

# Lecture Notes in Artificial Intelligence 2654

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

**Springer**

*Berlin*

*Heidelberg*

*New York*

*Hong Kong*

*London*

*Milan*

*Paris*

*Tokyo*

Ute Schmid

# Inductive Synthesis of Functional Programs

Universal Planning, Folding of Finite Programs,  
and Schema Abstraction by Analogical Reasoning



Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA  
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Author

Ute Schmid  
University of Osnabrück  
Institute of Computer Science  
Department of Mathematics and Computer Science  
Albrechtstr. 28, 49069 Osnabrück, Germany  
E-mail: schmid@informatik.uni-osnabrueck.de

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek  
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie;  
detailed bibliographic data is available in the Internet at <<http://dnb.ddb.de>>.

CR Subject Classification (1998): I.2.2, I.2.3, I.2.4, I.2.8, D.1.2, F.3.1, F.4.1, D.2.11

ISSN 0302-9743

ISBN 3-540-40174-1 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York  
a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2003  
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign

Coverillustration: "Nachtigall II" by Heinrich Neuy

Printed on acid-free paper      SPIN: 10932261      06/3142      5 4 3 2 1 0

*For My Parents and  
In Memory of My Grandparents*

# Foreword

Analogical reasoning is ubiquitous, whether in everyday common sense reasoning, in scientific discovery, or anywhere in between. Examples of analogical reasoning range from scientific theory creation, such as Bohr's planetary model of the atom, to problem solving where a teacher's solution of an illustrative example problem is used to guide the student in solving new, similar problems. Psychologists have studied how people reason analogically, though often severely simplifying the reasoning task in order to run controlled experiments. Artificial intelligence researchers, including this writer, have built computational models that exhibit various forms of analogical transfer, ranging from simple copy-and-modify processes to complex derivational-trace tracking and rejustifying reasoning steps for new problems. The underlying issues are not simple. For instance, in drawing an analogy, what should be kept invariant, what should be modified or mapped, and what should be discarded? At what level of reasoning is analogy most profitably applied – i.e., should the solution to a problem be transferred and modified, should the derivation of the solution be transferred instead, or should the underlying principles invoked in the derivation be the primary transfer vehicle? How does analogical reasoning interact with classical deduction or with inductive reasoning? And how can a solution drawn analogically be formally verified or refuted, in the sense of formal proof checking? These and other key issues lie at the heart of analogical reasoning research.

Artificial intelligence researchers and cognitive psychologists have addressed subsets of the analogical reasoning challenge. However, until now there has not been a true marriage of the psychological and the computational in the realm of analogical reasoning. Although both camps cite each other and mutually benefit from new results, Ute Schmid is the first to develop, implement, test, and evaluate analogical reasoning models in depth based directly on data from subjects performing that reasoning.

This book is a very thorough and clear report of Dr. Schmid's deep analysis of inductive and analogical reasoning, combining key aspects of artificial intelligence and algorithms on the one hand, and cognitive psychology on the other. Compared with related work, the comprehensive nature of the analogical reasoning model is evident: The case-based reasoning (CBR) community focuses primarily on indexing and retrieving relevant past cases, rather than

deriving new solutions or solving significantly different problems. Earlier analogical reasoning work focused directly on problem solving – how to use past solutions for similar problems to help construct the solution to the new problem. Veloso combined CBR and analogical reasoning, enabling large-scale problem solving from second-principles. Subsequently, analogical reasoning has seen new extensions such as Melis’s method for analogical construction of mathematical proofs and the use of analogy in intelligent tutoring systems. This book combines all the aspects of analogical reasoning, extends it to include other forms of inductive and deductive reasoning, and directly ties the computational methods to psychological results.

June 2003

Jaime Carbonell

# Preface

In this book a novel approach to inductive synthesis of recursive functions is proposed, combining universal planning, folding of finite programs, and schema abstraction by analogical reasoning. In a first step, an example domain of small complexity is explored by universal planning. For example, for all possible lists over four fixed natural numbers, their optimal transformation sequences into the sorted list are calculated and represented as a DAG. In a second step, the plan is transformed into a finite program term. Plan transformation mainly relies on inferring the data type underlying a given plan. In a third step, the finite program is folded into (a set of) recursive functions. Folding is performed by syntactical pattern-matching and corresponds to inducing a special class of context-free tree grammars. It is shown that the approach can be successfully applied to learn domain-specific control rules. Control rule learning is important to gain the efficiency of domain-specific planners without the need to hand-code the domain-specific knowledge. Furthermore, an extension of planning based on a purely relational domain description language to function applications is presented. This extension makes planning applicable to a larger class of problems that are of interest for program synthesis.

