RWTH Aachen

Department of Computer Science
*Technical Report*

# SAT Encodings: From Constraint-Based Termination Analysis to Circuit Synthesis

Carsten Fuhs

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2011-17

RWTH Aachen · Department of Computer Science · September 2012

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

`http://aib.informatik.rwth-aachen.de/`

# SAT Encodings: From Constraint-Based Termination Analysis to Circuit Synthesis

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

**Diplom-Informatiker**
**Carsten Fuhs**
aus
Düren

Berichter: Prof. Dr. Jürgen Giesl
Prof. Dr. Michael Codish

Tag der mündlichen Prüfung: 21. Dezember 2011

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

Carsten Fuhs
Lehr- und Forschungsgebiet Informatik 2
fuhs@informatik.rwth-aachen.de

---

Aachener Informatik-Bericht AIB-2011-17

Herausgeber: Fachgruppe Informatik
RWTH Aachen University
Ahornstr. 55
52074 Aachen
GERMANY

ISSN 0935-3232

# Abstract

*Termination* is one of the most prominent undecidable problems in computer science. At the same time, the problem whether a given program terminates for all inputs is sufficiently important for the area of program verification to spur decades-long efforts in developing sufficient criteria for concluding termination. In the last decade, the focus of this research has been on automation, giving rise to several powerful fully automatic termination provers. However, the search problems that arise during the synthesis of a successful termination proof are typically NP-complete.

To tackle these algorithmic challenges, over the last years a two-stage process has turned out to be extremely successful in practice: First encode the arising problem instance to the *satisfiability problem of propositional logic (SAT),* and then invoke a state-of-the-art SAT solver on this SAT instance. The solution found by the SAT solver is then used in the termination proof.

While in the worst case still prohibitive due to NP-completeness of SAT, this approach has increased performance on practical problem instances for existing termination techniques by orders of magnitude. At the same time, the approach has also made new automated techniques possible that were out of reach before. This thesis contributes efficient SAT-based automation both for several existing termination techniques and also for new techniques that we have developed in the dependency pair framework for termination analysis of term rewriting.

The usefulness of SAT encodings goes beyond the field of termination analysis. We have transferred the approach used for termination techniques to the—at first glance—quite distinct application domain of circuit synthesis, allowing us to obtain a provably optimal implementation of a part of the Advanced Encryption Standard.

The contributions of this thesis are implemented within our fully automated termination prover AProVE. The significance of our results is demonstrated also by AProVE reaching the highest scores in the annual International Termination Competitions of 2007 – 2011. At these competitions, the leading automated termination analysis tools try to prove or disprove termination of programs originating from various areas of computer science.

# Acknowledgments

First of all, I would like to thank my PhD advisor Jürgen Giesl. Over the years I have been working in his group, he has always been at the same time trusting, demanding, and supporting. He granted me the freedom to expand to new directions of research, and there was always time for discussion.

I am also very grateful to Mike Codish for agreeing to be the second supervisor of this thesis. We had many enriching discussions, we shared many a cup of coffee all around the world, and our joint work and mutual research visits have always turned out to be very satisfying.

I would also like to thank my colleague and friend Peter Schneider-Kamp for going the way together with me. Together we explored new topics and directions, which broadened both our perspectives, and our mutual research visits have always been very rewarding. Additionally, I am also thankful to Peter for co-developing the SAT framework of AProVE.

Many thanks go as well to René Thiemann, with whom I enjoyed several detailed discussions on the intricacies of the dependency pair framework, and to Stephan Swiderski, who often provided interesting and helpful perspectives.

My thanks go also to my office mates Fabian Emmes and Thomas Ströder for providing a relaxed yet at the same time also productive atmosphere, and also to my colleagues Marc Brockschmidt, Carsten Otto, and Martin Plücker for many interesting discussions.

Especially, I would like to thank Thomas Ströder, Stephan Falke, and Peter Schneider-Kamp for proof-reading preliminary versions of this thesis.

I am also grateful to my co-authors for the fruitful cooperation and to the members of the AProVE team for their contributions. Likewise, I wish to thank the members of the MOVES and the Theory of Hybrid Systems groups for the pleasant working atmosphere. Moreover, I enjoyed the presence of and the work with our international guests Beatriz Alarcón, Yoav Fekete, Raúl Gutiérrez, Cynthia Kop, and Fausto Spoto.

Finally, I would like to thank my family for supporting me and for being there for me during all those years.

*Carsten Fuhs*

# Contents


**1 Introduction** **1**

**2 Preliminaries** **23**

2.1 Term Rewriting . . . . . 23

2.2 The Dependency Pair Framework . . . . . 25

2.3 Weakly Monotone Algebras and Polynomial Interpretations . . . . . 30

**3 Polynomials with Negative Constants** **37**

3.1 Polynomial Interpretations with a Negative Constant . . . . . 38

3.2 SMT-Based Automation . . . . . 41

3.3 A Necessary Criterion for Negative Constants . . . . . 47

3.4 Experiments . . . . . 53

3.5 Summary and Outlook . . . . . 55

**4 Maximal Termination** **59**

4.1 Max-Polynomial Interpretations . . . . . 60

4.2 SMT-Based Automation . . . . . 61

4.3 Shape Heuristics and Optimizations . . . . . 65

4.4 Experiments . . . . . 73

4.5 Summary and Outlook . . . . . 74

**5 SAT Encodings for Arctic Termination Revisited** **77**

5.1 Arctic Interpretations . . . . . 78

5.2 A Binary SAT Encoding for Arctic Constraints . . . . . 85

5.3 A Unary SAT Encoding for Arctic Constraints . . . . . 90

5.4 Related Work . . . . . 94

5.5 Experiments . . . . . 95

5.6 Summary and Outlook . . . . . 101

**6 Lazy Abstraction for Size-Change Termination 103**
6.1 Size-Change Termination and Dependency Pairs . . . 105
6.2 Tuple-Typed DP Problems . . . 107
6.3 Approximating SCT in NP . . . 112
6.4 A Challenge Example . . . 125
6.5 SAT-Based Automation . . . 127
6.6 Experiments . . . 129
6.7 Summary and Outlook . . . 131

**7 SAT Encodings for Optimal XOR Circuits 133**
7.1 Linear Straight-Line Programs . . . 134
7.2 Encoding to Propositional Logic . . . 139
7.3 From Decision Problem to Optimization . . . 144
7.4 Case Study: Advanced Encryption Standard . . . 145
7.5 Handling the UNSAT Case . . . 148
7.6 Summary and Outlook . . . 150

**8 Conclusion 153**

**Bibliography 157**

**Index 179**

# 1 Introduction

Termination analysis of programs is a major task in the area of software verification. Although the problem whether a given program will terminate on all inputs is in general not even semi-decidable, in recent years there has been a wealth of research in correct, yet incomplete automated methods for showing program termination. This approach is considered worthwhile for practical application since the termination argument the programmer has in mind (implicitly or explicitly) often is sufficiently simple to make a fully automated termination proof feasible.

Termination analysis has been a topic of active research for several programming paradigms, including logic programming [BCG+07, CLS05, DD94, LMS03, NDGS11, NGSD08, SGN10, SGS+10, Sma04], functional programming [Abe04, Gie95, GWB98, LJB01, MV06, PS97, SJ05, Wal94, Xi02], and also imperative programming [BCDO06, BMS05, CS02, CPR05, CPR09, PR04a, PR04b, Tiw04, Tur49].

One of the most prevalent areas for research in termination analysis, however, is *term rewriting*. The reason is that while term rewriting is a Turing-complete calculus which allows for a convenient representation of user-defined data-structures, at the same time it provides a suitably simple mathematical model for analysis. Indeed, in recent years, there have been ongoing successful efforts to use term rewriting as back-end language for termination analysis for programs written in widely used programming languages. These approaches are based on automated translations to *term rewrite systems (TRSs)*, and they have turned out to be viable for languages from different programming paradigms, such as logical programming languages like Prolog (cf., e.g., [Ohl01, SGST09]), functional languages like Haskell [GSST06, GRS+11] or the ML-like functional language used in the proof assistant Isabelle/HOL ([KST+11], cf. also [NPW02]), and recently also imperative languages like the object-oriented language Java Bytecode [OBEG10, BOEG10, BOG11] or the compiler intermediate language LLVM-IR [FKS11a, FKS11b].

Therefore, any advance for automated termination analysis for TRSs also has an impact on termination analysis for programming languages. Keeping this in mind, the primary goal of the present thesis is to improve upon the state of the art of termination analysis, with respect to both efficiency and power.

Termination analysis for TRSs has been studied for many years now. Here, initially the focus of research was mainly to find *reduction orders* [MN70] to prove termination directly, cf., e.g., [Der82, Der87, KB70, KL80, Lan79, Les83, Ste95]. Later, transforma-

tional techniques such as the *dependency pair (DP) approach* [AG00] or *semantic labeling* [Zan95] were developed. These approaches are designed to yield simpler constraints for a termination proof via orders on terms.

Building on [AG00], in recent work Giesl *et al.* propose the *dependency pair (DP) framework* [GTS05a, HM05, GTSF06, Thi07]. This framework allows for modular decomposition of termination problems and an incremental application and combination of termination techniques (in this setting called *DP processors*).

With the beginning of the $21^{st}$ century, the focus in termination analysis shifted more and more towards automation. Since 2004, each year the *International Termination Competition*[1] has taken place with many different participating termination tools.[2]

Starting from termination analysis of term rewriting, the Termination Competition has in the mean time extended its scope significantly. This extension has taken place with respect to the supported input languages (in addition to various flavors of rewriting, nowadays also inputs from logical languages like Prolog, higher-order functional languages like Haskell, and imperative languages like Java Bytecode are used as benchmarks at the competition). Also for the statement that is to be shown, extensions have been introduced. After initially, only the question "Is this program terminating on all inputs?" was asked, nowadays there are dedicated categories where this question has been refined to the problem statement "Give upper/lower bounds on the asymptotic complexity of the program".

The benchmarks used for the competition are collected in the *Termination Problem Data Base (TPDB)*,[3] which nowadays contains several thousands of termination problems from different programming paradigms.

