65 |
| Delivery | 3 | 534 | 411,720 | **2** | **325** | **1.64** | **3,346** | 123.05 |
| Ferry | 1 | **69** | 8,430 | 1 | 91 | 0.76 | **265** | 31.81 |
| Gripper | 1 | **5** | 1,084 | 1 | 6 | 0.83 | **90** | 12.04 |
| Miconic | 1 | **38** | 32,400 | 1 | 58 | 0.66 | **12,339** | 2.63 |
| Reward | 1 | 20 | 13,394 | 1 | **14** | **1.43** | **7,026** | 1.91 |
| Spanner | 1 | **7** | 9,291 | 1 | 8 | 0.88 | **283** | 32.83 |
| Visitall | **2** | **77** | 476,766 | 3 | 95 | 0.98 | **402,880** | 1.18 |

**Table 6.1:** Comparison of learning general policies with equivalence-based reductions and without, showing the memory usage in GiB (M), the wall-clock time in seconds (T), the total number of states in the training set (#S) and the reduced training set ($\#S/\sim_{iso}$), and ratios for the speedup in time and the reduction in the number of states. We use boldface to highlight the winner in a pairwise comparison, i.e., the one that needed strictly fewer resources.

## 6.3 Discussion

We introduced a class of abstractions for generalized planning based on state equivalence in terms of the existence of an isomorphism between their relational structures that can result in an exponential reduction of redundancy in the training data. We introduced a structural restriction for general policies, called uniformity, that allows the solvability of general policies to be lifted from state spaces to our abstractions. General policies over isomorphism-invariant feature language are naturally uniform over our abstractions because isomorphism-invariant languages cannot distinguish between isomorphic relational structures (Immerman 1998). Our abstractions are applicable to and other works in generalized classical planning that use description logics (e.g., Martín and Geffner 2004; Fern et al. 2004; Yoon et al. 2008; Francès, Corrêa, et al. 2019; Francès, Bonet, et al. 2021; Ståhlberg, Francès, et al. 2021; Drexler, Seipp, and Geffner 2022; Drexler, Seipp, and Geffner 2023), or graph neural networks (e.g., Ståhlberg, Bonet, et al. 2022a; Aichmüller and Geffner 2024).

An interesting question for future work is whether or under which conditions we can learn general policies for abstractions based on stronger notions of symmetry. For example, we can prune statically irrelevant information from

states before isomorphism testing, such as objects, ground atoms, or goal atoms that are static and irrelevant in a state. While this does not improve the symmetry reduction on a per-problem level, it increases the amount of symmetry reduction among collections of problems. States with statically irrelevant information may exist if the state space contains more than one strongly connected component. Dominance pruning (Torralba and Hoffmann 2015; Torralba 2018) could increase the number of strongly connected components by removing unnecessary state transitions from the state space.

# 7 Expressive Learning Requirements

## Core Publication of this Chapter

- Dominik Drexler, Simon Ståhlberg, Blai Bonet, and Hector Geffner (2024b). "Symmetries and Expressive Requirements for Learning General Policies". In: *Proceedings of the Twenty-First International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*. IJCAI Organization.

In the previous chapter, we presented equivalence-based abstractions that map equivalent states, i.e., whose relational structures are isomorphic, to the same abstract state. Isomorphism-invariant languages cannot distinguish between two equivalent states, which implies consistent decision-making across equivalent state transitions or state pairs. Examples of isomorphism-invariant languages are description logics (Baader et al. 2003), graph neural networks (Scarselli et al. 2009), and first-order logic with counting quantifiers (Immerman 1998). Therefore, the policy sketches language is isomorphism-invariant if the feature language also is because policy sketches operate directly on top of the feature valuations.

In this chapter, we take a different perspective on feature languages by identifying sufficiently expressive languages to distinguish between all non-equivalent

states in planning benchmark sets. Ideally, its expressive power should be neither excessive nor insufficient, as overly expressive power incurs high computational costs during the learning and testing of general plans over such languages. We restrict ourselves to using benchmark sets because testing whether a language has sufficient expressive power for an infinite class of problems $\mathcal{Q}$ is challenging, as it requires a logical characterization of $\mathcal{Q}$.

Before going into more detail, we give a positive answer: three-variable first-order logic with counting quantifiers $\mathsf{C}_3$ suffices for all considered benchmark sets. Moreover, its two-variable fragment $\mathsf{C}_2$ often suffices, showing that the expressive power of a specific family of graph neural networks also suffices.

Our approach to obtaining these results is simple and effective. It uses two important theoretical results from the literature:

**R1** The $k$-dimensional Weisfeiler-Leman algorithm, also called $k$-WL, has the same expressive power as $k+1$-variable first-order logic with counting quantifiers $\mathsf{C}_{k+1}$ (Cai et al. 1992).

**R2** The family of aggregate-combine-readout graph neural networks, also called ACR-GNNs, can capture any two-variable first-order logic with counting quantifiers $\mathsf{C}_2$ formula (Barceló et al. 2020; Grohe 2021).[4]

Our method runs the $k$-WL algorithm (Weisfeiler and Leman 1968) to find expressivity conflicts among all pairs of states in the benchmark set. Then, we use the abovementioned results from the literature to deduce whether the expressive power of other languages suffices.

The chapter is organized as follows: First, we describe our method. Second, we present experimental results, using our method to exhaustively check for expressivity conflicts in several planning benchmark sets from the International Planning Competition. Last, we summarize our results while discussing related work, present ideas for future research, and conclude.

## 7.1 Method

Building on our graph encoding of states (relational structures) from Chapter 6, we develop a method to find languages whose expressive power is sufficient for a benchmark set $\mathcal{Q}_\mathcal{T}$ of planning problems. Recall that our encoding preserves state information, ensuring that two graphs are isomorphic if and only if their relational structures are isomorphic.

[4] ACR-GNNs are traditional graph neural networks where global aggregate information is taken into account.

The *expressive power* of a feature language $\mathcal{L}$ captures its ability to distinguish graphs, which, in our case, precisely encode states (Immerman 1998). For example, in problems from the class $\mathcal{Q}_L$ over the laundry domain, the first-order sentence $\phi \equiv \exists x.holding(x)$ distinguishes any state where the robot holds a laundry piece from any state where it does not hold a piece. Failure to distinguish non-isomorphic states may disqualify the assignment of different behaviors to those states. Hence, a feature language $\mathcal{L}$ has sufficient expressivity power if and only if it can distinguish between all pairs of non-isomorphic states in problems from a class of problems $\mathcal{Q}_\mathcal{T}$, and we may say $\mathcal{L}$ has sufficient expressivity.

Our method analyzes the required expressive power relative to $k$-variable first-order logic with counting quantifiers ($\mathsf{C}_k$). More precisely, we are looking for the smallest $k$ such that $\mathsf{C}_k$ has sufficient expressivity. We use result **R1** from the literature to use the $k$-WL algorithm for finding such a smallest $k$.

The $k$-WL algorithm (Weisfeiler and Leman 1968) is a graph coloring algorithm for approximate graph isomorphism testing. It takes as input a graph and computes a stable coloring. The algorithm runs in time polynomial in the graph size if $k$ is fixed. If the stable colorings of two graphs are identical, then $k$-WL cannot distinguish them. Moreover, if the two graphs are non-isomorphic, then $k$-WL falsely identifies them as isomorphic. Conflicts arise when $k$-WL falsely identifies two non-isomorphic states as isomorphic, resulting in an expressivity conflict or simply an **E-conflict**. Moreover, if both states share the same goal distance $V^*$, they form a **V-conflict**. V-conflicts are less problematic for learning optimal goal distances as they do not affect the accuracy of distance predictions. However, E-conflicts are generally problematic for learning general plans, as the algorithm treats different states that represent different problem aspects as identical, potentially disqualifying the assignment of different behaviors, i.e., actions, to those states.

Hence, if $k$-WL does not find a conflict on any pair of non-equivalent states in $\mathcal{Q}_\mathcal{T}$, then its expressivity suffices, and we can apply result **R1** from the literature to deduce that the expressivity of $\mathsf{C}_{k+1}$ must also suffice. In the special case of $k$ equal to $1$, we can apply result **R2** from the literature on top to deduce that the expressivity of the family of ACR-GNN must also suffice.

## 7.2 Experiments

We ran experiments to find conflicts in benchmark sets of problems over planning domains from the International Planning Competition. The benchmark sets contain small problems that we sampled using PDDL generators (Seipp et al. 2022). We first computed the abstractions presented in Chapter 6 to par-

| | | | | 1-WL | | 2-FWL | |
|---|---|---|---|---|---|---|---|
| Domain | $\#\mathcal{Q}$ | #S | $\#\mathrm{S}/\!\sim_{iso}$ | #E | #V | #E | #V |
| Barman | 510 | 115 M | 38 M | 1,326 | 537 | 0 | 0 |
| Blocks3ops | 600 | 146 K | 133 K | 50 | 20 | 0 | 0 |
| Blocks4ops | 600 | 122 K | 110 K | 54 | 27 | 0 | 0 |
| Blocks4ops-clear | 120 | 31 K | 3 K | 0 | 0 | 0 | 0 |
| Blocks4ops-on | 150 | 31 K | 8 K | 0 | 0 | 0 | 0 |
| Childsnack | 30 | 58 K | 5 K | 0 | 0 | 0 | 0 |
| Delivery | 540 | 412 K | 62 K | 0 | 0 | 0 | 0 |
| Ferry | 180 | 8 K | 4 K | 36 | 36 | 0 | 0 |
| Grid | 1,799 | 438 K | 370 K | 42 | 38 | 0 | 0 |
| Gripper | 5 | 1 K | 90 | 0 | 0 | 0 | 0 |
| Hiking | 720 | 44 M | 5 M | 0 | 0 | 0 | 0 |
| Logistics | 720 | 69 K | 38 K | 131 | 131 | 0 | 0 |
| Miconic | 360 | 32 K | 22 K | 0 | 0 | 0 | 0 |
| Reward | 240 | 14 K | 11 K | 0 | 0 | 0 | 0 |
| Rovers | 514 | 39 M | 34 M | 0 | 0 | 0 | 0 |
| Satellite | 960 | 14 M | 8 M | 5,304 | 4,226 | 0 | 0 |
| Spanner | 270 | 9 K | 4 K | 0 | 0 | 0 | 0 |
| Visitall | 660 | 3 M | 2 M | 0 | 0 | 0 | 0 |

**Table 7.1:** Overview of the number of conflicts detected in our benchmark sets, showing the number of problems used in our experiments ($\#\mathcal{Q}$), the number of total states (#S), the number of equivalence classes where states of each problem are partitioned individually ($\#\mathrm{S}/\!\sim_{iso}$), the number of expressivity conflicts caused by 1-WL and 2-FWL (#E), and the number of E-conflicts where states have the same optimal goal distance $V^*$ (#V).

tition the states into their equivalence classes. We ran color refinement (1-WL) and the 2-dimensional (Folklore) Weisfeiler-Leman algorithm (2-FWL) to find conflicts among pairs of representative states from each equivalence class in each problem. We set a time limit of 6 hours per problem. We accumulated the conflicts of successful runs with problems that contain at most 1M states.