As a last step, a hierarchy of program schemes (patterns) is generated by generalizing over already synthesized recursive functions. Generalization can be considered as the last step of problem solving by analogy or programming by analogy. Some psychological experiments were performed to investigate which kind of structural relations between problems can be exploited by human problem-solvers. Anti-unification is presented as an approach to mapping and generalizing program structures. It is proposed that the integration of planning, program synthesis, and analogical reasoning contributes to cognitive science research on skill acquisition by addressing the problem of extracting generalized rules from some initial experience. Such (control) rules represent domain-specific problem-solving strategies.

All parts of the approach are implemented in Common Lisp.

**Acknowledgements.** Research is a kind of work one can never do completely alone. Over the years I profited from the guidance of several professors who supervised or supported my work, namely Fritz Wysotzki, Klaus Eyferth, Bernd Mahr, Jaime Carbonell, Arnold Upmeyer, Peter Pepper, and



Gerhard Strube. I learned a lot from discussions with them, with colleagues, and students, such as Jochen Burghardt, Bruce Burns, the group of Hartmut Ehrig, Pierre Flener, Hector Geffner, Peter Geibel, Peter Gerjets, Jürgen Giesl, Wolfgang Grieskamp, Maritta Heisel, Ralf Herbrich, Laurie Hiyakumoto, Petra Hofstedt, Rune Jensen, Emanuel Kitzelmann, Jana Koehler, Steffen Lange, Martin Mühlpfordt, Brigitte Pientka, Heike Pisch, Manuela Veloso, Ulrich Wagner, Bernhard Wolf, and Thomas Zeugmann (sorry to everyone I forgot). I am very grateful for the time I could spend at Carnegie Mellon University. Thanks to Klaus Eyferth and Bernd Mahr who motivated me to go, to Gerhard Strube for his support, to Fritz Wysotzki who accepted my absence from teaching, and, of course, to Jaime Carbonell who was my very helpful host. My work profited much from the inspiration I got from talks, classes, and discussions, and from the very special atmosphere suggesting that everything is all right as long as “the heart is in the work.” I thank all my diploma students who supported the work reported in this book – Dirk Matzke, Rene Mercy, Martin Mühlpfordt, Marina Müller, Mark Müller, Heike Pisch, Knut Polkehn, Uwe Sinha, Imre Szabo, Janin Toussaint, Ulrich Wagner, Joachim Wirth, and Bernhard Wolf. Additional thanks to some of them and Peter Pollmanns for proof-reading parts of the draft of this book. I owe a lot to Fritz Wysotzki for giving me the chance to move from cognitive psychology to artificial intelligence, for many interesting discussions, and for critically reading and commenting on the draft of this book. Finally, thanks to my colleagues and friends Berry Claus, Robin Hörnig, Barbara Kaup, and Martin Kindsmüller, to my family, and my husband Uwe Konerding for support and high-quality leisure time, and to all authors of good crime novels.

# Table of Contents

<b>1. Introduction</b> .....	1
------------------------------	---

---

## Part I. Planning

---

<b>2. State-Based Planning</b> .....	13
2.1 Standard Strips .....	13
2.1.1 A Blocks-World Example .....	14
2.1.2 Basic Definitions .....	14
2.1.3 Backward Operator Application .....	18
2.2 Extensions and Alternatives to Strips .....	20
2.2.1 The Planning Domain Definition Language .....	20
2.2.2 Situation Calculus .....	24
2.3 Basic Planning Algorithms .....	27
2.3.1 Informal Introduction of Basic Concepts .....	28
2.3.2 Forward Planning .....	29
2.3.3 Formal Properties of Planning .....	32
2.3.4 Backward Planning .....	35
2.4 Planning Systems .....	40
2.4.1 Classical Approaches .....	40
2.4.2 Current Approaches .....	42
2.4.3 Complex Domains and Uncertain Environments .....	44
2.4.4 Universal Planning .....	45
2.4.5 Planning and Related Fields .....	48
2.4.6 Planning Literature .....	50
2.5 Automatic Knowledge Acquisition for Planning .....	51
2.5.1 Pre-planning Analysis .....	51
2.5.2 Planning and Learning .....	51
<b>3. Constructing Complete Sets of Optimal Plans</b> .....	55
3.1 Introduction to DPlan .....	55
3.1.1 DPlan Planning Language .....	56
3.1.2 DPlan Algorithm .....	57