At the Termination Competition, each termination tool is provided with a termination problem and a timeout of 60 seconds to provide an answer to the statement on termination (or complexity) together with a human- or machine-readable termination proof. Since soundness is crucial for verification tools and since thus a wrong answer on a termination problem leads to disqualification, it is naturally vital to use only provably sound termination techniques for the proof steps.

To ensure soundness, since 2007 also categories with *certified* termination proofs are part of the competition. In a two-stage-approach, first the participating termination tools provide machine-readable termination proofs as certificates in a standardized format.[4] In the second stage, the participating certification tools are invoked on these certificates and check whether the given certificates indeed are valid termination proofs for the original

[1] For details, see also: `http://termination-portal.org/wiki/Termination_Competition`

[2] See `http://termination-portal.org/wiki/Category:Tools` for an up-to-date list of termination tools. As of this writing, 26 tools are reported there.

[3] More information on the TPDB is available at: `http://termination-portal.org/wiki/TPDB`

[4] Nowadays the standard *Certification Problem Format (CPF)* is used. For more information on CPF see also: `http://cl-informatik.uibk.ac.at/software/cpf/`

TRSs. These *proof certification tools* are based on a trusted theorem prover such as Coq [Coq10] or Isabelle [NPW02]. Examples for projects that have sent certification tools to the termination competition are A3PAT [CCF+07, CPU+10] with the tool C*i*ME [CCF+11], CoLoR [BK11] with the tool Rainbow (both tools invoke the theorem prover Coq for analysis of the termination proof certificates), and IsaFoR [TS09] with the tool CeTA, which is generated from Isabelle via code extraction.

Since the beginning of the annual termination competition in 2004, our automated termination prover AProVE [GST06] has been reaching the highest score for most categories concerned with term rewriting. AProVE has also reached the highest scores for functional and logic programs, and it has reached top positions in the categories for Java Bytecode. These results were made possible among other things also by the efficient SAT-based automation of the termination techniques that ultimately solve the rewriting problems obtained from the translations of the input programs. In particular, AProVE uses *integer term rewrite systems (ITRSs)* [FGP+09] (i.e., TRSs where also pre-defined operations for the integers can be used) as a back-end translation target for the analysis of programming languages. This thesis provides a precursor of the main termination technique used for ITRSs. Moreover, in the years 2009 – 2011 AProVE scored highest in the category *SRS Standard*, where *string rewrite systems (SRSs)* are analyzed for termination. This category is renowned for its particularly hard problems, and without the contributions of this thesis, AProVE would not have risen to the top in this category.

## Historical Development of Constraint-Based Termination Analysis

However, one question that is crucial for automation remains: How to solve the search problems associated with finding the termination argument in the individual proof steps? Since these problems often are NP-hard, devising corresponding algorithms that are efficient at least on practical problem instances is far from trivial.

Recently, it has become a very active topic of research to solve such problems not via dedicated algorithms developed from scratch, but via *encodings* to a suitable class of constraints and subsequently using *satisfiability solvers (SAT solvers)*. This way, the search for proof steps using certain classes of well-founded orders is reduced to a satisfiability problem. Here, satisfiability is meant in a broad sense: Of course, satisfiability of propositional logic (this quintessential NP-complete decision problem [Coo71] is commonly denoted by the *SAT problem*) is a target for many translations. In addition, also instances of the satisfiability problem for *Pseudo-Boolean (PB) constraints* (also known as 0-1 linear programming; linear integer inequalities where variables may only be assigned 0 or 1) and instances of *satisfiability modulo theories (SMT)* problems (first-order formulas where the semantics of certain function and predicate symbols is fixed by a background theory) are reduction targets.

In 1999, Kondo and Kurihara pioneered the area of satisfiability-based automated termination analysis. In [KK99a, KK99b], they describe an encoding for the class of *lexicographic path orders (LPOs)* [KL80] for a given TRS to a (reduced ordered) *binary decision diagram (BDD)* (cf. also [KK04]). This BDD is used to represent the Boolean function (or formula) that maps from Boolean variables parameterizing the *precedence* of an LPO to the truth value of the following question: "Does this LPO prove termination of the given TRS?" By construction of BDDs, then satisfiability can trivially be checked for an explicitly given BDD via a disequality test with the zero BDD.

However, it was not until 2006 that the topic of automating termination techniques by encodings as Boolean formulas (or related formalisms) received broader attention by the termination community. Starting with these first papers [ACG+06, CLSS06, CLS06, CSL+06, EWZ06, HW06, ZM06], it has become widely accepted to avoid the explicit construction of BDDs, which can be a time-consuming process. Instead, a representation of Boolean formulas as SAT instances in conjunctive normal form is chosen, This way, the burden of the satisfiability check is shifted from purely symbolical level to dedicated *SAT solvers*.

In later work, also Pseudo-Boolean constraints [FGM+07, ZM07, ZHM09, Zan09] and SMT instances [BLN+09, FK09, ZHM09, BLO+12, FKS11a] are targeted; the latter usually being solved by SMT tools, which use dedicated internal theory solvers in addition to a SAT front-end for the Boolean skeleton of the formula. Note that the SMT formalism also allows for higher-level problem descriptions which can nevertheless still boil down to standard SAT encodings. For instance, for the case of *non-linear integer arithmetic (NIA)* and the corresponding SMT problem class *SAT modulo non-linear integer arithmetic (SMT-NIA)*, earlier SAT encodings [FGM+07, Fuh07, EWZ08] can be harnessed to render all required theory knowledge explicit on Boolean level via an *eager encoding* of the atomic subformulas. Via this *bit-blasting* approach, now also standard propositional SAT solvers suffice to provide answers for formulas from a richer language. This layered approach turns out to be beneficial for ease of description and implementation of many later works (e.g., [FGM+07, GTSS07, FGM+08, ZM08, FGP+09, ZM09, CFGS10, NDGS11]), where SAT encodings for SMT problems can be used as "black boxes". It should be noted, though, that satisfiability of non-linear integer arithmetic is undecidable due to the undecidability of Hilbert's $10^{th}$ problem [Mat70]. Hence, these encodings only provide a sufficient criterion for satisfiability of non-linear integer arithmetic, which is nonetheless very powerful in practice.

## Techniques

A common theme in almost all encoded termination proving techniques is that satisfiability of the encoded instance implies that at least a part of the termination problem (a

TRS or a dependency pair problem) can be deleted and need not be considered for further analysis any more. This is done by using *orders* on terms (or, more generally speaking, on the program states). Usually one searches for an order that orients all rules of a termination problem weakly and at least one rule strictly, allowing to delete the strictly oriented rules. This way, to advance a termination proof, it suffices to solve a formula over (weak or strict) inequalities over terms.

Broadly, the *termination techniques* following this scheme that have been encoded to satisfiability problems in recent years can be grouped as follows:

- As mentioned earlier, the first encodings for precedence-based *path orders* like the lexicographic path order via Boolean functions by Kurihara and Kondo [KK99a, KK99b, KK04] use a BDD representation. Via encodings to satisfiability instances in conjunctive normal form, this approach has been improved and extended in several papers since 2006 by Annov, Codish, Giesl, Lagoon, Schneider-Kamp, Stuckey, and Thiemann to encompass quasi-precedences [CLS06, CLS08], argument filterings [CSL$^+$06], and permutations as well as multiset comparison [STA$^+$07]. These publications finally culminate in a SAT-based implementation of full *recursive path orders* [Der82, KL80, Les83] also combined with argument filterings and quasi-precedences [Sch08, CGST12]. As these publications report, the SAT-based approaches outperform dedicated solving techniques to search for path orders by orders of magnitude on practical example collections.

  In [AY09] Aoto and Yamada mention an adaption of the encodings from [CLS06, STA$^+$07] to the setting of *simply-typed S-expression rewriting systems (STSRSs)* [Yam01], a framework for higher-order rewriting without binders.

  In the context of path orders, one should also mention the *polynomial path order (POP*)* [AM08]. This restriction of the multiset path order can be used to show not only termination, but also polynomial bounds on the *innermost runtime complexity* of a given TRS, i.e., the length of the longest derivation from a *constructor-based term* $t$ [AM08] as a function of its size. (Here, a constructor-based term is a term $t = f(t_1, \ldots, t_n)$ such that the only occurrence of a non-constructor-symbol is at the root of $t$; in later work [AM09] also called a *basic term*.) For automation, the authors adapt previous encodings for precedences [ZHM07] and multiset comparison [STA$^+$07] to the setting of POP$^*$.

  Likewise, a termination proof using the *exponential path order (EPO*)* [AEM11a] implies an exponential upper bound on the innermost runtime complexity of a given TRS. In the corresponding technical report [AEM11b], the authors describe automation of EPO$^*$ by means of a SAT encoding.

- *Knuth-Bendix orders (KBOs)* are similar to classic path orders since also here we have an explicit precedence on function symbols. In addition, KBOs also have

an arithmetic aspect since each function symbol has an associated non-negative *weight*. For this class of orders, Korovin and Voronkov [KV03] provide a dedicated polynomial-time algorithm. Successful later work on encoding KBOs has been conducted by Hirokawa, Middeldorp, and Zankl, who also support argument filterings. Here, they successively develop a SAT encoding [ZHM07], an encoding to Pseudo-Boolean constraints [ZM08], and an encoding to SAT modulo the theory of linear integer/rational/real arithmetic [ZHM09, Zan09]. Perhaps a bit surprisingly at first glance, the authors of [ZHM09, Zan09] report that also their search procedures based on encodings to SAT and to Pseudo-Boolean constraints, where generally exponential runtime in the worst case may occur with current solving back-ends, outperform implementations of the polynomial-time algorithm of [KV03] on benchmark sets arising in practice.

- In recent years, a lot of research has been done on term orders that are based on interpretations to *weakly* or *extended monotone algebras* [EWZ08], both on conceptual level and on the level of automation. A common theme here is to choose an algebra equipped with a well-founded order as well as a compatible quasi-order and to identify a class of (weakly or strongly) monotonic functions on the carrier that is closed under composition. By interpreting (term) function symbols as monotonic functions (on the algebra) and by extending this interpretation homomorphically to terms, the order on the algebra then induces a corresponding order on terms as well.