Table 7.1 shows the total number of problems ($\#\mathcal{Q}$), the total number of states in the state spaces (#S), the total number of state equivalence classes ($\#\mathrm{S}/\!\sim_{iso}$) where the states of each problem are partitioned individually, and the number of E-conflicts (#E) and V-conflicts (#V) caused by 1-WL and 2-FWL.

The number of states ranges from 1K in Gripper to 115M in Barman, showing that our method can handle large benchmark sets. We can observe that the 1-WL algorithm finds no conflicts in 11 out of all 18 domains, and in the remaining 7 domains, there are both types of conflicts. Most importantly, all

conflicts disappear when using the more expressive 2-FWL algorithm. Using the equivalences from the literature, we can summarize the main results:

1. 1-WL, $C_2$, and ACR-GNNs have sufficient expressivity for 11 benchmark sets, and
2. 2-FWL, and $C_3$ have sufficient expressivity for *all* benchmark sets.

While these findings do not give strong guarantees for successfully learning general plans, our findings often show empirical alignment with previous work on learning general policies using graph neural networks (Ståhlberg, Bonet, et al. 2023). For example, we do not find any 1-WL conflicts in domains such as Delivery, Gripper, Miconic, and Visitall where Ståhlberg, Bonet, et al. (2023) successfully learn a nearly perfect general policy. Moreover, we find 1-WL conflicts in domains such as Logistics and Grid, where the same approach of the same authors fails. However, we do not find conflicts in Rovers where the same authors claim that it requires $C_3$ expressivity because of the importance of a ternary predicate, indicating that our benchmark set does not cover all aspects of the domain (Ståhlberg, Bonet, et al. 2022a). In recent work (Ståhlberg, Bonet, et al. 2025), the same authors show that more expressive families of graph neural networks can successfully learn nearly perfect general policies for planning domains where the two-variable fragment is insufficient.

## 7.3 Discussion

We introduced a method for testing whether $k$-WL, $C_k$, and ACR-GNNs have sufficient expressivity for several planning benchmark sets. Our method leverages the close relationship between those languages from the literature (Cai et al. 1992; Barceló et al. 2020; Grohe 2021). Our experimental results show that 2-FWL and $C_3$ always suffice for our benchmark sets, a computationally manageable level of expressive power. Furthermore, 1-WL, $C_2$, and ACR-GNNs often suffices (Barceló et al. 2020). While our method does not give strong guarantees for learning general plans, our findings often show empirical alignment with previous work on learning general policies using graph neural networks (Ståhlberg, Bonet, et al. 2023).

Our method complements work done in parallel that evaluates whether several families of graph neural networks have sufficient expressivity for planning benchmark sets (Horcík and Sír 2024). While their approach uses randomly initialized graph neural networks from different families to find conflicts in distinguishing non-isomorphic states, our method precisely assesses whether the expressive power of the specific ACR-GNN family suffices.

Our method could enhance previous work on learning heuristic functions over $k$-WL features in planning (Chen, Trevizan, et al. 2024). These features are colorings of earlier iterations $j$ of the $k$-WL that are not necessarily stable colorings and, hence, less expressive in distinguishing non-isomorphic states. These heuristics do not aim to represent general plans but to balance heuristic informativity with evaluation speed effectively. Our work can find suitable values of $j$ and $k$, automatically adapting the method to more complex planning domains. More specifically, for a given training set, one can find values for $j$ and $k$ such that the language of $k$-WL colorings at iteration $j$ has sufficient expressivity for a benchmark set.

In the context of this work, a promising direction for future work is getting more fine-grained control over the construction of explicit feature pools over description logic grammars. This includes developing methods to test whether the expressive power of a specific description logic suffices. For example, the expressive power of description logic $\mathcal{ALCQ}$ is equivalent to $\mathrm{GC}_2$, which in turn is less expressive than $\mathrm{C}_2$ (Baader et al. 2003). Creating explicit feature pools with sufficient expressivity for the training set can improve combinatorial learning methods by reducing the number of overly complex features.

In summary, our method can assess whether the expressive power of several languages suffices for a given benchmark set, helping systems dynamically adapt language expressivity for learning to plan.

Building on the insights of this chapter on analyzing the expressivity requirements and the previous chapter on equivalence-based abstractions, we suggest that *further analyzing collections of problems and exploiting the obtained information is a key challenge for the effective integration of learning and planning*. An effective integration requires systems to process collections of problems efficiently. Furthermore, since general plans are highly informative, an exponential preprocessing step, called grounding (Helmert 2009), is often unnecessary. The next chapter presents a library tailored to address these tasks while supporting expressive planning language features. More specifically, it works with collections of problems and avoids grounding by working directly on first-order problem representations.

# 8 Lifted Planning With Expressive Extensions

A planner is an algorithm that takes as input a problem and aims to find a plan. The planning domain definition language (PDDL) is a standardized language for representing input problems (McDermott et al. 1998). Researchers incrementally extended the language features for more accurately modeling complex real-world problems such as support for numeric state variables (Fox and Long 2003), derived predicates, which are relations derived from basic predicates (Edelkamp and Hoffmann 2004). Explicit search planners construct a search graph where nodes represent states and edges represent state transitions caused by applying actions. The two central operations are the *computation of the applicable actions* in a state and *computing the successor state* when applying an action in a state. The primary planning paradigms are *lifted planning* and *grounded planning*. Lifted planners operate on the lifted problem representation and determine the applicable ground actions in a state online using combinatorial approaches such as constraint satisfaction (Francès 2017), conjunctive queries (Corrêa, Pommerening, et al. 2020), or maximum clique enumeration (Ståhlberg 2023). In the past three decades, research has shifted focus from the lifted to the grounded planning paradigm (Corrêa and Giacomo 2024). This trend stems from grounded planners' effectiveness and computational efficiency on many benchmark sets. Grounded planners first generate a ground problem representation by substituting variables in action schemas with all possible object combinations. This grounding process may

result in a combinatorial explosion in the number of ground atoms, ground actions, and size of the tree data structure for efficiently retrieving the applicable ground actions in a state (Helmert 2006). Grounding becomes infeasible with high-arity predicates, complex dependencies, or a large set of objects.

Heuristic search (Hart et al. 1968) is one of the most widely used explicit search techniques to solve planning problems. Heuristic search techniques often solve a relaxed version of the problem, i.e., tractable approximation, and use it to guide the search in the actual problem (Bonet and Geffner 2001). Since the relaxed version is a simplified, heuristic values often deviate from true goal distances, leading to more state expansions. Grounded heuristic search planners achieve the best results by effectively balancing heuristic informativity with evaluation speed (e.g., Hoffmann and Nebel 2001). Conversely, lifted planners require stronger heuristic informativity to balance the higher costs associated with generating the applicable ground actions in a state by expanding fewer states. High informativity is central in generalized planning, where general plans ensure a polynomially bounded number of state expansions. Consequently, general plans can balance the higher costs of generating the applicable ground actions in a state, making lifted planning a powerful paradigm for generalized planning (Drexler, Seipp, and Geffner 2024).

As lifted planning has been sidelined for many years, the available planners to solve planning problems with this paradigm are insufficient. The primary reason is that several expressive PDDL language features, such as conditional effects and derived predicates, are not supported. However, these language features cannot be compiled away without an exponential blowup in the problem size (Gazen and Knoblock 1997; Nebel 2000; Thiébaux et al. 2005). Hence, it is beneficial for planners to support these features natively.

To bridge this gap, we present a new version of Mimir (Ståhlberg 2023), a C++ planning library that seamlessly integrates grounded and lifted planning where we added native support of the abovementioned expressive PDDL features.[5] To the best of our knowledge, no other lifted PDDL-based planner supports these expressive extensions. We evaluate Mimir against two state-of-the-art planning systems on three benchmark sets with uninformed (blind) A$^*$ search (Hart et al. 1968): Fast Downward (Helmert 2006), a specialized grounded planner and Powerlifted (Corrêa, Pommerening, et al. 2020), a specialized lifted planner. Our experimental results show that Mimir competes with state-of-the-art and performs favorably in lifted planning. In addition, while Mimir is fully written in C++, it provides Python bindings that support the development of heuristics directly in Python. Furthermore, it can process multiple problems in memory, making it a versatile tool for addressing generalized planning.

[5] Mimir is available at `https://github.com/simon-stahlberg/mimir`.

This chapter is organized as follows. First, we briefly discuss the expressive PDDL language features. Second, we present the experimental evaluation, including the objective, the setup, the configurations, the benchmarks, and the results. Last, we summarize, conclude, and present ideas for future work.

# 8.1 Expressive Language Extensions

## 8.1.1 Conditional Effects

Conditional effects are an action effect structure that triggers specific effects depending on the current state. For example, consider an action to compile a LaTeX file into a PDF file where the choice of several compile options affects the outcome. A plan length preserving representation without conditional effects requires an exponential number of actions, i.e., multiplying out the conditional effects, resulting in an action without conditional for each combination (Nebel 2000). Conditional effects are also helpful in modeling state-dependent action costs that also cannot concisely be compiled away (Speck et al. 2021) and are crucial for accurately modeling complex real-world problems (Geißer 2018). For example, the action cost for driving a truck from one location to another might depend on the load in the truck, the traffic on the road, and the gas price.

## 8.1.2 Derived Predicates

Derived predicates are predicates whose interpretation, i.e., ground state atoms, are derived from the ground state atoms over the *basic* predicates through axioms. An axiom has a precondition and a single ground atom over a derived predicate in its effect. Applying an axiom in a state does not result in a state change but instead adds additional ground atoms to the state. Axioms are evaluated using fixed-point iteration until no more ground atoms can be derived. Ground atoms over basic predicates may appear in action effects. However, ground atoms over derived predicates may only appear in conditions such as action preconditions, conditional effect preconditions, or goal conditions (Thiébaux et al. 2005).

Derived predicates are helpful because they can capture complex action preconditions, conditional effect conditions, or goal conditions. For instance, we can compile quantified conditions into axioms to remove quantifiers and significantly simplify the problem structure (Helmert 2009). Derived predicates and axioms can easily express transitive closure, which is less intuitive with actions that additionally result in state changes (Thiébaux et al. 2005).

## 8.2 Experiments

We compare Mimir against two state-of-the-art planning systems, Fast Downward (Helmert 2006) and Powerlifted (Corrêa, Pommerening, et al. 2020), on three benchmark sets. The outline is as follows: First, we describe the objectives of our experimental evaluation. Second, we describe the setup, including resource limits and performance metrics. Third, we describe the technical details of each planner configuration. Fourth, we discuss the benchmark sets. Last, we present the gathered data and interpret it.