3.1.3	Efficiency Concerns .....	58
3.1.4	Example Problems .....	59
3.2	Optimal Full Universal Plans .....	64
3.3	Termination, Soundness, Completeness .....	66
3.3.1	Termination of DPlan .....	66
3.3.2	Operator Restrictions .....	67
3.3.3	Soundness and Completeness of DPlan .....	70
<b>4.</b>	<b>Integrating Function Application in Planning .....</b>	<b>71</b>
4.1	Motivation .....	71
4.2	Extending Strips to Function Applications .....	74
4.3	Extensions of FPlan .....	79
4.3.1	Backward Operator Application .....	79
4.3.2	Introducing User-Defined Functions .....	81
4.4	Examples .....	82
4.4.1	Planning with Resource Variables .....	83
4.4.2	Planning for Numerical Problems .....	85
4.4.3	Functional Planning for Standard Problems .....	87
4.4.4	Mixing ADD/DEL Effects and Updates .....	88
4.4.5	Planning for Programming Problems .....	88
4.4.6	Constraint Satisfaction and Planning .....	90
<b>5.</b>	<b>Conclusions and Further Research .....</b>	<b>93</b>
5.1	Comparing DPlan with the State of the Art .....	93
5.2	Extensions of DPlan .....	94
5.3	Universal Planning versus Incremental Exploration .....	95

---

## Part II. Inductive Program Synthesis

---

<b>6.</b>	<b>Automatic Programming .....</b>	<b>99</b>
6.1	Overview of Automatic Programming Research .....	100
6.1.1	AI and Software Engineering .....	100
6.1.2	Approaches to Program Synthesis .....	102
6.1.3	Pointers to Literature .....	109
6.2	Deductive Approaches .....	110
6.2.1	Constructive Theorem Proving .....	110
6.2.2	Program Transformation .....	115
6.3	Inductive Approaches .....	124
6.3.1	Foundations of Induction .....	124
6.3.2	Genetic Programming .....	134
6.3.3	Inductive Logic Programming .....	140
6.3.4	Inductive Functional Programming .....	150

6.4	Final Comments .....	164
6.4.1	Inductive versus Deductive Synthesis .....	164
6.4.2	Inductive Functional versus Logic Programming .....	165
<b>7.</b>	<b>Folding of Finite Program Terms .....</b>	<b>167</b>
7.1	Terminology and Basic Concepts .....	168
7.1.1	Terms and Term Rewriting .....	168
7.1.2	Patterns and Anti-unification .....	171
7.1.3	Recursive Program Schemes .....	172
7.2	Synthesis of RPSs from Initial Programs .....	182
7.2.1	Folding and Fixpoint Semantics .....	182
7.2.2	Characteristics of RPSs .....	182
7.2.3	The Synthesis Problem .....	185
7.3	Solving the Synthesis Problem .....	185
7.3.1	Constructing Segmentations .....	186
7.3.2	Constructing a Program Body .....	195
7.3.3	Dealing with Further Subprograms .....	198
7.3.4	Finding Parameter Substitutions .....	206
7.3.5	Constructing an RPS .....	215
7.4	Example Problems .....	220
7.4.1	Time Effort of Folding .....	220
7.4.2	Recursive Control Rules .....	222
<b>8.</b>	<b>Transforming Plans into Finite Programs .....</b>	<b>227</b>
8.1	Overview of Plan Transformation .....	228
8.1.1	Universal Plans .....	228
8.1.2	Introducing Data Types and Situation Variables .....	228
8.1.3	Components of Plan Transformation .....	229
8.1.4	Plans as Programs .....	229
8.1.5	Completeness and Correctness .....	231
8.2	Transformation and Type Inference .....	231
8.2.1	Plan Decomposition .....	231
8.2.2	Data Type Inference .....	233
8.2.3	Introducing Situation Variables .....	234
8.3	Plans over Sequences of Objects .....	235
8.4	Plans over Sets of Objects .....	240
8.5	Plans over Lists of Objects .....	246
8.5.1	Structural and Semantic List Problems .....	246
8.5.2	Synthesizing ‘Selection-Sort’ .....	248
8.5.3	Concluding Remarks on List Problems .....	257
8.6	Plans over Complex Data Types .....	259
8.6.1	Variants of Complex Finite Programs .....	259
8.6.2	The ‘Tower’ Domain .....	260
8.6.3	Tower of Hanoi .....	267