  For automation, one usually fixes the algebra and a certain shape of *parametric* functions [CMTU05] as interpretations for the symbols in advance. While in many cases the orders are not decidable even for concrete interpretations, one can nonetheless often identify decidable sufficient criteria. By using suitable constraint solving techniques for the resulting constraints on the parameters, one then finds concrete interpretations automatically from a—usually exponentially sized—set of functions.

  - Perhaps the most classic shape of such interpretations uses the algebra with carrier $\mathbb{N} = \{0, 1, 2, \ldots\}$, with the usual orders "$\geq$" and "$>$" on $\mathbb{N}$, and with "$+$" and "$*$" as operations used as components for the interpretations of function symbols. This gives rise to interpreting function symbols (and terms) as *polynomial functions* (or *polynomials*), and the correspondingly induced term orders are known as *polynomial orders* [Lan79, BL87].[5] In the paper [CMTU05], Contejean *et al.* propose a completely automatic approach to *search* for such orders. Using parametric interpretations and the absolute positiveness criterion [HJ98] to eliminate universal quantification, Contejean *et al.* encode the

[5] In this thesis, we often do not distinguish between (classes of) term interpretations and the corresponding (classes of) term orders if the difference is clear from the context or irrelevant.

search for polynomial orders fulfilling given term constraints to a quantifier-free conjunction of atomic constraints in non-linear integer arithmetic (NIA).

However, existence of a decision procedure for satisfiability of even this subset of SMT-NIA would be in contradiction to undecidability of Hilbert's $10^{th}$ problem [Mat70]. Thus, Contejean *et al.* [CMTU05] restrict the search space for the satisfiability check to finite integer intervals, and hence to a *finite domain constraint satisfaction problem*. Following [CMTU05], standard solving techniques for this general class of problems from *constraint logic programming (CLP)* [CD96, JL87] make use of an adapted Davis-Putnam procedure [DP60] for SAT solving. Inspired by techniques used in CLP for linear constraints, [CMTU05] provide a variant of this general approach to the case of non-linear constraints. So instead of a SAT encoding, [CMTU05] present a dedicated algorithm based on ideas that are also used in current SAT solvers.

Despite its sophistication, however, performance of this algorithm turns out to be lacking on many examples arising in termination proving practice. In [FGM+07, Fuh07], we present an alternative approach to solving the resulting NIA constraints based on *bit-blasting* of the unknowns and the corresponding arithmetic operations to a SAT instance. Using the same idea, we also provide a similar encoding to Pseudo-Boolean constraints; both encodings can be computed in polynomial time w.r.t. the size of the SMT-NIA problem [FGM+07]. Experimental results in [FGM+07, Fuh07] indicate that these encoding-based approaches outperform the algorithm of [CMTU05] by orders of magnitude when applied in the same setting for termination proving. A further advantage of the SAT-based approach in [FGM+07, Fuh07] is that an extension from conjunctions of NIA constraints to SMT-NIA is straightforward. In later work, Borralleras *et al.* [BLN+09, BLO+12] present a polynomial-size encoding of SMT-NIA to *SAT modulo linear integer arithmetic (SMT-LIA)* with empirical results comparable to those of [FGM+07, Fuh07].

By these performance improvements on the side of the *back-end*, also more complex encodings of constraint-based termination techniques to SMT-NIA have become feasible for practical problem instances, paving the ground for several further publications. Here, more involved settings for the underlying algebras are considered.

For instance, in [FGM+07] we provide an SMT-NIA encoding to search for polynomial orders where the interpretations may also contain *negative constants* [HM07]. Here, well-definedness of the interpretation to $\mathbb{N}$ is ensured by considering the maximum of 0 and the polynomial (possibly with a negative constant). The SMT-NIA encoding makes use of the convenient expressiveness of Boolean connectives provided by the SMT formalism.

Moreover, the paper [GTSS07] presents work on automatically proving *innermost* termination for term rewriting by *bounded increase*. Here, the underlying algebra uses $\mathbb{Z}$ as carrier, and also negative coefficients are allowed in the interpretations of (certain) function symbols. In this setting, termination is shown by proving boundedness of any assumed infinitely decreasing *dependency pair chain* [AG00] (corresponding to an infinite sequence of function calls in the TRS). Since for the standard order ">" on $\mathbb{Z}$, any infinite $>$-chain eventually exceeds any given fixed bound, an interpretation of terms to $\mathbb{Z}$ here leads to a valid termination argument, even though the order ">" is not well founded.

In [FGM+08] we show how to use "max" (and "min") together with "+" and "$*$" as constituent operations in interpretations used for showing (full) termination. Moreover, we also propose an SMT-NIA-based automation for interpretations with "max" in combination with negative coefficients [HM07], where we enhance the DP framework accordingly to allow for further improvements over [HM07].

Zankl and Middeldorp present *increasing interpretations* [ZM08, ZM09, Zan09], where polynomial orders can be used in the DP framework also if they lead to an *increase* in some dependency pairs, as long as over each *cycle* of the dependency graph ($\approx$ the control-flow graph) an overall decrease occurs. Also here, SMT-NIA is used as reduction target for automation.

In [SPG+09, FGP+11] we integrate *inductive theorem provers* into the DP framework for proving innermost termination. Here, automation also makes use of an underlying order on terms with certain monotonicity requirements, which can again be expressed, e.g., for polynomial orders by using an SMT-NIA encoding.

– Polynomial orders have also been extended to the non-negative rational and real numbers $\mathbb{Q}_{\geq 0}$ and $\mathbb{R}_{\geq 0}$, respectively. After initial work by Dershowitz [Der79], where the polynomial orders are restricted to simplification orders, Lucas [Luc05, Luc07] uses a minimum *distance* $\delta > 0$ for strict comparison (i.e., $x >_\delta y :\Leftrightarrow x - y \geq \delta$) to ensure well-foundedness of the strict order (cf. also [Ges90, Hof01]). For automation with satisfiability-based techniques, Lucas [Luc05] proposes a reduction to SMT-NIA by expressing parametric non-negative rational coefficients $c$ as $c = \frac{a}{b}$ (where $a \geq 0$ and $b > 0$) and by subsequently multiplying with the common denominator. Our experiments with an SMT-NIA-based implementation of this approach [FNO+08] (combined with bit-blasting) indicate that this reduction has beneficial results in practice.

– Generalizing linear polynomial orders, Endrullis, Hofbauer, Waldmann, and Zantema present *matrix orders* in [HW06, EWZ06, EWZ08]. Here, instead of

$\mathbb{N}$, now the set of *tuples* (or *vectors*) from $\mathbb{N}^d$ for some $d \geq 1$ is considered as carrier of the monotone algebra. As coefficients of these vectors, square matrices from $\mathbb{N}^{d\times d}$ are used, and addition and multiplication are the corresponding standard matrix operations.

For automation, [HW06, EWZ06, EWZ08] propose an approach similar to the one described in [CMTU05]. Again, parametric interpretations are used, which give rise to an SMT-NIA instance over the parameters. For solving such SMT-NIA problems, [HW06] propose a satisfiability-based approach. The authors remark that they put a bound on the search space to render the problem decidable, represent the integer variables by sequences of propositional variables in unary or binary notation, and use formulas to express relations over the variables. Further details on the encoding are not given in [HW06], however. In [EWZ06], the authors state that integer variables are encoded to a list of Boolean variables in binary representation. For intermediate results, likewise additional variables are introduced. The journal version [EWZ08] provides more detail on the SAT encodings for SMT-NIA, describing an encoding that essentially coincides with that of [FGM+07, Fuh07], but has been developed independently.

Analogously to polynomial orders, recently also matrix orders have been extended to carriers like $\mathbb{Q}^d_{\geq 0}$ and $\mathbb{R}^d_{\geq 0}$ [GHW07, ALN09, ZM10]. In the paper [Luc10], Lucas discusses the interplay between the required dimensions of matrix interpretations and the required search space for the matrix entries for successful termination proofs. Here both interpretations to $\mathbb{N}^d$ and $\mathbb{Q}^d_{\geq 0}$ are considered. In [ST10], Sternagel and Thiemann lift the setting of polynomial interpretations with negative constants [HM07] to matrix orders where $\mathbb{N}^d$ and also $\mathbb{Q}^d_{\geq 0}$ are considered as carriers.

For automation, [GHW07] proposes evolutionary algorithms. Later, Zankl and Middeldorp [ZM10] improve the encoding approach employed for polynomial orders in [FNO+08] for use with rational matrix orders by a heuristic that enforces *cancellation* in fractions. Moreover, they introduce extensions of the SAT encoding allowing to search also for *irrational* real numbers as matrix entries.

– As underlying monotone algebras for interpretation-based termination methods in string and term rewriting, recently also structures known as *exotic algebras* (cf., e.g., [Kro98]) have been introduced [Wal07, KW08, KW09]. These algebras are semi-rings where "max" or "min" are used as semi-ring addition operation and standard addition is used as semi-ring multiplication. Instead of, e.g., only $\mathbb{N}$ as carrier, also the neutral element of the semi-ring addition operation is included in the carrier ($-\infty$ for "max", $\infty$ for "min"). Semi-ring multiplication

and the comparison operations are adapted accordingly. To highlight the role of the operations, one often also writes "$\oplus$" instead of "max" or "min" and "$\otimes$" instead of "+" for a given semi-ring. The semi-ring $(\mathbb{N} \cup \{\infty\}, \min, +)$ is known as the *tropical semi-ring*, and the semi-ring $(\mathbb{N} \cup \{-\infty\}, \max, +)$ is known as the *arctic semi-ring*.[6] So here the building blocks of interpretations for function symbols are, besides numbers from $\mathbb{N}$, also the operations "+" and "min" (or "max") together with the respective neutral elements for the operations (not only 0, but also $\infty$ or $-\infty$.) As for standard (semi-)rings, also here matrix orders with vectors of tropical (or arctic, respectively) numbers are used as the underlying framework. In the case of arctic matrix orders, [KW08, KW09] also present a variant of orders with negative numbers from $\mathbb{Z}$ as matrix entries (similar to those described in [HM07, FGM+07]). In [Wal07], Waldmann also considers the algebra $(\mathbb{N} \cup \{-\infty, \infty\}, \min, \max)$, relating it to the automata-based *match-bounds* termination technique [GHW04].