### 8.2.1 Objective

Our experimental evaluation aims to understand the effectiveness of the internal data structures and algorithms for searching for a plan by iteratively expanding states to find a goal state of a given planning problem. Performance depends primarily on the planner's ability to compactly store states, search nodes, ground actions, and ground axioms, and efficiently generate the applicable actions for a given state.

Our evaluation uses an uninformed $A^*$ search with a blind heuristic. While using the same heuristics could also ensure fairness, the choice of a blind heuristic simplifies the comparison by removing heuristic guidance entirely. Hence, our comparison focuses on the raw computational efficiency of the planners' internal data structures and algorithms.

### 8.2.2 Common Setup

We use the following resource constraint to test the planners' ability to operate efficiently within both time and memory limitations:

- **Memory limit**: 8 GB
- **Time limit**: 30 minutes

We use the following performance metrics to compare all planners:

- **Coverage**: the total number of solved problems
- **Total time**: the geometric mean over the total time in milliseconds needed to solve all problems.
- **Search time**: the geometric mean over the search time in milliseconds needed to solve all problems.

Search time and total time consider problems solved by all planners.

### 8.2.3 Planner configurations

We consider the following planner configurations in our evaluation.

- **Fast-Downward** is a grounded planner (Helmert 2006). It uses a dense finite-domain state representation, i.e., the value assigned to a variable in the state represents one of several mutually exclusive atoms or potentially none of those. A preprocessing step called mutex analysis detects these mutually exclusive variables and can result in a very compact state representation (Helmert 2009). We turn off the detection of irrelevant variables to obtain comparable search behavior.
- **Powerlifted** is a lifted planner that uses conjunctive queries for computing the applicable actions in a state (Corrêa, Pommerening, et al. 2020). It uses a sparse state representation, i.e., it only contains ground atoms that are true in the state, which can be more compact than a finite-domain representation. However, the contrary usually holds if the number of atoms in a ground problem representation is small. Powerlifted does not support negative preconditions, derived predicates, conditional effects, and quantified preconditions.
- **Mimir-lifted** and **Mimir-grounded** are lifted, respectively grounded, planners. The lifted planner uses k-clique enumeration to compute applicable actions in a state (Ståhlberg 2023). It uses a sparse state representation. In addition to Powerlifted, Mimir compresses the state to the smallest required bit-width to represent all atoms in the state.

### 8.2.4 Benchmarks

In our evaluation, we consider the following benchmark sets, which cover a diverse set of challenges occurring in planning problems: easy-to-ground benchmarks that use the simple STRIPS formalism, hard-to-ground benchmarks in the same formalism, and benchmarks over more expressive formalisms that allow for more compact encodings.

- **STRIPS**: The STRIPS benchmark set consists of easy-to-ground planning domains from the International Planning Competition that use simple formal STRIPS planning.
- **HTG**: The HTG benchmark set consists of hard-to-ground planning domains that use the simple STRIPS planning formalism where the ground problem representation contains infeasibly large numbers of ground atoms and/or actions.

- **ADL**: The ADL benchmark set consists of easy-to-ground planning domains from the International Planning Competition that use more PDDL language features such as negative preconditions, derived predicates, disjunctive preconditions, quantified-preconditions, and conditional effects.

### 8.2.5 Overview of the Results

| | Planner | Coverage | Total Time [ms] | Search Time [ms] |
|---|---|---|---|---|
| STRIPS | Fast-Downward | **659** | 1273 | **261** |
| | Mimir-grounded | 611 | **756** | 313 |
| | Powerlifted | 539 | 6958 | 6050 |
| | Mimir-lifted | 598 | 2574 | 1692 |
| HTG | Fast-Downward | 108 | 3132 | 102 |
| | Mimir-grounded | 88 | 3270 | **61** |
| | Powerlifted | 135 | **344** | 217 |
| | Mimir-lifted | **137** | 522 | 270 |
| ADL | Fast-Downward | **365** | 2394 | **399** |
| | Mimir-grounded | 336 | **1334** | 599 |
| | Powerlifted | – | – | – |
| | Mimir-lifted | 299 | 5722 | 4544 |

**Table 8.1:** Comparison of the planner configurations on three benchmark sets: hard-to-ground (HTG), optimal STRIPS (STRIPS), and optimal ADL (ADL) of the International Planning Competition (IPC), showing the total number of solved problems (Coverage), the geometric mean of the total time in milliseconds (Total Time), and the search time in milliseconds (Search Time). We denote configurations with insufficient PDDL language support on a benchmark set by "–". We highlight the best configuration with boldface.

Table 8.1 shows the results of running all planner configurations on all benchmark sets. On easy-to-ground benchmarks, Fast-Downward has the lowest search time and overall highest coverage on the IPC STRIPS and ADL benchmarks. The preprocessing step is costly, resulting in a higher total time than Mimir-grounded. Fast-Downward and Mimir-grounded achieve comparable search time scores. The higher coverage suggests that Fast Downward's state representation is more compact, allowing it to generate more states. Powerlifted has the best runtime score on hard-to-ground benchmarks. However, Powerlifted achieves significantly worse scores in all three performance metrics in easy-to-ground benchmarks. The comparison to lifted planners suggest that Fast-Downward's and Mimir's grounding step costs a comparable and significant amount of time on hard-to-ground benchmarks. Mimir's costs are

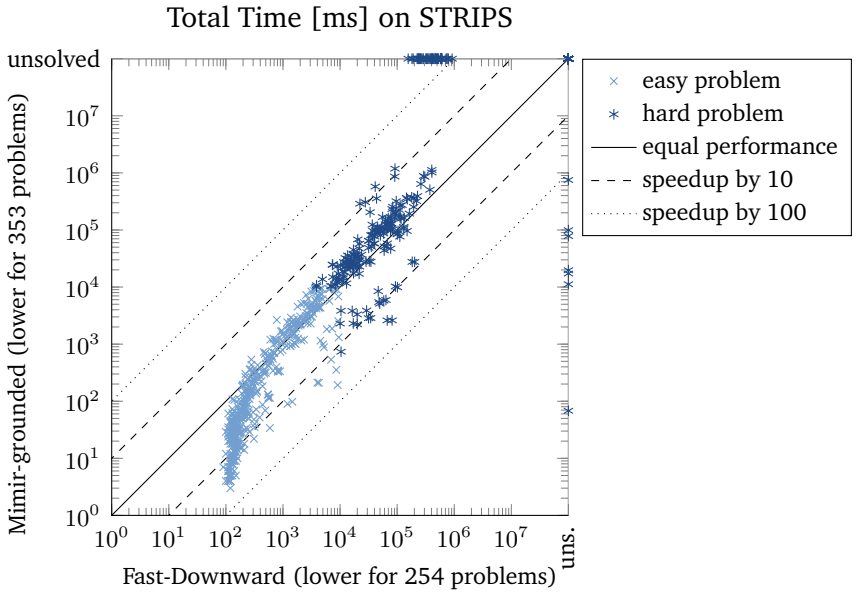



**Figure 8.1:** A pairwise comparison between Mimir-grounded and Fast Downward on the STRIPS benchmark, comparing the total time in milliseconds required to find a plan. We label a problem "easy" if both configurations require less than 10000 milliseconds, and otherwise, we label it "hard".

even higher because the tree structure used to retrieve applicable actions is based on a less compact state representation than Fast-Downward's, based on a compact finite-domain state representation (FDR). While the overall comparison in Table 8.1 highlights major trends, a deeper dive into pairwise comparisons helps explain the observed differences in total time.

### 8.2.6 Grounded Comparison

Figure 8.1 shows the pairwise comparison between Fast-Downward and Mimir-grounded on the STRIPS benchmark set. We label a problem "easy" if both configurations require less than 10000 milliseconds, and otherwise, we label it "hard". On easy problems, Mimir-grounded often finds a plan quicker because it does not perform the mutex analysis preprocessing step, which Fast-Downward uses to derive a compact finite-domain state representation. On hard problems, Fast-Downward is often faster by a small constant factor. Fast-Downward solves significantly more problems, indicating that its state representation is more compact, allowing it to generate more states.

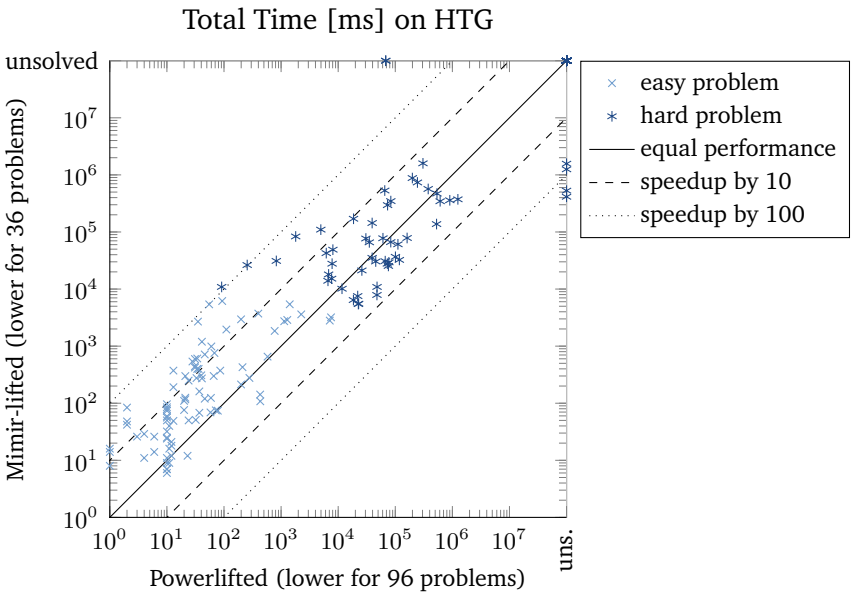



**Figure 8.2:** A pairwise comparison between Mimir-lifted and Powerlifted configurations on the HTG benchmark set, comparing the total time in milliseconds required to find a plan. We label a problem "easy" if both configurations require less than 10000 milliseconds, and otherwise, we label it "hard".

### 8.2.7 Lifted Comparison

Figure 8.2 shows the pairwise comparison between Powerlifted and Mimir-lifted on the HTG benchmark set. We use the labeling of "easy" and "hard" problems from before. On easy problems, Powerlifted performs better because Mimir computes static consistency graphs for each action schema (Ståhlberg 2023) that are quadratic in the number of possible assignments of objects to variables in the action. The advantage diminishes for hard problems, showing only a slight advantage of Powerlifted but solving two fewer problems overall.