<b>9. Conclusions and Further Research</b>	271
9.1 Combining Planning and Program Synthesis	271
9.2 Acquisition of Problem Solving Strategies	272
9.2.1 Learning in Problem Solving and Planning	272
9.2.2 Three Levels of Learning	273

---

## Part III. Schema Abstraction

---

<b>10. Analogical Reasoning and Generalization</b>	279
10.1 Analogical and Case-Based Reasoning	279
10.1.1 Characteristics of Analogy	279
10.1.2 Sub-processes of Analogical Reasoning	281
10.1.3 Transformational versus Derivational Analogy	282
10.1.4 Quantitive and Qualitative Similarity	283
10.2 Mapping Simple Relations or Complex Structures	284
10.2.1 Proportional Analogies	284
10.2.2 Causal Analogies	286
10.2.3 Problem Solving and Planning by Analogy	286
10.3 Programming by Analogy	288
10.4 Pointers to Literature	290
<b>11. Structural Similarity in Analogical Transfer</b>	291
11.1 Analogical Problem Solving	291
11.1.1 Mapping and Transfer	292
11.1.2 Transfer of Non-isomorphic Source Problems	293
11.1.3 Structural Representation of Problems	294
11.1.4 Non-isomorphic Variants in a Water Redistribution Domain	296
11.1.5 Measurement of Structural Overlap	300
11.2 Experiment 1	300
11.2.1 Method	302
11.2.2 Results and Discussion	303
11.3 Experiment 2	305
11.3.1 Method	306
11.3.2 Results and Discussion	307
11.4 General Discussion	309
<b>12. Programming by Analogy</b>	311
12.1 Program Reuse and Program Schemes	311
12.2 Restricted 2nd-order Anti-unification	312
12.2.1 Recursive Program Schemes Revisited	312
12.2.2 Anti-unification of Program Terms	314

12.3 Retrieval Using Term Subsumption .....	316
12.3.1 Term Subsumption .....	316
12.3.2 Empirical Evaluation .....	317
12.3.3 Retrieval from Hierarchical Memory .....	318
12.4 Generalizing Program Schemes .....	319
12.5 Adaptation of Program Schemes .....	320
<b>13. Conclusions and Further Research .....</b>	<b>323</b>
13.1 Learning and Applying Abstract Schemes .....	323
13.2 A Framework for Learning from Problem Solving .....	324
13.3 Application Perspective .....	325
<b>Bibliography .....</b>	<b>327</b>

---

## Appendices

---

<b>A. Implementation Details .....</b>	<b>343</b>
A.1 Short History of DPlan .....	343
A.2 Modules of DPlan .....	345
A.3 DPlan Specifications .....	345
A.4 Development of Folding Algorithms .....	347
A.5 Modules of TFold .....	348
A.6 Time Effort of Folding .....	349
A.7 Main Components of Plan-Transformation .....	349
A.8 Plan Decomposition .....	350
A.9 Introduction of Situation Variables .....	351
A.10 Number of MSTs in a DAG .....	351
A.11 Extracting Minimal Spanning Trees from a DAG .....	352
A.12 Regularizing a Tree .....	353
A.13 Programming by Analogy Algorithms .....	355
<b>B. Concepts and Proofs .....</b>	<b>357</b>
B.1 Fixpoint Semantics .....	357
B.2 Proof: Maximal Subprogram Body .....	360
B.3 Proof: Uniqueness of Substitutions .....	365
<b>C. Sample Programs and Problems .....</b>	<b>369</b>
C.1 Fibonacci with Sequence Referencing Function .....	369
C.2 Inducing ‘Reverse’ with Golem .....	370
C.3 Finite Program for ‘Unstack’ .....	373
C.4 Recursive Control Rules for the ‘Rocket’ Domain .....	375
C.5 The ‘Selection Sort’ Domain .....	376
C.6 Recursive Control Rules for the ‘Tower’ Domain .....	377

C.7 Water Jug Problems .....	383
C.8 Example RPSs .....	388
<b>Index</b> .....	391