For automation of the search for matrix orders over these algebras, Koprowski and Waldmann [Wal07, KW08, KW09] propose a parametric approach similar to that of [CMTU05, HW06, EWZ06, EWZ08], with suitable modifications for the properties of the algebra operations. To encode the resulting symbolic constraints with the operations "max" and "+" arising with arctic algebras, [KW08, KW09] suggest a representation of numbers from $\mathbb{N}$ in binary. Moreover, an extra Boolean variable is used to represent $-\infty$. The paper [Wal07] suggests an analogous encoding for symbolic constraints with "min" and "+". Moreover, for the algebra $(\mathbb{N} \cup \{-\infty, \infty\}, \min, \max)$, [Wal07] also proposes an encoding in unary notation, corresponding to an *order encoding* representation [CB94, TTKB09]. Later, [HKW10] suggests also unary SAT encodings for constraints with "min" and "+" that stem from tropical algebras. For representing the "+" operation, [HKW10] suggests the use of *sorting networks* (cf. [ES06]). As an alternative, [HKW10] also suggests mixed linear programming [JLN+10].

– In [ZW07], Zantema and Waldmann present *quasi-periodic interpretations* as a means of proving termination of string rewrite systems by quasi-periodic functions, i.e., functions $f : \mathbb{N} \to \mathbb{N}$ with $f(x + p) = f(x) + s * p$, where $p$ is called the *period* and $s$ the *slope* of the function.

  For automation, Zantema and Waldmann again use a parametric approach. Given fixed values for $p$, $s$ and the maximum numbers of bits for function values

[6] According to [Gat06], the tropical semi-ring is named as such because Imre Simon, who was among the first to investigate this particular structure, comes from the tropical country Brazil. Correspondingly, in [Goo98] the arctic semi-ring (using $\mathbb{R} \cup \{-\infty\}$ as carrier) is named as such because "max" is the "opposite" operation to "min" and likewise, arctic latitudes are located "opposite" to tropical latitudes on Earth.

and intermediate results respectively, they propose a SAT encoding based on binary representation of natural numbers.

In [KW09], Koprowski and Waldmann show that for string rewrite systems, arctic matrix orders subsume weakly monotonic quasi-periodic interpretations of slope 1. In [ZW07], Zantema and Waldmann report that in their implementation of quasi-periodic interpretations using slopes higher than 1 does not add much power on the TPDB when combined with other termination techniques.

– *Context-dependent interpretations* [Hof01] are a variant of polynomial interpretations that can be used to assess the *derivational* complexity of a TRS (i.e., the length of the longest derivation in a given TRS from an *arbitrary* term $t$ as a function of its size).[7] The paper [MS08] presents an NIA encoding for the subclasses of $\Delta$-*linear* and $\Delta$-*restricted interpretations* [MS08] via parametric interpretations. The parametric approach is similar to that of [CMTU05] and enables SMT-based search also for this class of interpretations.

- Koprowski and Middeldorp [KM07] provide a SAT encoding for the simultaneous search for a *predictive labeling* [HM06] over $\mathbb{N}$ in combination with a lexicographic path order on the labeled DP problem that allows to delete at least one dependency pair from the original DP problem. Here, encodings for LPO [CSL+06, ZHM07] (adapted for infinite sets of term constraints), for quasi-models based on matrix interpretations [EWZ06, EWZ08], and for finite branchingness of the labeled (infinite) TRS are combined.

  In [Ava10], Avanzini presents a SAT encoding for POP* [AM08] in combination with *semantic labeling* [Zan95] over *finite* carriers with the goal of proving upper bounds for the innermost runtime complexity of TRSs.

- Real-life programming languages usually come with pre-defined datatypes, e.g., the built-in integers. With the goal of non-termination preserving translations from programming languages to term rewriting in mind, we have extended classic term rewriting by infinitely many rules for predefined operations over the integers and the Booleans, calling the resulting formalism *integer term rewriting* [FGP+09]. Building on ideas from [GTSS07, FGM+08], we adapt the DP framework and also the SMT-NIA encodings for polynomial orders to the setting of the integers, where the usual well-founded term orders are not directly applicable. In the mean time, integer term rewriting has been used as a translation target language for Java Bytecode [OBEG10, BOEG10, BOG11], which renders these adaptions of SMT-based termination techniques for rewriting accessible also for termination analysis of imperative

[7] In this sense, complexity analysis is a refinement of termination analysis. One not only wishes to state *that* a given TRS is terminating, but also to give an asymptotic upper bound for *how quickly* it is terminating. Any upper bound on the derivational complexity of a TRS implies its termination.

languages.

At approximately the same time, Falke and Kapur [FK08, FK09, Fal09] introduce *$\mathcal{PA}$-based TRSs* (where $\mathcal{PA}$ denotes Presburger arithmetic, i.e., *linear integer arithmetic (LIA)*) as a back-end formalism for termination analysis of imperative programs. Here, rewrite rules may take SMT-LIA formulas as *constraints* that must be satisfied for a rule to be applicable. In [FK10] they extend their approach by allowing context-sensitive rewrite restrictions. In contrast to [FGP$^+$09], non-linear arithmetic operations such as multiplication of variables are not supported.

Falke and Kapur use SMT-LIA encodings to compute the dependency graph (called a *termination graph* in [FK09]; it is worth noting that [FK09] does not allow nested user-defined function symbols, which renders the dependency graph for a $\mathcal{PA}$-based TRS *decidable*, e.g., using an SMT-LIA solver) and to search for polynomial interpretations (also here, the restricted term structure renders SMT-LIA directly applicable) via SMT solving. This way, [FK09] achieves significantly better performance when compared to [FGP$^+$09] on similar TRSs with built-in integers. However, this performance comes at the expense of expressivity and power, since non-linear combinations of variables and user-defined data-structures in imperative input programs have to be linearized by a suitable abstraction to allow for an encoding of imperative programs (without side effects) to $\mathcal{PA}$-based TRSs. These points highlight that the approaches from [FGP$^+$09] and [FK09] are complementary.

In [FKS11a, FKS11b], Falke, Kapur, and Sinz adapt this approach to *`int`-based TRSs*, which are an extension of the $\mathcal{PA}$-based TRSs of [FK09] where also multiplication is allowed. This way, one can also represent division and modulo operations via slack variables. Due to these non-linear operations on integer variables, the dependency graph becomes undecidable for `int`-based TRSs, but one can still apply an SMT-LIA solver to get an approximation of the dependency graph. The SMT-LIA encoding from [FK09] for finding polynomial interpretations directly carries over to this setting. In [FKS11a, FKS11b] the authors successfully use `int`-based TRSs as the target of a non-termination-preserving translation from programs in the compiler intermediate language *LLVM-IR* [LA04]. LLVM-IR is a compiler intermediate representation used in the LLVM compiler framework. Since this framework has front-ends, e.g., for the languages C, Objective-C, C++, ..., one can thus use the approach of [FKS11a, FKS11b] to prove termination of programs written in these languages. As a restriction of this approach, however, user-defined data-structures in imperative input programs again must be eliminated by abstraction beforehand. Such fixed abstractions are applied in a preprocessing step, e.g., by the termination tool KITTeL, which implements the approach of [FKS11a, FKS11b].

- Also outside of term rewriting, constraint-based methods for the automated dis-

covery of termination arguments have been applied. In [PR04a], Podelski and Rybalchenko present a complete method to discover *linear ranking functions* for *linear arithmetic simple while* programs, i.e., imperative programs where all variables range over $\mathbb{Z}$ and where update statements are given by linear inequalities. Thus, also non-deterministic updates are possible. Similar to the imperative language considered in the later work [FK09], this formalism requires that non-linear arithmetic operations and user-defined data structures have been eliminated by suitable abstraction techniques before this technique is applied. However, in contrast to [FK09], Podelski and Rybalchenko require for analysis of imperative programs that moreover also all control structures like conditional statements or inner loops have been eliminated in a preprocessing step, e.g., by abstraction. For automation of the constraint-based method, [PR04a] proposes the use of *linear programming over rationals* [Hol95] (one can also use an SMT solver for *Linear Rational Arithmetic (LRA)*).

Also for direct termination analysis of *logic programming*, techniques based on encodings to NIA are in use nowadays. For instance, the termination analyzer Polytool [NDGS11], which is based on an adaption of polynomial interpretations from term rewriting to logic programming, automates the search for polynomial interpretations via a reduction to NIA (using our tool AProVE [GST06] as NIA solver [FGM+07]).

- The *size-change termination* principle [LJB01] is used in a variety of different settings for termination proving (cf. Section 6 for a more in-depth discussion). Here, one usually first abstracts a program to a representation via size-change graphs, which then in a second stage are analyzed to check whether this abstract program is terminating with respect to the size-change termination criterion. Thiemann and Giesl present a suitable adaption of this first stage to term rewriting in [TG05]. Classically, for automation of the second stage one applies a purely graph-based algorithm in order to construct a closure under composition. In [CLSS06], Codish *et al.* encode the check for size-change termination to BDDs instead, resulting in significant performance improvements in practice. Still, the question whether a given set of size-change graphs is size-change terminating is PSPACE-complete [LJB01], and the resulting BDDs may require exponential time and space [CLSS06]. Based on an equivalent characterization of size-change termination via a class of *ranking functions* published in [Lee09], in [BC08] Ben-Amram and Codish identify an *NP-complete fragment* of size-change termination which they name *Size-Change in NP (SCNP)*. They automate this new criterion via a suitable SAT encoding for the subclass of ranking functions corresponding to SCNP. In [CFGS10], we integrate SCNP into the DP framework as a class of *reduction pairs*, allowing to use SCNP as a component of the widely applied *reduction pair processor* [GTS05a, GTSF06]. For

automation, we extend the SAT encoding of [BC08] to *defer* the abstraction step of the first stage. Benefiting from *compositionality* of SAT encodings, we combine SAT encodings for term orders (giving rise to size-change graphs) and for SCNP (implying termination of these size-change graphs). This way, we encode the first and the second stage of size-change termination analysis into a single SAT problem.