Figure 8.3 shows the pairwise comparison between Powerlifted and Mimir-lifted on the STRIPS benchmark set. We use the labeling of "easy" and "hard" problems from before. On easy and hard problems, Mimir-lifted performs better, indicating that computing the static consistency graph becomes less of a concern as the number of objects per problem in the benchmark set is much smaller than in the HTG benchmark set. On hard problems, Mimir-lifted almost strictly outperforms Powerlifted while solving a significantly higher number of problems. Since the number of ground atoms is small, we suspect that Mimir's state compression step is advantageous, allowing it to generate more states.

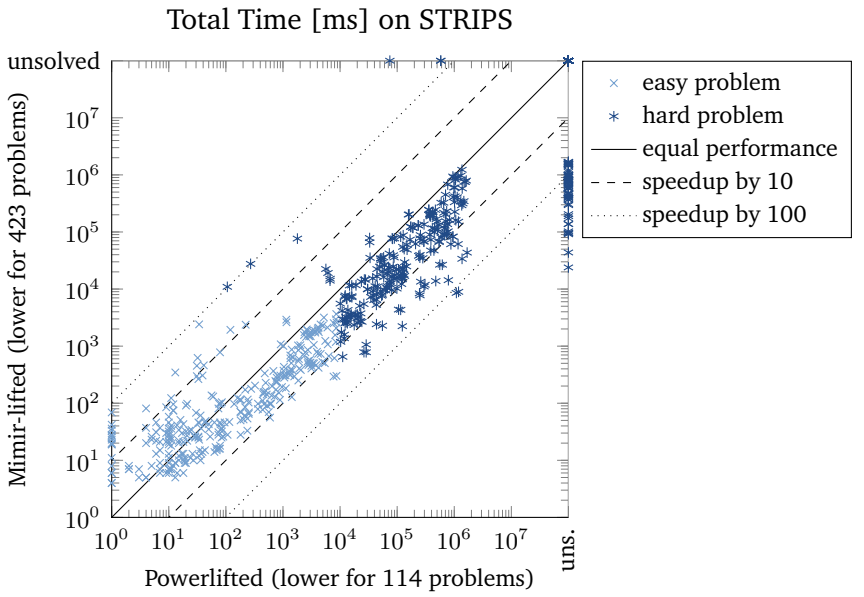



**Figure 8.3:** A pairwise comparison between Mimir-lifted and Powerlifted on the STRIPS benchmark, comparing the total time in milliseconds required to find a plan. We label a problem "easy" if both configurations require less than 10000 milliseconds, and otherwise, we label it "hard".

## 8.3 Discussion

We introduced a new version of Mimir, a planning library optimized for the lifted problem representation. We added support for expressive PDDL features like conditional effects and derived predicates for compactly modeling complex real-world problems. While Mimir is optimized for the lifted problem representation, it can optionally operate on the ground problem representation, which results in significant speedups if grounding is feasible. Mimir can work with sets of problems, establishing it as an invaluable tool for generalized planning, where general plans are often learned from sets of example problems.

While Mimir also incorporates well-known search algorithms like A$^*$ (Hart et al. 1968), its primary focus lies in employing efficient algorithms and data structures for the essential core functionalities critical to developing planners: 1) the computation of all applicable ground actions in a given state, and 2) the computation of the successor state when applying a ground action in the state. We use a k-clique enumeration for finding applicable ground actions (Ståhlberg 2023), which natively supports negative preconditions, while other precondition features are compiled away.

We conducted an experimental evaluation to test Mimir's ability to execute the above two core functionalities efficiently under time and memory constraints. Our comparison with state-of-the-art planners shows that Mimir is competitive on the grounded representation and competes or often outperforms on the lifted problem representation. The significant advantages of the lifted representation on the STRIPS benchmarks, which often contain few ground atoms, suggest that our state compression mechanism to the smallest required bit-width is advantageous.

Building on Mimir's strengths, our future work aims to expand its efficiency and scalability and handle increasingly more complex planning environments. The focus on the core planning functionality, such as computing the applicable ground actions in a state and computing the successor state by applying a ground action in a state, makes it straightforward to integrate more expressive formalisms, such as numeric, probabilistic, or non-deterministic planning.

A natural next step is integrating numeric planning by extending the state with numeric variables, extending actions with numeric effects, and incorporating numeric constraints and effects into the k-clique-based applicable action generator. Our future work can help researchers address more complex generalized planning problems with the grounded or lifted paradigm.

# 9 Conclusions

We showed that policy sketches can represent the common subgoal structure in many classical planning domains. Policy sketches are rules that split problems into subproblems whose polynomial complexity is characterized by the notion of width. We showed that policy sketches and the notion of width can illuminate the long-standing problem of uncovering effective hierarchical structures in artificial intelligence. We discussed its limitation in regards to reuse towards different problem classes. To overcome this limitation, we discussed a language extension called modules that parameterize policy sketches and make them reusable. We presented methods for learning policy sketches and hierarchical policies based on combinatorial optimization. Our solutions are interpretable and contain a few syntactic elements, allowing us to show their correctness for an entire problem class by hand. When the learning fails, it typically stems from limited scalability or a lack of expressivity. To address the scalability limitation, we introduced equivalence-based abstractions. Equivalence-based abstraction uses a notion of state symmetry (isomorphisms) to reduce the number of states during learning. To address the issue of expressivity, we introduced a method for testing the expressivity requirements of benchmark sets of problems over planning domains. An empirical evaluation of several benchmark sets shows that the expressivity needed is often upper bounded by three-variable first-order logic with counting quantifiers. Our method is a tool to understand failures and guide the approach towards the needed expressivity.

We also make steps towards building a library for generalized planning, which completely avoids an exponential grounding step that is often unnecessary in generalized planning and supports expressive language extensions such as conditional effects and derived predicates that cannot concisely be compiled away, allowing researchers to address and model more complex real-world planning domains. Our experimental results show that our library is competitive with state-of-the-art systems or outperforms them.

Our findings contribute to the practical and theoretical foundation of generalized planning by providing characterizations and methods for learning subgoal structures, abstractions based on symmetry reductions for more efficient learning, tools to explain failures of learning general plans, and an expressive planning library focusing on generalized planning.

# Bibliography


Aichmüller, Michael and Hector Geffner (2024). *Learning Sketch Decompositions in Planning via Deep Reinforcement Learning*. arXiv:2412.08574v1 [cs.AI].

Baader, Franz, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, eds. (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Babai, László (2016). "Graph isomorphism in quasipolynomial time [extended abstract]". In: *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, pp. 684–697.

Bacchus, Fahiem and Froduald Kabanza (2000). "Using Temporal Logics to Express Search Control Knowledge for Planning". In: *Artificial Intelligence* 116.1–2, pp. 123–191.

Barceló, Pablo, Egor Kostylev, Mikaël Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva (2020). "The Logical Expressiveness of Graph Neural Networks". In: *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*. OpenReview.net.

Barto, Andrew G. and Sridhar Mahadevan (2003). “Recent Advances in Hierarchical Reinforcement Learning”. In: *Discrete Event Dynamic Systems* 13.1, pp. 41–77.

Bengio, Yoshua, Jérôme Louradour, Ronan Collobert, and Jason Weston (2009). “Curriculum Learning”. In: *Proceedings of the 26th International Conference on Machine Learning (ICML 2009)*. ACM, pp. 41–48.

Bonet, Blai, Dominik Drexler, and Hector Geffner (2023). “General and Reusable Indexical Policies and Sketches”. In: *NeurIPS 2023 Workshop on Generalization in Planning*.

— (2024). “On Policy Reuse: An Expressive Language for Representing and Executing General Policies that Call Other Policies”. In: *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*. Ed. by Sara Bernardini and Christian Muise. AAAI Press, pp. 31–39.

Bonet, Blai and Héctor Geffner (2001). “Planning as Heuristic Search”. In: *Artificial Intelligence* 129.1, pp. 5–33.

Bonet, Blai and Hector Geffner (2018). “Features, Projections, and Representation Change for Generalized Planning”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*. Ed. by Jérôme Lang. IJCAI, pp. 4667–4673.

— (2020). “Qualitative Numeric Planning: Reductions and Complexity”. In: *Journal of Artificial Intelligence Research* 69, pp. 923–961.

— (2021). “General Policies, Representations, and Planning Width”. In: *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*. Ed. by Kevin Leyton-Brown and Mausam. AAAI Press, pp. 11764–11773.

— (2024). “General Policies, Subgoal Structure, and Planning Width”. In: *Journal of Artificial Intelligence Research* 80. DOI: 10.1613/jair.1.15581.

Bylander, Tom (1994). “The Computational Complexity of Propositional STRIPS Planning”. In: *Artificial Intelligence* 69.1–2, pp. 165–204.

Cai, Jin-Yi, Martin Fürer, and Neil Immerman (1992). “An optimal lower bound on the number of variables for graph identification”. In: *Combinatorica* 12.4, pp. 389–410.

Chen, Dillon Z., Sylvie Thiébaux, and Felipe W. Trevizan (2024). “Learning Domain-Independent Heuristics for Grounded and Lifted Planning”. In: *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*. Ed. by Jennifer Dy and Sriraam Natarajan. AAAI Press, pp. 20078–20086.

Chen, Dillon Z., Felipe W. Trevizan, and Sylvie Thiébaux (2024). “Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning”. In: *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*. Ed. by Sara Bernardini and Christian Muise. AAAI Press, pp. 68–76.

Corrêa, Augusto B. and Giuseppe De Giacomo (2024). “Lifted Planning: Recent Advances in Planning Using First-Order Representations”. In: *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*. IJCAI.

Corrêa, Augusto B., Florian Pommerening, Malte Helmert, and Guillem Francès (2020). “Lifted Successor Generation using Query Optimization Techniques”. In: *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*. Ed. by J. Christopher Beck, Erez Karpas, and Shirin Sohrabi. AAAI Press, pp. 80–89.

Dietterich, Thomas G. (2000). “Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition”. In: *Journal of Artificial Intelligence Research* 13, pp. 227–303.

Drexler, Dominik, Daniel Gnad, Paul Höft, Jendrik Seipp, David Speck, and Simon Ståhlberg (2023). “Ragnarok”. In: *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Drexler, Dominik, Javier Segovia-Aguas, and Jendrik Seipp (2022). “Learning General Policies and Helpful Action Classifiers from Partial State Spaces”. In: *IJCAI 2022 Workshop on Generalization in Planning*.

Drexler, Dominik and Jendrik Seipp (2023). “DLPlan: Description Logics State Features for Planning”. In: *ICAPS 2023 System Demonstrations and Exhibits*.

Drexler, Dominik, Jendrik Seipp, and Hector Geffner (2021). “Expressing and Exploiting the Common Subgoal Structure of Classical Planning Domains Using Sketches”. In: *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*. Ed. by Esra Erdem, Meghyn Bienvenu, and Gerhard Lakemeyer. IJCAI Organization, pp. 258–268.

— (2022). “Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width”. In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*. Ed. by Sylvie Thiébaux and William Yeoh. AAAI Press, pp. 62–70.