# List of Figures

1.1	Analogical Problem Solving and Learning .....	5
1.2	Main Components of the Synthesis System .....	6
2.1	A Simple Blocks-World .....	14
2.2	A Strips Planning Problem in the Blocks-World .....	18
2.3	An Alternative Representation of the Blocks-World .....	19
2.4	Representation of a Blocks-World Problem in PDDL-Strips .....	21
2.5	Blocks-World Domain with Equality Constraints and Conditioned Effects .....	22
2.6	Representation of a Blocks-World Problem in Situation Calculus .	26
2.7	A Forward Search Tree for Blocks-World .....	31
2.8	A Backward Search Tree for Blocks-World .....	36
2.9	Goal-Regression for Blocks-World .....	37
2.10	The Sussman Anomaly .....	39
2.11	Part of a Planning Graph as Constructed by Graphplan .....	43
2.12	Representation of the Boolean Formula $f(x_1, x_2) = x_1 \wedge x_2$ as OBDD	47
3.1	The <i>Clearblock</i> DPlan Problem .....	60
3.2	DPlan Plan for <i>Clearblock</i> .....	60
3.3	<i>Clearblock</i> with a Set of Goal States .....	60
3.4	The DPlan <i>Rocket</i> Problem .....	61
3.5	Universal Plan for <i>Rocket</i> .....	62
3.6	The DPlan <i>Sorting</i> Problem .....	62
3.7	Universal Plan for <i>Sorting</i> .....	63
3.8	The DPlan <i>Hanoi</i> Problem .....	63
3.9	Universal Plan for <i>Hanoi</i> .....	64
3.10	Universal Plan for <i>Tower</i> .....	64
3.11	Minimal Spanning Tree for <i>Rocket</i> .....	67
4.1	<i>Tower of Hanoi</i> (a) Without and (b) With Function Application .	73
4.2	<i>Tower of Hanoi</i> in Functional Strips (Geffner, 1999) .....	74
4.3	A Plan for <i>Tower of Hanoi</i> .....	80
4.4	<i>Tower of Hanoi</i> with User-Defined Functions .....	82
4.5	A Problem Specification for the <i>Airplane</i> Domain .....	83
4.6	Specification of the Inverse Operator $fly^{-1}$ for the <i>Airplane</i> Domain	84



4.7	A Problem Specification for the <i>Water Jug</i> Domain . . . . .	86
4.8	A Plan for the <i>Water Jug</i> Problem . . . . .	86
4.9	Blocks-World Operators with Indirect Reference and Update . . . . .	89
4.10	Specification of <i>Selection Sort</i> . . . . .	89
4.11	<i>Lightmeal</i> in Constraint Prolog . . . . .	91
4.12	Problem Specification for <i>Lightmeal</i> . . . . .	91
6.1	Programs Represent Concepts and Skills . . . . .	125
6.2	Construction of a Simple Arithmetic Function (a) and an Even-2-Parity Function (b) Represented as a Labeled Tree with Ordered Branches (Koza, 1992, figs. 6.1, 6.2) . . . . .	135
6.3	A Possible Initial State, an Intermediate State, and the Goal State for Block Stacking (Koza, 1992, figs. 18.1, 18.2) . . . . .	137
6.4	Resulting Programs for the Block Stacking Problem (Koza, 1992, chap. 18.1) . . . . .	139
6.5	$\theta$ -Subsumption Equivalence and Reduced Clauses . . . . .	142
6.6	$\theta$ -Subsumption Lattice . . . . .	143
6.7	An Inverse Linear Derivation Tree (Lavrač and Džeroski, 1994, pp. 46) . . . . .	144
6.8	Part of a Refinement Graph (Lavrač and Džeroski, 1994, p. 56) . . . . .	146
6.9	Specifying Modes and Types for Predicates . . . . .	147
6.10	Learning Function <i>unpack</i> from Examples . . . . .	152
6.11	Traces for the <i>unpack</i> Example . . . . .	153
6.12	Result of the First Synthesis Step for <i>unpack</i> . . . . .	155
6.13	Recurrence Relation for <i>unpack</i> . . . . .	155
6.14	Traces for the <i>reverse</i> Problem . . . . .	159
6.15	Synthesis of a Regular Lisp Program . . . . .	162
6.16	Recursion Formation with <i>Tinker</i> . . . . .	164
7.1	Example Term with Exemplaric Positions of Sub-terms . . . . .	170
7.2	Example First Order Pattern . . . . .	172
7.3	Anti-Unification of Two Terms . . . . .	173
7.4	Examples for Terms Belonging to the Language of an RPS and of a Subprogram of an RPS . . . . .	176
7.5	Unfolding Positions in the Third Unfolding of <i>Fibonacci</i> . . . . .	179
7.6	Valid Recurrent Segmentation of <i>Mod</i> . . . . .	187
7.7	Initial Program for <i>ModList</i> . . . . .	190
7.8	Identifying Two Recursive Subprograms in the Initial Program for <i>ModList</i> . . . . .	200
7.9	Inferring a Sub-Program Scheme for <i>ModList</i> . . . . .	201
7.10	The Reduced Initial Tree of <i>ModList</i> . . . . .	202
7.11	Substitutions for <i>Mod</i> . . . . .	207
7.12	Steps for Calculating a Subprogram . . . . .	216
7.13	Overview of Inducing an RPS . . . . .	218
7.14	Time Effort for Unfolding/Folding <i>Factorial</i> . . . . .	220