*Monotonicity constraints* [CLS05] are an extension of size-change graphs for the setting of constraints over $\mathbb{Z}$, which (with its usual comparison ">") is not well founded (cf. also [GTSS07, FGP+09]), in contrast to the setting where size-change termination is usually applied. Also here, classically one applies a graph-based algorithm for automation, and the question whether a given set of monotonicity constraints is terminating is PSPACE-complete. In [CGB+11], we identify an NP-complete subclass of monotonicity constraints which we name *Monotonicity Constraints in NP (MCNP)*. For automation, we lift the SAT encoding from [BC08] to this setting, where additional difficulties arise because $\mathbb{Z}$ is not well founded.

In his overview paper [Cod08], Codish identifies a common aspect shared by most SAT encodings used in termination analysis: Finite domain integer variables are represented as binary numbers with explicit bit representation, and operations on these bit tuples are encoded via Boolean functions. This aspect occurs in many of the more recent encodings for path orders (cf. [CGST12]), for Knuth-Bendix orders (cf. [ZHM09]), and for interpretation-based orders (cf. e.g. [FGM+07, EWZ08, KW08, ZM10]). Here, finite-domain integers are used to represent either numbers as such or elements of a (finite) partial order like a precedence on function symbols for path orders or Knuth-Bendix orders.

It is worth mentioning that techniques like polynomial or matrix interpretations have applications beyond termination analysis. For instance, in the setting of complexity analysis, polynomial interpretations (with certain additional restrictions) can be used to deduce asymptotic upper bounds on the *runtime complexity* of innermost rewriting [HM08, NEG11]. Similarly, [MSW08] presents *triangular matrix interpretations*, where a direct termination proof with such an interpretation of dimension $d$ induces an asymptotic upper bound by a polynomial of degree $d$ on the *derivational complexity* of the analyzed TRS.

SAT encodings for orders on terms can also be used as a black box to improve techniques like *(Knuth-Bendix) completion* [KB70], where an equivalent convergent TRS is synthesized from a given set of term equations. This TRS can then be used to solve the word problem for this set of equations in an automated way. While the original procedure requires a given order on terms, in very recent work Klein and Hirokawa [KH11] propose an encoding to *MaxSAT* (i.e., to an *optimization problem*, where the number of oriented term equations is maximized). This way, now a whole class of orders such as RPOs,

KBOs, or polynomial orders can be used for a single run of the completion procedure, which leads to improved flexibility and applicability.

Finally, SAT and SMT encodings have also been used successfully in proofs of *non-termination*. In [Zan09, ZSHM10], Zankl, Sternagel, Hofbauer, and Middeldorp provide a SAT encoding for *looping non-termination* of *string rewrite systems (SRSs)*, a Turing-complete subclass of term rewrite systems where all function symbols have exactly one argument. The encoding is parameterized by the maximal length of the loops and by the maximal length of the strings (i.e., depth of the terms) to be considered.

In [BSOG12], Brockschmidt, Ströder, Otto, and Giesl describe an SMT-based approach to detect non-termination and occurrences of `NullPointerException`s in Java Bytecode programs. Here, the target theories are both linear and non-linear integer arithmetic.

## Contributions of this Thesis

The author of this thesis has made several important contributions to this *satisfiability-based* research effort in *termination analysis*, which have been published in the 8 conference and journal papers [FGM+07, FGM+08, FNO+08, FGP+09, SPG+09, CFGS10, CGB+11, FGP+11] and have ultimately led to the Chapters 3 – 6 of the present thesis. Moreover, in the papers [KST+11, FK11] co-authored by the author of the present thesis, we provide suitable translations to make these contributions accessible also for termination analysis of functional HOL programs used in the proof assistant Isabelle [NPW02] and for termination analysis of higher-order rewriting (cf., e.g., [KOR93]). In addition, we have performed a *method transfer* from termination analysis to the area of *synthesis of optimal Boolean circuits*. The corresponding results have been published in [FS10] and have given rise to Chapter 7.

The contributions of this thesis can be summarized as follows:

(i) The paper [HM07] presents an extension of polynomial interpretations as a reduction pair where negative constants are allowed. However, the paper addresses the question of automation only partially, and it remains unclear how to conduct an automated synthesis of suitable interpretations in general.

To remedy this issue, we present a novel SMT-NIA encoding of the search problem for polynomial interpretations with negative constants and prove its correctness. This way, for the first time a systematic search for such interpretations becomes possible. Moreover, we provide and prove a novel necessary condition for negative constants to be successfully applicable (in contrast to using the constant 0) using the approach of [HM07], which reduces the search space notably. We show that with these improvements, our automation for this class of interpretations is orders of magnitude faster than previous techniques.

(ii) Polynomial interpretations lack the ability to express that a value is the *maximum* of two other values. However, this form of abstraction can be helpful for termination proofs in certain cases.

We present an extension of polynomial interpretations by the max- and the min-operations. We show how to automate this extension via an encoding to SMT-NIA. Finally, we demonstrate the need for heuristics for such interpretations by showing that the worst-case behavior of our automation leads to an exponential blowup, and we present suitable heuristics to remedy this issue. This way, we obtain a technique that can be mechanized efficiently and at the same time improves the overall power of a modern termination tool.

(iii) Arctic matrix interpretations [KW08, KW09] are an interesting new technique for termination analysis which provides an alternative means for expressing the max-function. However, the SAT encoding proposed by [KW08, KW09] based on a binary representation of numbers is not beneficial for the performance of modern SAT solvers.

Therefore, we propose an alternative SAT encoding which is based on a *unary* encoding of arithmetic. Our experiments show that this leads to a notable improvement in performance. In fact, it is likely that AProVE found the highest number of termination proofs in the *SRS Standard* category of the Termination Competition 2009 only because of this improvement.

(iv) The *size-change termination* principle [LJB01] is a widely used means of proving termination. Here, classically one first applies some abstraction to the input program and only then is the check for termination via the size-change criterion performed. In this approach, it is not clear how to select a suitable abstraction in advance such that a proof of size-change termination of the abstracted program becomes possible later. Therefore, the state of the art here still is to use a generate-and-test approach for obtaining abstractions for a size-change proof.

We show how to overcome this limitation via the compositionality of SAT encodings. For automation, we combine a SAT encoding for an important fragment of size-change termination called *Size-Change termination in NP (SCNP)* [BC08] with given SAT encodings for term orders (which constitute the abstractions used for terms). This development is underpinned by our formalization of this termination criterion as a reduction pair, which allows for a smooth integration into the DP framework.

For the correctness proof of our contribution, we introduce the novel notion of *tuple-typed* dependency pair problems and show that in practically important cases it suffices to consider this class of problems. This way, intuition from the original

dependency pair approach [AG00] carries over to the more general dependency pair framework [GTS05a], which facilitates the integration of approaches to termination proving originating outside the area of term rewriting like SCNP. We conjecture that this contribution will also be used for integration of further termination techniques into the DP framework.

Moreover, we present a challenge example that can be solved due to this contribution. The example highlights that SCNP can add, e.g., max-comparison for dependency pairs also in case of base orders such as matrix orders that do not allow max-comparison on their own.

SAT encodings are not only beneficial in the area of termination analysis. In fact, the *methods* used for SAT encodings also carry over to quite distinct application domains (cf., e.g., the overview article [Mar08]). This way, expertise on SAT encodings developed in one domain of application transfers to other domains as well. This further indicates the versatility of SAT encodings as a technology.

In Chapter 7 of the present thesis, we substantiate this claim for the application domain of *circuit synthesis*. Concretely, our goal is to generate minimal circuits containing only XOR gates. Such circuits can be used to represent straight-line programs over the Galois field of two elements, GF(2). A prominent application of such circuits is the *S-box* that is used as a part of an AES [Fed01] implementation in cryptography. So far, only incomplete heuristics [BP10] were known to approach this problem.

This setting gives rise to the contribution of Chapter 7, which is distinct from the other contributions by its application area, yet surprisingly similar by the applied methods:

(v) We propose a novel SAT encoding for synthesizing linear straight-line programs over GF(2) from a specification, where we aim for optimality, i.e., for shortest such programs. This class of programs represents Boolean circuits which are composed of XOR gates. We show that we can both obtain small programs and that we can also prove their optimality within reasonable time. We substantiate this claim via an empirical case study where we optimize part of the S-box of the AES cipher. Here, we obtain both an optimal solution and also a proof of its optimality.

## Implementation and Empirical Evaluation

Not only have the above theoretical contributions been elaborated on paper, but they have been implemented as well. This implementation mostly took place in our automated termination analyzer AProVE [GST06] and—for Contribution (v)—partly in an external tool [FS10] building upon AProVE. In the course of the development of AProVE, the author of the present thesis has been one of the lead developers and contributed over 57 000 lines

of code to the sources of AProVE, which is implemented in Java. Additionally, the author has also conducted extensive experiments to evaluate this implementation empirically.

The practical contributions of this thesis can be summarized as follows:

(vi) We have implemented the encodings described in contributions (i) – (iii) in such a way that they can be plugged into different processors of the dependency pair framework [GTS05a].

We have conducted extensive empirical evaluations to assess the impact of these contributions. Here, our experiments show performance improvements by orders of magnitude over existing algorithms for Contribution (i). For Contributions (ii) and (iii) we report on notable improvements of power of our termination tool AProVE in practical settings.

(vii) For Contribution (iv), we provide the first implementation that allows for a systematic search of suitable abstractions for the *size-change termination* method for term rewriting. Prior to our contributions, the state of the art was to enumerate possible abstractions and then to check for size-change termination.

Using suitable SAT encodings, we obtain the first implementation of size-change termination that allows to search for the abstraction at the same time as for the size-change termination argument. In our empirical evaluation, this implementation shows significant improvements in power and speed over previous implementations of size-change termination analysis for term rewriting.

(viii) We have built a small stand-alone tool for synthesis of Boolean XOR circuits on top of the SAT framework of AProVE. Using this tool and an arbitrary SAT solver as a back-end, we can automatically obtain minimal circuits (or, equivalently, linear straight-line programs over GF(2)) for a given specification. This class of programs has applications, e.g., in cryptography, and using our tool, we have obtained a minimal implementation as well as an optimality proof for an important part of the widely used symmetric cipher AES.

The significance of Contributions (vi) and (vii) is also underlined by the fact that AProVE could maintain its highest score in the term rewriting category of the International Termination Competition over the years 2007 – 2011 and that AProVE has been scoring highest in the string rewriting category since 2009.