— (2023). “Learning Hierarchical Policies by Iteratively Reducing the Width of Sketch Rules”. In: *Proceedings of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*. Ed. by Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner. IJCAI Organization, pp. 208–218.

— (2024). “Expressing and Exploiting Subgoal Structure in Classical Planning Using Sketches”. In: *Journal of Artificial Intelligence Research* 80, pp. 171–208.

Drexler, Dominik, Jendrik Seipp, and David Speck (2023). “Odin: A Planner Based on Saturated Transition Cost Partitioning”. In: *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Drexler, Dominik, Simon Ståhlberg, Blai Bonet, and Hector Geffner (2024a). “Equivalence-Based Abstractions for Learning General Policies”. In: *ICAPS 2024 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*.

— (2024b). “Symmetries and Expressive Requirements for Learning General Policies”. In: *Proceedings of the Twenty-First International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*. IJCAI Organization.

Dumancic, Sebastijan, Tias Guns, and Andrew Cropper (2021). "Knowledge refactoring for inductive program synthesis". In: *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*. Ed. by Kevin Leyton-Brown and Mausam. AAAI Press, pp. 7271–7278.

Edelkamp, Stefan and Jörg Hoffmann (2004). *PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition*. Tech. rep. 195. University of Freiburg, Department of Computer Science.

Ellis, Kevin, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum (2020). *DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning*. arXiv:2006.08381 [cs.AI].

Erol, Kutluhan, James Hendler, and Dana S. Nau (1994). "HTN planning: complexity and expressivity". In: *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994)*. AAAI Press, pp. 1123–1128.

Ferber, Patrick, Liat Cohen, Jendrik Seipp, and Thomas Keller (2022). "Learning and Exploiting Progress States in Greedy Best-First Search". In: *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI 2022)*. Ed. by Luc De Raedt. IJCAI, pp. 4740–4746.

Fern, Alan, Sung Wook Yoon, and Robert Givan (2004). "Learning Domain-Specific Control Knowledge from Random Walks". In: *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*. Ed. by Shlomo Zilberstein, Jana Koehler, and Sven Koenig. AAAI Press, pp. 191–198.

Fox, Maria and Derek Long (1999). "The Detection and Exploitation of Symmetry in Planning Problems". In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*. Ed. by Thomas Dean. Morgan Kaufmann, pp. 956–961.

Fox, Maria and Derek Long (2003). “PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains”. In: *Journal of Artificial Intelligence Research* 20, pp. 61–124.

Francès, Guillem (2017). “Effective Planning with Expressive Languages”. PhD thesis. Universitat Pompeu Fabra.

Francès, Guillem, Blai Bonet, and Hector Geffner (2021). “Learning General Planning Policies from Small Examples Without Supervision”. In: *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*. Ed. by Kevin Leyton-Brown and Mausam. AAAI Press, pp. 11801–11808.

Francès, Guillem, Augusto B. Corrêa, Cedric Geissmann, and Florian Pommerening (2019). “Generalized Potential Heuristics for Classical Planning”. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*. Ed. by Sarit Kraus. IJCAI, pp. 5554–5561.

Garey, Michael R. and David S. Johnson (1979). *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman.

Gazen, B. Cenk and Craig A. Knoblock (1997). “Combining the Expressivity of UCPOP with the Efficiency of Graphplan”. In: *Recent Advances in AI Planning. 4th European Conference on Planning (ECP 1997)*. Ed. by Sam Steel and Rachid Alami. Vol. 1348. Lecture Notes in Artificial Intelligence. Springer-Verlag, pp. 221–233.

Gebser, Martin, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub (2012). *Answer Set Solving in Practice*. Morgan & Claypool Publishers.

Geißer, Florian (2018). “On Planning with State-dependent Action Costs”. PhD thesis. University of Freiburg.

Gelfond, Michael and Vladimir Lifschitz (1988). “The stable model semantics for logic programming.” In: *ICLP/SLP*. Vol. 88. Cambridge, MA, pp. 1070–1080.

Grohe, Martin (2021). “The Logic of Graph Neural Networks”. In: *Proceedings of the Thirty-Sixth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)*, pp. 1–17.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Helmert, Malte (2006). “The Fast Downward Planning System”. In: *Journal of Artificial Intelligence Research* 26, pp. 191–246.

— (2009). “Concise Finite-Domain Representations for PDDL Planning Tasks”. In: *Artificial Intelligence* 173, pp. 503–535.

Hoffmann, Jörg and Bernhard Nebel (2001). “The FF Planning System: Fast Plan Generation Through Heuristic Search”. In: 14, pp. 253–302.

Horcík, Rostislav and Gustav Sír (2024). “Expressiveness of Graph Neural Networks in Planning Domains”. In: *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*. Ed. by Sara Bernardini and Christian Muise. AAAI Press, pp. 281–289.

Icarte, Rodrigo Toro, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith (2022). “Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning”. In: *Journal of Artificial Intelligence Research* 73, pp. 173–208.

Immerman, Neil (1998). *Descriptive Complexity*. Springer Verlag.

Jiménez, Sergio, Javier Segovia-Aguas, and Anders Jonsson (2019). “A Review of Generalized Planning”. In: *The Knowledge Engineering Review* 34, e5.

Knoblock, Craig A. (1994). “Automatically Generating Abstractions for Planning”. In: *Artificial Intelligence* 68.2, pp. 243–302.

LeCun, Yann (2022). *A Path Towards Autonomous Machine Intelligence*. `https://openreview.net/forum?id=BZ5a1r-kVsf`. Accessed: 2024-09-06.

Lipovetzky, Nir and Hector Geffner (2012). “Width and Serialization of Classical Planning Problems”. In: *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*. Ed. by Luc De Raedt, Christian Bessiere, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter Lucas. IOS Press, pp. 540–545.

Lipovetzky, Nir and Hector Geffner (2017). “Best-First Width Search: Exploration and Exploitation in Classical Planning”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*. Ed. by Satinder Singh and Shaul Markovitch. AAAI Press, pp. 3590–3596.

Martín, Mario and Hector Geffner (2000). “Learning Generalized Policies from Planning Examples Using Concept Languages”. In: *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR 2000)*. Ed. by Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman. Morgan Kaufmann, pp. 667–677.

— (2004). “Learning Generalized Policies from Planning Examples Using Concept Languages”. In: *Applied Intelligence* 20.1, pp. 9–19.

McDermott, Drew, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins (1998). *PDDL – The Planning Domain Definition Language – Version 1.2*. Tech. rep. CVC TR-98-003/DCS TR-1165. Yale University: Yale Center for Computational Vision and Control.

McGovern, Amy and Andrew G. Barto (2001). “Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density”. In: *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*. Ed. by Carla E. Brodley and Andrea Pohoreckyj Danyluk. ACM, pp. 361–368.

McKay, Brendan D. and Adolfo Piperno (2014). “Practical graph isomorphism, II”. In: *Journal of Symbolic Computation* 60, pp. 94–112. DOI: https://doi.org/10.1016/j.jsc.2013.09.003.

Nebel, Bernhard (2000). “On the Compilability and Expressive Power of Propositional Planning Formalisms”. In: *Journal of Artificial Intelligence Research* 12, pp. 271–315.

Parr, Ronald and Stuart Russell (1997). “Reinforcement Learning with Hierarchies of Machines”. In: *Proc. NeurIPS*, pp. 1043–1049.

Pochter, Nir, Aviv Zohar, and Jeffrey S. Rosenschein (2011). “Exploiting Problem Symmetries in State-Based Planners”. In:

*Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*. Ed. by Wolfram Burgard and Dan Roth. AAAI Press, pp. 1004–1009.

Ramirez, Miquel, Nir Lipovetzky, and Christian Muise (2015). *Lightweight Automated Planning ToolKiT*. `http://lapkt.org/`.

Richter, Silvia and Matthias Westphal (2010). "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *Journal of Artificial Intelligence Research* 39, pp. 127–177.

Sacerdoti, Earl D. (1974). "Planning in a Hierarchy of Abstraction Spaces". In: *Artificial Intelligence* 5, pp. 115–135.

Scarselli, Franco, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini (2009). "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80.

Segovia-Aguas, Javier, Sergio Jiménez, and Anders Jonsson (2019). "Computing programs for generalized planning using a classical planner". In: *Artificial Intelligence* 272, pp. 52–85.

Seipp, Jendrik, Álvaro Torralba, and Jörg Hoffmann (2022). *PDDL Generators*. `https://doi.org/10.5281/zenodo.6382173`.

Sievers, Silvan, Gabriele Röger, Martin Wehrle, and Michael Katz (2019). "Theoretical Foundations for Structural Symmetries of Lifted PDDL Tasks". In: *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*. Ed. by Nir Lipovetzky, Eva Onaindia, and David E. Smith. AAAI Press, pp. 446–454.

Singh, Satinder P., Richard L. Lewis, Andrew G. Barto, and Jonathan Sorg (2010). "Intrinsically motivated reinforcement learning: An evolutionary perspective". In: *IEEE Transactions on Autonomous Mental Development* 2, pp. 70–82.

Speck, David, David Borukhson, Robert Mattmüller, and Bernhard Nebel (2021). "On the Compilability and Expressive Power of State-Dependent Action Costs". In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*. Ed. by Robert P. Goldman, Susanne Biundo, and Michael Katz. AAAI Press, pp. 358–366.

Ståhlberg, Simon (2023). “Lifted Successor Generation by Maximum Clique Enumeration”. In: *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*. Ed. by Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Rădulescu. IOS Press, pp. 2194–2201.

Ståhlberg, Simon, Blai Bonet, and Hector Geffner (2022a). “Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits”. In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*. Ed. by Sylvie Thiébaux and William Yeoh. AAAI Press, pp. 629–637.

— (2022b). “Learning Generalized Policies without Supervision Using GNNs”. In: *Proceedings of the Nineteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2022)*. Ed. by Gabriele Kern-Isberner, Gerhard Lakemeyer, and Thomas Meyer. IJCAI Organization, pp. 474–483.

— (2023). “Learning General Policies with Policy Gradient Methods”. In: *Proceedings of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*. Ed. by Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner. IJCAI Organization.

— (2025). “Learning More Expressive General Policies for Classical Planning Domains”. In: *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2025)*. AAAI Press.

Ståhlberg, Simon, Guillem Francès, and Jendrik Seipp (2021). “Learning Generalized Unsolvability Heuristics for Classical Planning”. In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*. Ed. by Zhi-Hua Zhou. IJCAI, pp. 4175–4181.

Tate, Austin (1977). “Generating Project Networks”. In: *Proc. IJCAI*, pp. 888–893.

Thiébaux, Sylvie, Jörg Hoffmann, and Bernhard Nebel (2005). “In Defense of PDDL Axioms”. In: *Artificial Intelligence* 168.1–2, pp. 38–69.