7.15	Time Effort for Unfolding/Folding <i>Fibonacci</i> .....	221
7.16	Time Effort Calculating Valid Recurrent Segmentations and Substitutions for <i>Factorial</i> .....	221
7.17	Initial Tree for <i>Clearblock</i> .....	223
7.18	Initial Tree for <i>Tower of Hanoi</i> .....	224
8.1	Induction of Recursive Functions from Plans .....	227
8.2	Examples of Uniform Sub-Plans .....	232
8.3	Uniform Plans as Subgraphs .....	233
8.4	Generating the <i>Successor</i> -Function for a Sequence .....	236
8.5	The <i>Unstack</i> Domain and Plan .....	237
8.6	Protocol for <i>Unstack</i> .....	238
8.7	Introduction of Data Type <i>Sequence</i> in <i>Unstack</i> .....	238
8.8	LISP-Program for <i>Unstack</i> .....	239
8.9	Partial Order of <i>Set</i> .....	240
8.10	Functions Inferred/Provided for <i>Set</i> .....	242
8.11	Sub-Plans of <i>Rocket</i> .....	244
8.12	Introduction of the Data Type <i>Set</i> (a) and Resulting Finite Program (b) for the <i>Unload-All</i> Sub-Plan of <i>Rocket</i> ( $\Omega$ denotes “undefined”) .....	244
8.13	Protocol of Transforming the <i>Rocket</i> Plan .....	245
8.14	Partial Order (a) and Total Order (b) of Flat Lists over Numbers .....	247
8.15	A Minimal Spanning Tree Extracted from the <i>SelSort</i> Plan .....	252
8.16	The Regularized Tree for <i>SelSort</i> .....	254
8.17	Introduction of a “Semantic” Selector Function in the Regularized Tree .....	256
8.18	LISP-Program for <i>SelSort</i> .....	258
8.19	Abstract Form of the Universal Plan for the Four-Block <i>Tower</i> ...	264
9.1	Three Levels of Generalization .....	274
10.1	Mapping of Base and Target Domain .....	281
10.2	Example for a Geometric-Analogy Problem (Evans, 1968, p. 333) ..	285
10.3	Context Dependent Descriptions in Proportional Analogy (O’Hara, 1992) .....	285
10.4	The Rutherford Analogy (Gentner, 1983) .....	286
10.5	Base and Target Specification (Dershowitz, 1986) .....	288
11.1	Types and degrees of structural overlap between source and target Problems .....	295
11.2	A water redistribution problem .....	297
11.3	Graphs for the equations $2 \cdot x + 5 = 9$ (a) and $3 \cdot x + (6 - 2) = 16$ (b)	301
12.1	Adaptation of <i>Sub</i> to <i>Add</i> .....	321

C.1	Universal Plan for Sorting Lists with Three Elements . . . . .	376
C.2	Minimal Spanning Trees for Sorting Lists with Three Elements . .	377
C.3	Minimal Spanning Trees for Sorting Lists with Three Elements . .	378
C.4	Minimal Spanning Trees for Sorting Lists with Three Elements . .	378