## The SMT Perspective

From the point of view of SMT solving, the contributions of this thesis are interesting on different levels.

- From a high-level perspective, Contributions (i) and (ii) have the particularly nice property that the encodings lead to constraints in the rather expressive SMT-NIA language. While nowadays SAT encodings indeed do constitute the state of the art for this class of constraints, with the ongoing advances in SMT solving, it is quite possible that in the future other means of constraint solving are developed which are more efficient for the SMT-NIA instances from our applications. These can then directly be used as drop-in replacements for the current SAT-based approaches to solving SMT-NIA instances.

  From an SMT perspective, Contributions (i) and (ii) are thus concerned with the *generation* of constraints in SMT-NIA.

- Contribution (iii) in contrast primarily advances the state of the art in SAT encodings for SAT modulo the theory of *linear arctic arithmetic*. Thus, this contribution is independent of the application area of termination analysis and can also be used as a back-end for constraints from linear arctic arithmetic arising from entirely different applications.

  From an SMT perspective, Contribution (iii) is thus primarily concerned with *solving* of SMT problems (for linear arctic arithmetic).

- Contribution (iv) is based on an underlying SAT encoding. Since any SMT solver also includes a SAT solver, this contribution can also directly be lifted to underlying SMT encodings for the base orders instead of SAT encodings.

  From an SMT perspective, Contribution (iv) is thus concerned with building on a given (SAT or SMT) encoding to *generate* a more complex problem instance.

- Contribution (v) is presented by means of a SAT encoding. Here, we give the building blocks of this encoding on a sufficiently high level to allow for an easy implementation of the encoding also in a richer language. For instance, in [Ban10], Mutsunori Banbara reports on his implementation of the encoding in the input language of the *Constraint Satisfaction Problem (CSP)* solver Sugar [TTKB09] based on our problem description in [FS10].

  Thus, from an SMT perspective, Contribution (v) again is about *generating* SAT instances, which can also be implemented in richer SMT languages. In contrast to Contributions (i) – (iv), here not only a satisfiability proof is helpful, but also an unsatisfiability proof is needed if one wants to show optimality.

## Hitherto Unpublished Contributions

The author has already published preliminary versions of important parts of this thesis in 14 joint articles in international journals and peer-reviewed conference proceed-

ings [SES+12, FK11, KST+11, FGP+11, CGB+11, CFFS11, CFGS10, FS10, SPG+09, FGP+09, AEF+08, FNO+08, FGM+08, FGM+07].

Nevertheless, this thesis contains a large number of significant contributions that have not been published so far:

- Contribution (i) extends the conference publication [FGM+07] by the correctness proof for the presented SMT-NIA encoding. Moreover, we provide a novel necessary syntactic criterion for negative constants for the presented class of polynomial orders with negative constants, along with the correctness proof and an empirical evaluation.

- Contribution (ii) extends the conference publication [FGM+08] by means of several successful heuristics for the shape of the used max-interpretations. The need for such heuristics is pointed out both by empirical observations in practice as well as by showing that the presented technique may lead to an exponential blowup in size if no suitable heuristics are employed. This latter example also constitutes a novel contribution. Additionally, we provide novel transformation rules which allow to avoid the exponential blowup in certain cases. Moreover, we assess the applicability of Contribution (ii) for the setting of Contribution (i) in practice.

- Contribution (iii) is novel and unpublished.

- Contribution (iv) extends the conference paper [CFGS10] by the correctness proof for the presented technique. To facilitate this proof, we propose the novel notion of *tuple-typed* dependency pair problems as well as criteria for switching between the classic untyped and the tuple-typed setting. This new contribution allows for smoother adaptions of termination techniques that originate outside of term rewriting. Also the example TRS which demonstrates the advantages of SCNP has not been present in the conference paper.

- Contribution (v) extends the conference paper [FS10] by a new optimality proof for our AES case study, which has not been formally published.

## Thesis Structure

This thesis is organized as follows. In Chapter 2, we set up the scenario for most of the contributions of the thesis. Here, we recapitulate the basics of term rewriting, the dependency pair framework, and weakly monotone algebras for termination proving as well as the parametric approach to automatically search for suitable such algebras. In Chapter 3 we present Contribution (i) of this thesis, i.e., the novel SMT-based automation of polynomials with a negative constant. Contribution (ii) about extending polynomial interpretations by the maximum function is given in Chapter 4. We provide Contribution

(iii) on an improved SAT encoding for the constraints arising in termination analysis with arctic matrix interpretations in Chapter 5. Chapter 6 then contains Contribution (iv), i.e., the first automation of size-change termination in NP for term rewriting allowing to search for the underlying term abstraction and for the size-change termination argument via a single SAT instance. Contribution (v), which lifts the parametric approach followed throughout this thesis to the area of synthesis of optimal Boolean XOR circuits, is presented in Chapter 7. Finally, in Chapter 8 we conclude this thesis with an outlook on possible next steps in the endeavors of automated termination analysis and, more generally, improving the methodology of problem solving via SAT and SMT encodings.

# 2 Preliminaries

In this chapter we set up the scenario in which the first chapters of this thesis takes place. In Section 2.1 we recapitulate the basic definitions and theorems of (first-order) term rewriting (cf. [BN98]). Section 2.2 deals with dependency pairs [AG00] and the dependency pair framework (following [GTSF06]). In Section 2.3 we discuss weakly monotone algebras [EWZ08] and, specifically, polynomial interpretations [Lan79] to $\mathbb{N}$ and their automation [CMTU05, FGM+07]. Throughout this thesis, we define the set of natural numbers to be $\mathbb{N} := \{0, 1, 2, \ldots\}$.

Note that our main goal in the first chapters of this thesis is to make the dependency pair framework more powerful in practice. On the one hand, we aim for improving automation of existing techniques such that we need less resources to drive the automated termination proof effort and thus effectively increase the number of successful termination proofs within a given time limit. On the other hand, we define extensions of the dependency pair framework in the form of new reduction pairs and processors. This way, we increase termination proving power also from a theoretical point of view.

## 2.1 Term Rewriting

In this section we briefly recapitulate the basics and underlying concepts of term rewriting and fix the corresponding notations. For more details, we refer to the textbook [BN98], which also forms the basis for the presentation in this section.

A *signature* $\mathcal{F}$ is a non-empty finite set of *function symbols*. Each function symbol $f$ has an associated *arity* $ar(f) = n \in \mathbb{N}$. Function symbols with arity 0 are called *constant symbols* or *constants*. For a signature $\mathcal{F}$ and a (finite or infinite) set of *variables* $\mathcal{V}$ with $\mathcal{F} \cap \mathcal{V} = \varnothing$, the set of *terms* $\mathcal{T}(\mathcal{F}, \mathcal{V})$ (over $\mathcal{V}$) is inductively defined via (1) $\mathcal{V} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$ and (2) $f \in \mathcal{F}, ar(f) = n, t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \Rightarrow f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. For constants $c$, we write $c$ instead of $c()$ to denote the term that only consists of the symbol $c$. We write $\mathcal{T}(\mathcal{F})$ to abbreviate $\mathcal{T}(\mathcal{F}, \varnothing)$, i.e., the set of all terms without variables or *ground terms*. For a term $t$, $\mathcal{V}ar(t)$ denotes the set of all variables occurring in $t$, and for $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$, we define $\mathcal{V}ar(T) = \bigcup_{t \in T} \mathcal{V}ar(t)$. Likewise, $\mathcal{F}(t)$ denotes the set of all function symbols occurring in $t$, and for $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$, we define $\mathcal{F}(T) = \bigcup_{t \in T} \mathcal{F}(t)$.

$\mathcal{P}os(t)$ denotes the set of all *positions of* $t$ for a term $t$ where $\mathcal{P}os(t)$ is the smallest subset of $\mathbb{N}^*$ with (1) $\varepsilon \in \mathcal{P}os(t)$ and (2) $t = f(t_1, \ldots, t_n), 1 \leq i \leq n, \pi \in \mathcal{P}os(t_i) \Rightarrow i\pi \in \mathcal{P}os(t)$.

The *size* of a term $t$ is defined as $|t| = |\mathcal{P}os(t)|$. For $\pi \in \mathcal{P}os(t)$, we write $t|_\pi$ to denote the *subterm* of $t$ at position $\pi$ defined inductively via (1) $t|_\varepsilon = t$ and (2) $f(t_1, \ldots, t_n)|_{i\pi} = t_i|_\pi$. To denote that $s$ is a subterm of $t$ at some position $\pi$, we also write $t \trianglerighteq s$. We call $s$ a *strict subterm* of $t$ if $s$ is a subterm of $t$ and $s \neq t$ and write $s \triangleright t$. If $t$ and $s$ are terms and if $\pi \in \mathcal{P}os(t)$, we write $t[s]_\pi$ to denote the term resulting from $t$ if $t|_\pi$ is replaced by $s$, i.e., (1) $t[s]_\varepsilon = s$ and (2) $f(t_1, \ldots, t_n)[s]_{i\pi} = f(t_1, \ldots, t_i[s]_\pi, \ldots, t_n)$. The function $root : (\mathcal{T}(\mathcal{F}, \mathcal{V}) \setminus \mathcal{V}) \to \mathcal{F}$ is used to map a non-variable term $t = f(t_1, \ldots, t_n)$ to its *root symbol* $f$ at the *root position* $\varepsilon$.

A *substitution* is a mapping $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$. $DOM(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ denotes the *domain* of $\sigma$.[8] We often write $\{x/\sigma(x) \mid x \in DOM(\sigma)\}$ to represent the substitution $\sigma$ if $DOM(\sigma)$ is finite. The notion of a substitution $\sigma$ extends homomorphically to a mapping $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$, i.e., $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$. We also write $t\sigma$ for $\sigma(t)$. We call a relation $\sim$ *stable* if $s \sim t$ implies $s\sigma \sim t\sigma$ for all substitutions $\sigma$ and terms $s, t$. A relation $\sim$ is *monotonic* if $s \sim t$ implies $f(u_1, \ldots, u_{i-1}, s, u_{i+1}, \ldots, u_n) \sim f(u_1, \ldots, u_{i-1}, t, u_{i+1}, \ldots, u_n)$ for all $n \in \mathbb{N}, f \in \mathcal{F}$ with $ar(f) = n, 1 \leq i \leq n$, and $u_1, \ldots, u_{i-1}, s, t, u_{i+1}, \ldots, u_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

A *(rewrite) rule* (over a signature $\mathcal{F}$ and a set of variables $\mathcal{V}$) is a pair $(\ell, r)$ of two terms $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ (usually written $\ell \to r$) where (1) $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)$ and (2) $\ell \notin \mathcal{V}$. For a rule $\ell \to r$, we call $\ell$ the *left-hand side (lhs)* and $r$ the *right-hand side (rhs)*. A *term rewrite system (TRS)* (over $\mathcal{F}$ and $\mathcal{V}$) is a set of rewrite rules. We only consider finite TRSs.