Torralba, Álvaro (2018). “Completeness-Preserving Dominance Techniques for Satisficing Planning”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*. Ed. by Jérôme Lang. IJCAI, pp. 4844–4851.

Torralba, Álvaro and Jörg Hoffmann (2015). “Simulation-Based Admissible Dominance Pruning”. In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. Ed. by Qiang Yang and Michael Wooldridge. AAAI Press, pp. 1689–1695.

Vallati, Mauro, Lukás Chrpa, and Thomas Leo McCluskey (2018). “What you always wanted to know about the deterministic part of the International Planning Competition (IPC) 2014 (but were too afraid to ask)”. In: *The Knowledge Engineering Review* 33.

Weisfeiler, Boris and Andrei Leman (1968). “The Reduction of a Graph to a Canonical Form and an Algebra Arising During This Reduction.” In: *Nauchno-Technicheskaya Informatsia*.

Yoon, Sungwook, Alan Fern, and Robert Givan (2008). “Learning Control Knowledge for Forward Search Planning”. In: *Journal of Machine Learning Research* 9, pp. 683–718.

Zheng, Zeyu, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado van Hasselt, David Silver, and Satinder Singh (2020). “What Can Learned Intrinsic Rewards Capture?” In: *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*. JMLR.org, pp. 11436–11446.

# Papers

The papers associated with this thesis have been removed for copyright reasons. For more details about these see:

https://doi.org/10.3384/9789181180206

Department of Computer and Information Science
Linköpings universitet

## Dissertations

## Linköping Studies in Science and Technology
## Linköping Studies in Arts and Sciences
*Linköping Studies in Statistics*
*Linköping Studies in Information Science*

**Linköping Studies in Science and Technology**

No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.

No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.

No 18 **Mats Cedwall**: Semantisk analys av process-beskrivningar i naturligt språk, 1977, ISBN 91- 7372-168-9.

No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.

No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System, 1978, ISBN 91- 7372-232-4.

No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.

No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.

No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.

No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.

No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.

No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.

No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91- 7372-527-7.

No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.

No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.

No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984, ISBN 91-7372-801-2.

No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372- 805-5.

No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.

No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.

No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.

No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.

No 192 **Dimiter Driankov:** Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.

No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.

No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.

No 221 **Michael Reinfrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.

No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.

No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.

No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies, 1991, ISBN 91-7870-784-6.

No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.

No 260 **Nahid Shahmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.

No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.

No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.

No 270 **Ralph Rönnquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.

No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.

No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.

No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.

No 281 **Christer Bäckström:** Computational Complexity of Reasoning about Plans, 1992, ISBN 91-7870-979-2.

No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.

No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.

No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.

No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.

No 338 **Simin Nadjm-Tehrani:** Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms, 1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund:** Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld:** Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén:** Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Languages from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-X.

No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis, 1998, ISBN 91-7219-369-7.

No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.

No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.

No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.

No 582 **Vanja Josifovski:** Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.

No 589 **Rita Kovordányi:** Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.

No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.

No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.

No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.

No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.

No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.

No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.

No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.

No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.

No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.

No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.

No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN 91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91-7373-207-9.

No 725 **Tim Heyer:** Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91-7373-208-7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91-7373-212-5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91-7373-258-3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-supported Interorganisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X.

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationssystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing – An Empirical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5.

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004, ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linguistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modelling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.

No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.

No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.

No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN 91-85297-97-6.

No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.

No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.

No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005, ISBN 91-85457-07-8.

No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.

No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.

No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.

No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.

No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.

No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.

No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.

No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.

No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.

No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.

No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.

No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8

No 1018 **Ioan Chisalita:** Communication and Networking Techniques for Traffic Safety Systems, 2006, ISBN 91-85523-77-1.

No 1019 **Tarja Susi:** The Puzzle of Social Activity - The Significance of Tools in Cognition and Cooperation, 2006, ISBN 91-85523-71-2.

No 1021 **Andrzej Bednarski:** Integrated Optimal Code Generation for Digital Signal Processors, 2006, ISBN 91-85523-69-0.

No 1022 **Peter Aronsson:** Automatic Parallelization of Equation-Based Simulation Programs, 2006, ISBN 91-85523-68-2.

No 1030 **Robert Nilsson:** A Mutation-based Framework for Automated Testing of Timeliness, 2006, ISBN 91-85523-35-6.

No 1034 **Jon Edvardsson:** Techniques for Automatic Generation of Tests from Programs and Specifications, 2006, ISBN 91-85523-31-3.

No 1035 **Vaida Jakoniene:** Integration of Biological Data, 2006, ISBN 91-85523-28-3.

No 1045 **Genevieve Gorrell:** Generalized Hebbian Algorithms for Dimensionality Reduction in Natural Language Processing, 2006, ISBN 91-85643-88-2.

No 1051 **Yu-Hsing Huang:** Having a New Pair of Glasses - Applying Systemic Accident Models on Road Safety, 2006, ISBN 91-85643-64-5.

No 1054 **Åsa Hedenskog:** Perceive those things which cannot be seen - A Cognitive Systems Engineering perspective on requirements management, 2006, ISBN 91-85643-57-2.

No 1061 **Cécile Åberg:** An Evaluation Platform for Semantic Web Technology, 2007, ISBN 91-85643-31-9.

No 1073 **Mats Grindal:** Handling Combinatorial Explosion in Software Testing, 2007, ISBN 978-91-85715-74-9.

No 1075 **Almut Herzog:** Usable Security Policies for Runtime Environments, 2007, ISBN 978-91-85715-65-7.

No 1079 **Magnus Wahlström:** Algorithms, measures, and upper bounds for Satisfiability and related problems, 2007, ISBN 978-91-85715-55-8.

No 1083 **Jesper Andersson:** Dynamic Software Architectures, 2007, ISBN 978-91-85715-46-6.

No 1086 **Ulf Johansson:** Obtaining Accurate and Comprehensible Data Mining Models - An Evolutionary Approach, 2007, ISBN 978-91-85715-34-3.

No 1089 **Traian Pop:** Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, 2007, ISBN 978-91-85715-27-5.

No 1091 **Gustav Nordh:** Complexity Dichotomies for CSP-related Problems, 2007, ISBN 978-91-85715-20-6.

No 1106 **Per Ola Kristensson:** Discrete and Continuous Shape Writing for Text Entry and Control, 2007, ISBN 978-91-85831-77-7.

No 1110 **He Tan:** Aligning Biomedical Ontologies, 2007, ISBN 978-91-85831-56-2.

No 1112 **Jessica Lindblom:** Minding the body - Interacting socially through embodied action, 2007, ISBN 978-91-85831-48-7.

No 1113 **Pontus Wärnestål:** Dialogue Behavior Management in Conversational Recommender Systems, 2007, ISBN 978-91-85831-47-0.

No 1120 **Thomas Gustafsson:** Management of Real-Time Data Consistency and Transient Overloads in Embedded Systems, 2007, ISBN 978-91-85831-33-3.

No 1127 **Alexandru Andrei:** Energy Efficient and Predictable Design of Real-time Embedded Systems, 2007, ISBN 978-91-85831-06-7.

No 1139 **Per Wikberg:** Eliciting Knowledge from Experts in Modeling of Complex Systems: Managing Variation and Interactions, 2007, ISBN 978-91-85895-66-3.

No 1143 **Mehdi Amirijoo:** QoS Control of Real-Time Data Services under Uncertain Workload, 2007, ISBN 978-91-85895-49-6.

No 1150 **Sanny Syberfeldt:** Optimistic Replication with Forward Conflict Resolution in Distributed Real-Time Databases, 2007, ISBN 978-91-85895-27-4.

No 1155 **Beatrice Alenljung:** Envisioning a Future Decision Support System for Requirements Engineering - A Holistic and Human-centred Perspective, 2008, ISBN 978-91-85895-11-3.

No 1156 **Artur Wilk:** Types for XML with Application to Xcerpt, 2008, ISBN 978-91-85895-08-3.

No 1183 **Adrian Pop:** Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages, 2008, ISBN 978-91-7393-895-2.

No 1185 **Jörgen Skågeby:** Gifting Technologies - Ethnographic Studies of End-users and Social Media Sharing, 2008, ISBN 978-91-7393-892-1.

No 1187 **Imad-Eldin Ali Abugessaisa:** Analytical tools and information-sharing methods supporting road safety organizations, 2008, ISBN 978-91-7393-887-7.

No 1204 **H. Joe Steinhauer:** A Representation Scheme for Description and Reconstruction of Object Configurations Based on Qualitative Relations, 2008, ISBN 978-91-7393-823-5.

No 1222 **Anders Larsson:** Test Optimization for Core-based System-on-Chip, 2008, ISBN 978-91-7393-768-9.

No 1238 **Andreas Borg:** Processes and Models for Capacity Requirements in Telecommunication Systems, 2009, ISBN 978-91-7393-700-9.

No 1240 **Fredrik Heintz:** DyKnow: A Stream-Based Knowledge Processing Middleware Framework, 2009, ISBN 978-91-7393-696-5.

No 1241 **Birgitta Lindström:** Testability of Dynamic Real-Time Systems, 2009, ISBN 978-91-7393-695-8.

No 1244 **Eva Blomqvist:** Semi-automatic Ontology Construction based on Patterns, 2009, ISBN 978-91-7393-683-5.

No 1249 **Rogier Woltjer:** Functional Modeling of Constraint Management in Aviation Safety and Command and Control, 2009, ISBN 978-91-7393-659-0.

No 1260 **Gianpaolo Conte:** Vision-Based Localization and Guidance for Unmanned Aerial Vehicles, 2009, ISBN 978-91-7393-603-3.

No 1262 **AnnMarie Ericsson:** Enabling Tool Support for Formal Analysis of ECA Rules, 2009, ISBN 978-91-7393-598-2.

No 1266 **Jiri Trnka:** Exploring Tactical Command and Control: A Role-Playing Simulation Approach, 2009, ISBN 978-91-7393-571-5.

No 1268 **Bahlol Rahimi:** Supporting Collaborative Work through ICT - How End-users Think of and Adopt Integrated Health Information Systems, 2009, ISBN 978-91-7393-550-0.

No 1274 **Fredrik Kuivinen:** Algorithms and Hardness Results for Some Valued CSPs, 2009, ISBN 978-91-7393-525-8.

No 1281 **Gunnar Mathiason:** Virtual Full Replication for Scalable Distributed Real-Time Databases, 2009, ISBN 978-91-7393-503-6.

No 1290 **Viacheslav Izosimov:** Scheduling and Optimization of Fault-Tolerant Distributed Embedded Systems, 2009, ISBN 978-91-7393-482-4.

No 1294 **Johan Thapper:** Aspects of a Constraint Optimisation Problem, 2010, ISBN 978-91-7393-464-0.