# List of Tables

2.1	Informal Description of Forward Planning .....	29
2.2	A Simple Forward Planner .....	31
2.3	Number of States in the Blocks-World Domain .....	33
2.4	Planning as Model Checking Algorithm (Giunchiglia, 1999, fig. 4) .....	47
3.1	Abstract DPlan Algorithm .....	58
4.1	Database with Distances between Airports .....	84
4.2	Performance of FPlan: <i>Tower of Hanoi</i> .....	88
4.3	Performance of FPlan: <i>Selection Sort</i> .....	90
6.1	Different Specifications for <i>Last</i> .....	103
6.2	Training Examples .....	127
6.3	Background Knowledge .....	127
6.4	Fundamental Results of Language Learnability (Gold, 1967, tab. 1) .....	132
6.5	Genetic Programming Algorithm (Koza, 1992, p. 77) .....	136
6.6	Calculation the <i>Fibonacci</i> Sequence .....	139
6.7	Learning the <i>daughter</i> Relation .....	141
6.8	Calculating an <i>rlgg</i> .....	143
6.9	Simplified MIS-Algorithm (Lavrač and Džeroski, 1994, pp.54) ....	145
6.10	Background Knowledge and Examples for Learning <i>reverse(X, Y)</i> .	149
6.11	Constructing Traces from I/O Examples .....	153
6.12	Calculating the Form of an S-Expression .....	154
6.13	Constructing a Regular Lisp Program by Function Merging .....	161
7.1	A Sample of Function Symbols .....	169
7.2	Example for an RPS .....	174
7.3	Recursion Points and Substitution Terms for the <i>Fibonacci</i> Function	178
7.4	Unfolding Positions and Unfolding Indices for <i>Fibonacci</i> .....	180
7.5	Example of Extrapolating an RPS from an Initial Program .....	183
7.6	Calculation of the Next Position on the Right .....	195
7.7	<i>Factorial</i> and Its Third Unfolding with Instantiation <i>succ(succ(0))</i> (a) and <i>pred(3)</i> (b) .....	196
7.8	Segments of the Third Unfolding of <i>Factorial</i> for Instantiation <i>succ(succ(0))</i> (a) and <i>pred(3)</i> (b) .....	196

7.9	Anti-Unification for Incomplete Segments . . . . .	198
7.10	Variants of Substitutions in Recursive Calls . . . . .	207
7.11	Testing whether Substitutions are Uniquely Determined . . . . .	209
7.12	Testing whether a Substitution is Recurrent . . . . .	210
7.13	Testing the Existence of Sufficiently Many Instances for a Variable	211
7.14	Determining Hidden Variables . . . . .	213
7.15	Calculating Substitution Terms of a Variable in a Recursive Call .	214
7.16	Equality of Subprograms . . . . .	219
7.17	RPS for <i>Factorial</i> with Constant Expression in <i>Main</i> . . . . .	222
8.1	Introducing <i>Sequence</i> . . . . .	235
8.2	Linear Recursive Functions . . . . .	239
8.3	Introducing <i>Set</i> . . . . .	241
8.4	Structural Functions over Lists . . . . .	247
8.5	Introducing <i>List</i> . . . . .	248
8.6	Dealing with Semantic Information in Lists . . . . .	249
8.7	Functional Variants for <i>Selection-Sort</i> . . . . .	250
8.8	Extract an MST from a DAG . . . . .	253
8.9	Regularization of a Tree . . . . .	253
8.10	Structural Complex Recursive Functions . . . . .	260
8.11	Transformation Sequences for Leaf-Nodes of the <i>Tower</i> Plan for Four Blocks . . . . .	265
8.12	Power-Set of a List, Set of Lists . . . . .	266
8.13	Control Rules for <i>Tower</i> Inferred by Decision List Learning . . . . .	267
8.14	A <i>Tower of Hanoi</i> Program . . . . .	268
8.15	A <i>Tower of Hanoi</i> Program for Arbitrary Starting Constellations .	269
10.1	Kinds of Predicates Mapped in Different Types of Domain Com- parison (Gentner, 1983, Tab. 1, extended) . . . . .	280
10.2	Word Algebra Problems (Reed et al., 1990) . . . . .	287
11.1	Relevant information for solving the source problem . . . . .	299
11.2	Results of Experiment 1 . . . . .	304
11.3	Results of Experiment 2 . . . . .	308
12.1	A Simple Anti-Unification Algorithm . . . . .	315
12.2	An Algorithm for Retrieval of RPSs . . . . .	317
12.3	Results of the similarity rating study . . . . .	318
12.4	Example Generalizations . . . . .	320