The *(term) rewrite relation* $\to_\mathcal{R} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V})$ for a TRS $\mathcal{R}$ is defined such that $s \to_\mathcal{R} t$ iff $s = s[\ell\sigma]_\pi$ and $t = s[r\sigma]_\pi$ for a position $\pi \in \mathcal{P}os(s)$, a rule $\ell \to r \in \mathcal{R}$, and a substitution $\sigma$. We call $s \to_\mathcal{R} t$ a *(term) rewrite step*, and $s$ is *reduced* at position $\pi$. A term $s$ is called an ($\mathcal{R}$-)redex (reducible expression) iff there exist a rule $\ell \to r \in \mathcal{R}$ and a substitution $\sigma$ such that $s = \ell\sigma$ (i.e., $s$ can be rewritten by $\mathcal{R}$ at the root position). A (finite or infinite) sequence of rewrite steps $t_1 \to_\mathcal{R} t_2 \to_\mathcal{R} t_3 \to_\mathcal{R} \ldots$ is also called a *rewrite sequence*. A term $s$ is in *normal form* w.r.t. a TRS $\mathcal{R}$ iff there is no $t$ with $s \to_\mathcal{R} t$.

A TRS $\mathcal{R}$ is *terminating* iff $\to_\mathcal{R}$ is well founded, i.e., there is no infinite rewrite sequence $t_0 \to_\mathcal{R} t_1 \to_\mathcal{R} t_2 \to_\mathcal{R} \ldots$.

**Example 2.1** (Bits, version 1)**.** Consider the following TRS over the signature $\mathcal{F} = \{0, \mathsf{s}, \mathsf{half}, \mathsf{bits}\}$. Here $\mathsf{s}$ represents the successor function, $\mathsf{half}(x)$ computes $\lfloor \frac{x}{2} \rfloor$, and $\mathsf{bits}(x)$

[8] Note that we also allow infinite domains for substitutions.

is the number of bits needed to represent all numbers up to $x$.

$$\mathsf{half}(\mathsf{0}) \to \mathsf{0} \tag{2.1}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{0})) \to \mathsf{0} \tag{2.2}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{half}(x)) \tag{2.3}$$
$$\mathsf{bits}(\mathsf{0}) \to \mathsf{0} \tag{2.4}$$
$$\mathsf{bits}(\mathsf{s}(\mathsf{0})) \to \mathsf{s}(\mathsf{0}) \tag{2.5}$$
$$\mathsf{bits}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{bits}(\mathsf{s}(\mathsf{half}(x)))) \tag{2.6}$$

For the term $t = \mathsf{bits}(\mathsf{s}(\mathsf{s}(\mathsf{half}(\mathsf{0}))))$ we have the following rewrite sequence (where we underline the subterm $\ell\sigma$ at which the reduction takes place):

$$\mathsf{bits}(\mathsf{s}(\mathsf{s}(\underline{\mathsf{half}(\mathsf{0})}))) \to_{\mathcal{R}} \underline{\mathsf{bits}(\mathsf{s}(\mathsf{s}(\mathsf{0})))} \to_{\mathcal{R}} \mathsf{s}(\mathsf{bits}(\mathsf{s}(\underline{\mathsf{half}(\mathsf{0})}))) \to_{\mathcal{R}} \mathsf{s}(\underline{\mathsf{bits}(\mathsf{s}(\mathsf{0}))}) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(\mathsf{0}))$$

There is also the following alternative rewrite sequence starting from $t$:

$$\underline{\mathsf{bits}(\mathsf{s}(\mathsf{s}(\mathsf{half}(\mathsf{0}))))} \to_{\mathcal{R}} \mathsf{s}(\mathsf{bits}(\mathsf{s}(\mathsf{half}(\underline{\mathsf{half}(\mathsf{0})})))) \to_{\mathcal{R}} \mathsf{s}(\mathsf{bits}(\mathsf{s}(\underline{\mathsf{half}(\mathsf{0})}))) \to_{\mathcal{R}} \mathsf{s}(\underline{\mathsf{bits}(\mathsf{s}(\mathsf{0}))}) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(\mathsf{0}))$$

## 2.2 The Dependency Pair Framework

In this section, we describe the main components of the *dependency pair framework (DP framework)*. This framework has become the *de facto* standard setting for automated termination proofs for term rewrite systems, and we have embedded also the contributions of this thesis for term rewriting into this framework. For more details on the DP framework cf., e.g., [GTSF06, Thi07].

Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined symbols* $\mathcal{D}_\mathcal{R} \subseteq \mathcal{F}$ are the set of root symbols of the left-hand sides of the rules of $\mathcal{R}$. We often write $\mathcal{D}$ instead of $\mathcal{D}_\mathcal{R}$ if $\mathcal{R}$ is clear from the context (or irrelevant). For each defined symbol $f \in \mathcal{D}$, we introduce a corresponding fresh *tuple symbol* $f^\sharp$ with the same arity as $f$. For the sake of readability, in examples we often write $F$ instead of $f^\sharp$. If $t = g(t_1, \ldots, t_n)$ with $g \in \mathcal{D}$, $t^\sharp$ denotes the term $g^\sharp(t_1, \ldots, t_n)$.

Now we can define the notion of *dependency pairs*.

**Definition 2.2** (Dependency Pair [AG00])**.** *The set of* dependency pairs (DPs) *for a TRS* $\mathcal{R}$ *is* $\mathcal{DP}(\mathcal{R}) = \{\ell^\sharp \to t^\sharp \mid \ell \to r \in \mathcal{R}, r \trianglerighteq t, root(t) \in \mathcal{D}_\mathcal{R}\}$.

Intuitively, a dependency pair corresponds to a function call that occurs because of a rewrite step.

**Example 2.3** (Dependency Pairs for Example 2.1)**.** The dependency pairs $\mathcal{DP}(\mathcal{R})$ of the TRS $\mathcal{R}$ from Example 2.1 are:

$$\mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{HALF}(x) \tag{2.7}$$

$$\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{HALF}(x) \tag{2.8}$$

$$\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{BITS}(\mathsf{s}(\mathsf{half}(x))) \tag{2.9}$$

We can use dependency pairs for proving termination of the rewrite relation $\to_{\mathcal{R}}$. For this, we need the concept of *chains*. Intuitively, a chain of dependency pairs corresponds to a sequence of function calls that occur during the reduction of a term. Here, we assume different occurrences of dependency pairs to be variable disjoint.

**Definition 2.4** ($(\mathcal{P},\mathcal{R})$-Chain [GTSF06])**.** *Let $\mathcal{P}$ and $\mathcal{R}$ be TRSs. A (finite or infinite) sequence of pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ from $\mathcal{P}$ is a* $(\mathcal{P},\mathcal{R})$-chain *(or just* chain*) iff there exists a substitution $\sigma$ such that $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ holds for every two consecutive pairs $s_i \to t_i$ and $s_{i+1} \to t_{i+1}$. The $(\mathcal{P},\mathcal{R})$-chain is* minimal *if there exists no $t_i\sigma$ with an infinite reduction $t_i\sigma \to_{\mathcal{R}} u_1 \to_{\mathcal{R}} u_2 \to_{\mathcal{R}} \ldots$ for some $u_1, u_2, \ldots$*

In other words, a chain is minimal if all arguments of the right-hand sides of the pairs are terminating with respect to $\to_{\mathcal{R}}$. This corresponds to function calls where all arguments of the call themselves are terminating. Hence, one must evaluate the function call at the root of the term itself in order to obtain an infinite evaluation sequence.

**Theorem 2.5** (Termination Criterion with $(\mathcal{P},\mathcal{R})$-Chains [AG00, GTS05a])**.** *A TRS $\mathcal{R}$ is terminating iff no infinite minimal $(DP(\mathcal{R}),\mathcal{R})$-chain exists.*

Absence of infinite minimal chains for $(DP(\mathcal{R}),\mathcal{R})$ thus implies that there cannot be infinitely many function calls during any reduction of an arbitrary term $t$. Of course, this implies that absence of infinite minimal $(DP(\mathcal{R}),\mathcal{R})$-chains is undecidable.

To prove automatically that there is no infinite minimal chain, we consider the notions of *DP problems* and of *DP processors* to transform such DP problems. Together, these definitions result in the *DP framework*. This way, we can perform modularized and incremental termination proofs since we can transform and simplify intermediate subproblems independently from each other via different techniques.

**Definition 2.6** (DP Problem [GTS05a, GTSF06, Thi07])**.** *A* dependency pair problem (DP problem) *is a pair $(\mathcal{P},\mathcal{R})$ of two TRSs $\mathcal{P}$ and $\mathcal{R}$.*[9] *By extension of the DP approach*

[9] For the sake of readability, we do not use a set $\mathcal{Q}$ of terms for fine-grained control over the *rewrite strategy* of the rewrite relation [GTS05a, Thi07]. Neither do we introduce a flag which indicates if we consider full or innermost termination (as in [GTSF06]). In this thesis, we do not impose any strategy restriction on the rewrite relation. Note that a proof of (full) termination also implies termination w.r.t. arbitrary rewrite strategies.

Moreover, we do not introduce a flag which states whether we regard *arbitrary* chains or only *minimal* chains (as in [GTS05a, Thi07]). To present the contributions of this thesis, it suffices to regard minimal chains.