No 1306 **Susanna Nilsson:** Augmentation in the Wild: User Centered Development and Evaluation of Augmented Reality Applications, 2010, ISBN 978-91-7393-416-9.

No 1313 **Christer Thörn:** On the Quality of Feature Models, 2010, ISBN 978-91-7393-394-0.

No 1321 **Zhiyuan He:** Temperature Aware and Defect-Probability Driven Test Scheduling for System-on-Chip, 2010, ISBN 978-91-7393-378-0.

No 1333 **David Broman:** Meta-Languages and Semantics for Equation-Based Modeling and Simulation, 2010, ISBN 978-91-7393-335-3.

No 1337 **Alexander Siemers:** Contributions to Modelling and Visualisation of Multibody Systems Simulations with Detailed Contact Analysis, 2010, ISBN 978-91-7393-317-9.

No 1354 **Mikael Asplund:** Disconnected Discoveries: Availability Studies in Partitioned Networks, 2010, ISBN 978-91-7393-278-3.

No 1359 **Jana Rambusch**: Mind Games Extended: Understanding Gameplay as Situated Activity, 2010, ISBN 978-91-7393-252-3.

No 1373 **Sonia Sangari**: Head Movement Correlates to Focus Assignment in Swedish, 2011, ISBN 978-91-7393-154-0.

No 1374 **Jan-Erik Källhammer**: Using False Alarms when Developing Automotive Active Safety Systems, 2011, ISBN 978-91-7393-153-3.

No 1375 **Mattias Eriksson**: Integrated Code Generation, 2011, ISBN 978-91-7393-147-2.

No 1381 **Ola Leifler**: Affordances and Constraints of Intelligent Decision Support for Military Command and Control – Three Case Studies of Support Systems, 2011, ISBN 978-91-7393-133-5.

No 1386 **Soheil Samii**: Quality-Driven Synthesis and Optimization of Embedded Control Systems, 2011, ISBN 978-91-7393-102-1.

No 1419 **Erik Kuiper**: Geographic Routing in Intermittently-connected Mobile Ad Hoc Networks: Algorithms and Performance Models, 2012, ISBN 978-91-7519-981-8.

No 1451 **Sara Stymne**: Text Harmonization Strategies for Phrase-Based Statistical Machine Translation, 2012, ISBN 978-91-7519-887-3.

No 1455 **Alberto Montebelli**: Modeling the Role of Energy Management in Embodied Cognition, 2012, ISBN 978-91-7519-882-8.

No 1465 **Mohammad Saifullah**: Biologically-Based Interactive Neural Network Models for Visual Attention and Object Recognition, 2012, ISBN 978-91-7519-838-5.

No 1490 **Tomas Bengtsson**: Testing and Logic Optimization Techniques for Systems on Chip, 2012, ISBN 978-91-7519-742-5.

No 1481 **David Byers**: Improving Software Security by Preventing Known Vulnerabilities, 2012, ISBN 978-91-7519-784-5.

No 1496 **Tommy Färnqvist**: Exploiting Structure in CSP-related Problems, 2013, ISBN 978-91-7519-711-1.

No 1503 **John Wilander**: Contributions to Specification, Implementation, and Execution of Secure Software, 2013, ISBN 978-91-7519-681-7.

No 1506 **Magnus Ingmarsson**: Creating and Enabling the Useful Service Discovery Experience, 2013, ISBN 978-91-7519-662-6.

No 1547 **Wladimir Schamai**: Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool, 2013, ISBN 978-91-7519-505-6.

No 1551 **Henrik Svensson**: Simulations, 2013, ISBN 978-91-7519-491-2.

No 1559 **Sergiu Rafiliu**: Stability of Adaptive Distributed Real-Time Systems with Dynamic Resource Management, 2013, ISBN 978-91-7519-471-4.

No 1581 **Usman Dastgeer**: Performance-aware Component Composition for GPU-based Systems, 2014, ISBN 978-91-7519-383-0.

No 1602 **Cai Li**: Reinforcement Learning of Locomotion based on Central Pattern Generators, 2014, ISBN 978-91-7519-313-7.

No 1652 **Roland Samlaus**: An Integrated Development Environment with Enhanced Domain-Specific Interactive Model Validation, 2015, ISBN 978-91-7519-090-7.

No 1663 **Hannes Uppman**: On Some Combinatorial Optimization Problems: Algorithms and Complexity, 2015, ISBN 978-91-7519-072-3.

No 1664 **Martin Sjölund**: Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models, 2015, ISBN 978-91-7519-071-6.

No 1666 **Kristian Stavåker**: Contributions to Simulation of Modelica Models on Data-Parallel Multi-Core Architectures, 2015, ISBN 978-91-7519-068-6.

No 1680 **Adrian Lifa**: Hardware/Software Codesign of Embedded Systems with Reconfigurable and Heterogeneous Platforms, 2015, ISBN 978-91-7519-040-2.

No 1685 **Bogdan Tanasa**: Timing Analysis of Distributed Embedded Systems with Stochastic Workload and Reliability Constraints, 2015, ISBN 978-91-7519-022-8.

No 1691 **Håkan Warnquist**: Troubleshooting Trucks – Automated Planning and Diagnosis, 2015, ISBN 978-91-7685-993-3.

No 1702 **Nima Aghaee**: Thermal Issues in Testing of Advanced Systems on Chip, 2015, ISBN 978-91-7685-949-0.

No 1715 **Maria Vasilevskaya**: Security in Embedded Systems: A Model-Based Approach with Risk Metrics, 2015, ISBN 978-91-7685-917-9.

No 1729 **Ke Jiang**: Security-Driven Design of Real-Time Embedded System, 2016, ISBN 978-91-7685-884-4.

No 1733 **Victor Lagerkvist**: Strong Partial Clones and the Complexity of Constraint Satisfaction Problems: Limitations and Applications, 2016, ISBN 978-91-7685-856-1.

No 1734 **Chandan Roy**: An Informed System Development Approach to Tropical Cyclone Track and Intensity Forecasting, 2016, ISBN 978-91-7685-854-7.

No 1746 **Amir Aminifar**: Analysis, Design, and Optimization of Embedded Control Systems, 2016, ISBN 978-91-7685-826-4.

No 1747 **Ekhiotz Vergara**: Energy Modelling and Fairness for Efficient Mobile Communication, 2016, ISBN 978-91-7685-822-6.

No 1748 **Dag Sonntag**: Chain Graphs – Interpretations, Expressiveness and Learning Algorithms, 2016, ISBN 978-91-7685-818-9.

No 1768 **Anna Vapen**: Web Authentication using Third-Parties in Untrusted Environments, 2016, ISBN 978-91-7685-753-3.

No 1778 **Magnus Jandinger**: On a Need to Know Basis: A Conceptual and Methodological Framework for Modelling and Analysis of Information Demand in an Enterprise Context, 2016, ISBN 978-91-7685-713-7.

No 1798 **Rahul Hiran**: Collaborative Network Security: Targeting Wide-area Routing and Edge-network Attacks, 2016, ISBN 978-91-7685-662-8.

No 1813 **Nicolas Melot**: Algorithms and Framework for Energy Efficient Parallel Stream Computing on Many-Core Architectures, 2016, ISBN 978-91-7685-623-9.

No 1823 **Amy Rankin**: Making Sense of Adaptations: Resilience in High-Risk Work, 2017, ISBN 978-91-7685-596-6.

No 1831 **Lisa Malmberg**: Building Design Capability in the Public Sector: Expanding the Horizons of Development, 2017, ISBN 978-91-7685-585-0.

No 1851 **Marcus Bendtsen**: Gated Bayesian Networks, 2017, ISBN 978-91-7685-525-6.

No 1852 **Zlatan Dragisic**: Completion of Ontologies and Ontology Networks, 2017, ISBN 978-91-7685-522-5.

No 1854 **Meysam Aghighi**: Computational Complexity of some Optimization Problems in Planning, 2017, ISBN 978-91-7685-519-5.

No 1863 **Simon Ståhlberg**: Methods for Detecting Unsolvable Planning Instances using Variable Projection, 2017, ISBN 978-91-7685-498-3.

No 1879 **Karl Hammar**: Content Ontology Design Patterns: Qualities, Methods, and Tools, 2017, ISBN 978-91-7685-454-9.

No 1887 **Ivan Ukhov**: System-Level Analysis and Design under Uncertainty, 2017, ISBN 978-91-7685-426-6.

No 1891 **Valentina Ivanova**: Fostering User Involvement in Ontology Alignment and Alignment Evaluation, 2017, ISBN 978-91-7685-403-7.

No 1902 **Vengatanathan Krishnamoorthi**: Efficient HTTP-based Adaptive Streaming of Linear and Interactive Videos, 2018, ISBN 978-91-7685-371-9.

No 1903 **Lu Li**: Programming Abstractions and Optimization Techniques for GPU-based Heterogeneous Systems, 2018, ISBN 978-91-7685-370-2.

No 1913 **Jonas Rybing**: Studying Simulations with Distributed Cognition, 2018, ISBN 978-91-7685-348-1.

No 1936 **Leif Jonsson**: Machine Learning-Based Bug Handling in Large-Scale Software Development, 2018, ISBN 978-91-7685-306-1.

No 1964 **Arian Maghazeh**: System-Level Design of GPU-Based Embedded Systems, 2018, ISBN 978-91-7685-175-3.

No 1967 **Mahder Gebremedhin**: Automatic and Explicit Parallelization Approaches for Equation Based Mathematical Modeling and Simulation, 2019, ISBN 978-91-7685-163-0.

No 1984 **Anders Andersson**: Distributed Moving Base Driving Simulators – Technology, Performance, and Requirements, 2019, ISBN 978-91-7685-090-9.

No 1993 **Ulf Kargén**: Scalable Dynamic Analysis of Binary Code, 2019, ISBN 978-91-7685-049-7.

No 2001 **Tim Overkamp**: How Service Ideas Are Implemented: Ways of Framing and Addressing Service Transformation, 2019, ISBN 978-91-7685-025-1.

No 2006 **Daniel de Leng**: Robust Stream Reasoning Under Uncertainty, 2019, ISBN 978-91-7685-013-8.

No 2048 **Biman Roy**: Applications of Partial Polymorphisms in (Fine-Grained) Complexity of Constraint Satisfaction Problems, 2020, ISBN 978-91-7929-898-2.

No 2051 **Olov Andersson**: Learning to Make Safe Real-Time Decisions Under Uncertainty for Autonomous Robots, 2020, ISBN 978-91-7929-889-0.

No 2065 **Vanessa Rodrigues**: Designing for Resilience: Navigating Change in Service Systems, 2020, ISBN 978-91-7929-867-8.

No 2082 **Robin Kurtz**: Contributions to Semantic Dependency Parsing: Search, Learning, and Application, 2020, ISBN 978-91-7929-822-7.

No 2108 **Shanai Ardi**: Vulnerability and Risk Analysis Methods and Application in Large Scale Development of Secure Systems, 2021, ISBN 978-91-7929-744-2.

No 2125 **Zeinab Ganjei**: Parameterized Verification of Synchronized Concurrent Programs, 2021, ISBN 978-91-7929-697-1.

No 2153 **Robin Keskisärkkä**: Complex Event Processing under Uncertainty in RDF Stream Processing, 2021, ISBN 978-91-7929-621-6.

No 2168 **Rouhollah Mahfouzi**: Security-Aware Design of Cyber-Physical Systems for Control Applications, 2021, ISBN 978-91-7929-021-4.

No 2205 **August Ernstsson**: Pattern-based Programming Abstractions for Heterogeneous Parallel Computing, 2022, ISBN 978-91-7929-195-2.

No 2218 **Huanyu Li**: Ontology-Driven Data Access and Data Integration with an Application in the Materials Design Domain, 2022, ISBN 978-91-7929-267-6.

No 2219 **Evelina Rennes**: Automatic Adaption of Swedish Text for Increased Inclusion, 2022, ISBN 978-91-7929-269-0.

No 2220 **Yuanbin Zhou**: Synthesis of Safety-Critical Real-Time Systems, 2022, ISBN 978-91-7929-271-3.

No 2247 **Azeem Ahmad**: Contributions to Improving Feedback and Trust in Automated Testing and Continuous Integration and Delivery, 2022, ISBN 978-91-7929-422-9.

No 2248 **Ana Kuštrak Korper**: Innovating Innovation: Understanding the Role of Service Design in Service Innovation, 2022, ISBN 978-91-7929-424-3.

No 2256 **Adrian Horga:** Performance and Security Analysis for GPU-Based Applications, 2022, ISBN 978-91-7929-487-8.

No 2262 **Mattias Tiger:** Safety-Aware Autonomous Systems: Preparing Robots for Life in the Real World 2022, ISBN 978-91-7929-501-1.

No 2266 **Chih-Yuan Lin:** Network-based Anomaly Detection for SCADA Systems: Traffic Generation and Modeling, 2022, ISBN 978-91-7929-517-2.

No 2280 **Filip Strömbäck:** Teaching and Learning Concurrent Programming in the Shared Memory Model, 2023, ISBN 978-91-8075-000-4.

No 2298 **Fiona Lambe:** Devising Capabilities: Service Design for Development Interventions, 2023, ISBN 978-91-8075-080-6.

No 2309 **Alachew Mengist:** Model-Based Tools Integration and Ontology-Driven Traceability in Model-Based Development Environments, 2023, ISBN 978-91-8075-143-8.

No 2322 **Mariusz Wzorek:** Selected Functionalities for Autonomous Intelligent Systems in Public Safety Scenarios, 2023, ISBN 978-91-8075-195-7.

No 2329 **Felipe Boeira:** Authentic Communication and Trustworthy Location in Mobile Networks, 2023, ISBN 978-91-8075-256-5.

No 2351 **Johan Källström:** Reinforcement Learning for Improved Utility of Simulation-Based Training, 2023, ISBN 978-91-8075-366-1.

No 2364 **Jenny Kunz:** Understanding Large Language Models: Towards Rigorous and Targeted Interpretability Using Probing Classifiers and Self-Rationalisation, 2024, ISBN 978-91-8075-470-5.

No 2366 **Sijin Cheng:** Query Processing over Heterogeneous Federations of Graph Data, 2024, ISBN 978-91-8075-488-0.

No 2368 **George Osipov:** On Infinite-Domain CSPs Parameterized by Solution Cost, 2024, ISBN 978-91-8075-496-5.

No 2382 **Fredrik Präntare:** Dividing the Indivisible: Algorithms, Empiricial Advances, and Complexity Results for Value-Maximizing Combinatorial Assignment Problems, 2024, ISBN 978-91-8075-600-6.

No 2383 **Alireza Mohammadinodooshan:** Data-driven Contributions to Understanding User Engagement Dynamics on Social Media, 2024, ISBN 978-91-8075-606-8.

No 2384 **Rodrigo Saar de Moraes:** Exploring Trade-offs in Concept Design of Integrated Modular Avionic Platform Configurations: Topology Generation, Resource Adequacy, and Dependability, 2024, ISBN 978-91-8075-13-16.

No 2392 **Minh Ha Le:** Beyond Recognition: Privacy Protections in a Surveilled World, 2024, ISBN 978-91-8075-675-4.

No 2403 **Klervie Toczé:** Orchestrating a Resource-aware Edge, 2024, ISBN 978-91-8075-747-8.

No 2264 **Robert Johansson:** Empirical Studies in Machine Psychology, 2024, ISBN 978-91-7929-505-9.

No 2429 **Mina Niknafs:** Prediction-Based Resource Management for Heterogeneous Multi-Core Embedded Systems, 2025, ISBN 978-91-8075-960-1.

No 2439 **Dominik Drexler:** Learning and Exploiting Subgoal Structures in Classical Planning: Towards Reliable and Transparent Intelligent Agents that Learn to Plan on Multiple Levels, 2025, ISBN 978-91-8118-019-0.

**Linköping Studies in Arts and Sciences**

No 504 **Ing-Marie Jonsson:** Social and Emotional Characteristics of Speech-based In-Vehicle Information Systems: Impact on Attitude and Driving Behaviour, 2009, ISBN 978-91-7393-478-7.

No 586 **Fabian Segelström:** Stakeholder Engagement for Service Design: How service designers identify and communicate insights, 2013, ISBN 978-91-7519-554-4.

No 618 **Johan Blomkvist:** Representing Future Situations of Service: Prototyping in Service Design, 2014, ISBN 978-91-7519-343-4.

No 620 **Marcus Mast:** Human-Robot Interaction for Semi-Autonomous Assistive Robots, 2014, ISBN 978-91-7519-319-9.

No 677 **Peter Berggren:** Assessing Shared Strategic Understanding, 2016, ISBN 978-91-7685-786-1.

No 695 **Mattias Forsblad:** Distributed cognition in home environments: The prospective memory and cognitive practices of older adults, 2016, ISBN 978-91-7685-686-4.

No 787 **Sara Nygårdhs:** Adaptive behaviour in traffic: An individual road user perspective, 2020, ISBN 978-91-7929-857-9.

No 811 **Sam Thellman:** Social Robots as Intentional Agents, 2021, ISBN, 978-91-7929-008-5.

No 878 **Sofia Thunberg:** Companion Robots for Older Adults: A Mixed-Methods Approach to Deployments in Care Homes 2024, ISBN, 978-91-8075-573-3.

No 891 **Kajsa Weibull:** Emergency Vehicle Approaching: Warning Drivers using Cooperative Intelligent Transport Systems 2024, ISBN, 978-91-8075-804-8.

*Linköping Studies in Statistics*

No 9 **Davood Shahsavani:** Computer Experiments Designed to Explore and Approximate Complex Deterministic Models, 2008, ISBN 978-91-7393-976-8.

No 10 **Karl Wahlin:** Roadmap for Trend Detection and Assessment of Data Quality, 2008, ISBN 978-91-7393-792-4.

No 11 **Oleg Sysoev:** Monotonic regression for large multivariate datasets, 2010, ISBN 978-91-7393-412-1.

No 13 **Agné Burauskaite-Harju:** Characterizing Temporal Change and Inter-Site Correlations in Daily and Subdaily Precipitation Extremes, 2011, ISBN 978-91-7393-110-6.

No 14 **Måns Magnusson:** Scalable and Efficient Probabilistic Topic Model Inference for Textual Data, 2018, ISBN 978-91-7685-288-0.

No 15 **Per Sidén:** Scalable Bayesian spatial analysis with Gaussian Markov random fields, 2020, 978-91-7929-818-0.

No 16 **Caroline Svahn:** Prediction Methods for High Dimensional Data with Censored Covariates, 2022, 978-91-7929-398-7.

No 17 **Héctor Rodriguez Déniz:** Bayesian Models for Spatiotemporal Data from Transportation Networks, 2023, 978-91-8075-035-6.

No 18 **Amanda Olmin:** Perspectives on Predictive and Annotation Uncertainty in Probablistic Machine Learning 2024, 978-91-8075-798-0.

*Linköping Studies in Information Science*

No 1 **Karin Axelsson:** Metodisk systemstrukturering- att skapa samstämmighet mellan informationssystemarkitektur och verksamhet, 1998. ISBN 9172-19-296-8.

No 2 **Stefan Cronholm:** Metodverktyg och användbarhet - en studie av datorstödd metodbaserad systemutveckling, 1998, ISBN 9172-19-299-2.

No 3 **Anders Avdic:** Användare och utvecklare - om anveckling med kalkylprogram, 1999. ISBN 91-7219-606-8.

No 4 **Owen Eriksson:** Kommunikationskvalitet hos informationssystem och affärsprocesser, 2000, ISBN 91-7219-811-7.

No 5 **Mikael Lind:** Från system till process - kriterier för processbestämning vid verksamhetsanalys, 2001, ISBN 91-7373-067-X.

No 6 **Ulf Melin:** Koordination och informationssystem i företag och nätverk, 2002, ISBN 91-7373-278-8.

No 7 **Pär J. Ågerfalk:** Information Systems Actability - Understanding Information Technology as a Tool for Business Action and Communication, 2003, ISBN 91-7373-628-7.

No 8 **Ulf Seigerroth:** Att förstå och förändra systemutvecklingsverksamheter - en taxonomi för metautveckling, 2003, ISBN 91-7373-736-4.

No 9 **Karin Hedström:** Spår av datoriseringens värden – Effekter av IT i äldreomsorg, 2004, ISBN 91-7373-963-4.

No 10 **Ewa Braf:** Knowledge Demanded for Action - Studies on Knowledge Mediation in Organisations, 2004, ISBN 91-85295-47-7.

No 11 **Fredrik Karlsson:** Method Configuration method and computerized tool support, 2005, ISBN 91-85297-48-8.

No 12 **Malin Nordström:** Styrbar systemförvaltning - Att organisera systemförvaltningsverksamhet med hjälp av effektiva förvaltningsobjekt, 2005, ISBN 91-85297-60-7.

No 13 **Stefan Holgersson:** Yrke: POLIS - Yrkeskunskap, motivation, IT-system och andra förutsättningar för polisarbete, 2005, ISBN 91-85299-43-X.

No 14 **Benneth Christiansson, Marie-Therese Christiansson:** Mötet mellan process och komponent - mot ett ramverk för en verksamhetsnära kravspecifikation vid anskaffning av komponentbaserade informationssystem, 2006, ISBN 91-85643-22-X.

**FACULTY OF SCIENCE AND ENGINEERING**

Linköping Studies in Science and Technology, Dissertation No. 2439, 2025
Department of Computer and Information Science

Linköping University
SE-581 83 Linköping, Sweden

www.liu.se