*[AG00], we call a rule from $\mathcal{P}$ also a* dependency pair*.*

*We call a DP problem $(\mathcal{P}, \mathcal{R})$* finite *if no infinite minimal $(\mathcal{P}, \mathcal{R})$-chain exists.*

The initial DP problem for termination analysis of a TRS $\mathcal{R}$ has the form $(\mathcal{DP}(\mathcal{R}), \mathcal{R})$. Using Theorem 2.5 and Definition 2.6, we can now restate the termination criterion:

**Corollary 2.7** (Termination Criterion for DP Problems [GTS05a, GTSF06, Thi07])**.** *A term rewrite system $\mathcal{R}$ is terminating iff the DP problem $(\mathcal{DP}(\mathcal{R}), \mathcal{R})$ is finite.*

With the goal of showing termination of the original term rewrite system, we transform DP problems via termination proving techniques in the form of *dependency pair processors (DP processors).*

**Definition 2.8** (DP Processor [GTS05a, GTSF06, Thi07])**.** *A* dependency pair processor (DP processor) *is a function Proc that takes a DP problem as input and returns a (possibly empty) set of DP problems as output. A DP processor Proc is* sound *if for all DP problems $d$, $d$ is finite whenever $Proc(d)$ is a set of finite DP problems.*

Soundness of *Proc* is required to use *Proc* as part of a proof of termination. Corollary 2.9, which is a consequence of Corollary 2.7 and Definition 2.8, then yields a framework for termination analysis where several different DP processors can be combined. We start with the initial DP problem $(DP(\mathcal{R}), \mathcal{R})$ and transform the resulting DP problems repeatedly by sound DP processors. If the final processors return empty sets of DP problems, termination is proved.

**Corollary 2.9** (Dependency Pair Framework [GTS05a, GTSF06, Thi07])**.** *Let $\mathcal{R}$ be a TRS. We construct a tree whose nodes are labeled with DP problems or "yes" and whose root is labeled with $(DP(\mathcal{R}), \mathcal{R})$. For every inner node labeled with $d$, there is a sound DP processor Proc that satisfies one of the following conditions:*

- *$Proc(d) = \varnothing$ and the node has just one child, labeled with "yes"*
- *$Proc(d) \neq \varnothing$, and the children of the node are labeled with the DP problems in $Proc(d)$*

*If all leaves of the tree are labeled with "yes", then $\mathcal{R}$ is terminating.*[10]

Well-founded orders on terms are among the most classic techniques for showing termination. In the DP framework, these are used most prominently in the form of *reduction pairs.*

[10] Analogously, [GTS05a, GTSF06, Thi07] define a more extended version of the DP framework which can also be used for proofs of non-termination. There, processors can also return "no" instead of a set of DP problems. This allows to conclude non-termination of the original input TRS, provided that all processors applied on the path to the root of the tree are *complete* (a dual property to *sound*). Since the present thesis is not about non-termination, we omit this aspect of the definition of DP processors and the DP framework.

**Definition 2.10** (Reduction Pair)**.** *Let $\succsim$ be a stable and monotonic quasi-ordering, let $\succ$ be a (not necessarily monotonic) stable and well-founded ordering. Moreover, let $\succsim \circ \succ \subseteq \succ$ and $\succ \circ \succsim \subseteq \succ$ (i.e., $\succ$ is* compatible *with $\succsim$). Here, $\circ$ denotes the* composition *of two relations, i.e., $\succ_1 \circ \succ_2 = \{(a, c) \mid \exists b : a \succ_1 b \wedge b \succ_2 c\}$. Then we call $(\succsim, \succ)$ a* (weakly monotonic) reduction pair*. If moreover $\succ$ is monotonic as well, we call $(\succsim, \succ)$ a* strongly monotonic reduction pair*. Given a reduction pair $(\succsim, \succ)$ and a rule $s \to t$, we say that the reduction pair* orients *the rule* strictly *if $s \succ t$ holds, and we call the rule* strictly decreasing*. Moreover, we say that the reduction pair* orients *the rule* weakly *if $s \succsim t$ holds, and we call the rule* weakly decreasing*.*

For the application of reduction pairs to a DP problem in the DP framework, in many cases it suffices not to deal with *all* rules of $\mathcal{R}$ in $(\mathcal{P}, \mathcal{R})$. Instead, it suffices to orient only the *usable rules*. These include all rules that can be used to reduce the terms in right-hand sides of $\mathcal{P}$ when their variables are instantiated with normal forms. We use the following definition of usable rules:

**Definition 2.11** (Usable Rules [AG00, GTS05b])**.** *Let $\mathcal{R}$ be a TRS. For any function symbol $f$, let $Rls(f) = \{\ell \to r \in \mathcal{R} \mid root(\ell) = f\}$. For any term $t$, the* usable rules $\mathcal{U}_\mathcal{R}(t)$ *are the smallest set such that*

- $\mathcal{U}_\mathcal{R}(x) = \varnothing$ *for every variable $x$ and*
- $\mathcal{U}_\mathcal{R}(f(t_1, \ldots, t_n)) = Rls(f) \cup \bigcup_{\ell \to r \in Rls(f)} \mathcal{U}_\mathcal{R}(r) \cup \bigcup_{1 \le i \le n} \mathcal{U}_\mathcal{R}(t_i)$

*For a TRS $\mathcal{P}$, its* usable rules *are $\mathcal{U}_\mathcal{R}(\mathcal{P}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}_\mathcal{R}(t)$.*

**Example 2.12** (Initial DP Problem and Usable Rules for Example 2.1)**.** Consider again the TRS $\mathcal{R}$ from Example 2.1. The initial DP problem for termination analysis of $\mathcal{R}$ is $(\mathcal{DP}(\mathcal{R}), \mathcal{R})$ (cf. Example 2.3). The usable rules for the set of dependency pairs $\mathcal{DP}(\mathcal{R})$ are the $\mathsf{half}$-rules, i.e., $\mathcal{U}_\mathcal{R}(\ \mathcal{DP}(\mathcal{R})\ ) = \{(2.4), (2.5), (2.6)\}$.

An additional requirement on the quasi-order $\succsim$ of the reduction pair is needed to make the restriction to the usable rules in the reduction pair processor sound. One has to demand that $\succsim$ is $\mathcal{C}_\varepsilon$*-compatible*, i.e., that $\mathsf{c}(x, y) \succsim x$ and $\mathsf{c}(x, y) \succsim y$ hold for a fresh function symbol $\mathsf{c}$ [GTSF06, HM07].[11]

Using reduction pairs, we can now formulate a corresponding DP processor which allows to delete dependency pairs from a DP problem:

**Theorem 2.13** (Reduction Pair Processor [GTSF06])**.** *Let $(\succsim, \succ)$ be a reduction pair. Then the following DP processor Proc, called the* reduction pair processor*, is sound.*[12]

[11] This is not a strong restriction in practice since most commonly used quasi-orders in reduction pairs are $\mathcal{C}_\varepsilon$-compatible. An exception are equivalence relations.

[12] A more sophisticated version of the reduction pair processor also makes use of usable rules w.r.t. the (implicit) argument filtering of the reduction pair. To present the contributions of this thesis, this simpler version of the reduction pair processor is sufficient.

*Here,* $Proc(\ (\mathcal{P},\mathcal{R})\ ) =$

- $\{(\mathcal{P} \setminus \succ, \mathcal{R})\}$, *if* $\mathcal{P} \subseteq \succ \cup \succsim$ *and one of the following holds:*
  - $\mathcal{U}_\mathcal{R} \subseteq \succsim$ *and* $\succsim$ *is* $\mathcal{C}_\varepsilon$*-compatible*
  - $\mathcal{R} \subseteq \succsim$
- $\{(\mathcal{P},\mathcal{R})\}$, *otherwise*

**Example 2.14** (Reduction Pair Processor Constraints for Example 2.1)**.** Consider again the TRS $\mathcal{R}$ from Example 2.1, its initial DP problem $(\mathcal{DP}(\mathcal{R}),\mathcal{R})$, and the usable rules $\mathcal{U}_\mathcal{R}(\,\mathcal{DP}(\mathcal{R})\,)$ (cf. Example 2.12). We can use the reduction pair processor to delete all dependency pairs from $\mathcal{DP}(\mathcal{R})$ in the DP problem $(\mathcal{DP}(\mathcal{R}),\mathcal{R})$ if we can find a $\mathcal{C}_\varepsilon$-compatible reduction pair $(\succsim,\succ)$ such that the following term constraints hold:

$$\mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{HALF}(x) \tag{2.10}$$
$$\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{HALF}(x) \tag{2.11}$$
$$\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{BITS}(\mathsf{s}(\mathsf{half}(x))) \tag{2.12}$$
$$\mathsf{half}(\mathsf{0}) \succsim \mathsf{0} \tag{2.13}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{0})) \succsim \mathsf{0} \tag{2.14}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succsim \mathsf{s}(\mathsf{half}(x)) \tag{2.15}$$

Of course, it is worth mentioning that the literature contains many more DP processors. Another widely applied DP processor is the *dependency graph processor*. It is based on the notion of the *dependency graph* [AG00] of a DP problem. In a dependency graph for a DP problem $(\mathcal{P},\mathcal{R})$, the nodes are DPs from $\mathcal{P}$, and there is an edge between two DPs $s \to t$ and $u \to v$ iff a $(\mathcal{P},\mathcal{R})$-chain $s \to t, u \to v$ exists. Since this is undecidable, one works on overestimations of the dependency graph. The dependency graph processor [AG00, GTS05a] then deletes DPs that are not on a cycle in the dependency graph and decomposes the dependency graph into its strongly connected components with the sets of nodes $\mathcal{P}_1,\ldots,\mathcal{P}_n$. As output, one then obtains the set of DP problems $\{(\mathcal{P}_1,\mathcal{R}),\ldots,(\mathcal{P}_n,\mathcal{R})\}$. This way, one can conduct termination proofs in a modularized way (i.e., termination proofs for functions that are not mutually recursive can be performed separately).

Moreover, there are processors using *DP transformations* like narrowing, rewriting, or instantiation [GTSF06], which essentially perform partial and symbolic evaluation on the DPs to simplify DP problems. Techniques like the $\mathcal{A}$*-transformation* [GTS05b, Thi07] and *uncurrying* [HMZ08, ST11a] allow to convert a DP problem in higher-order curried notation into uncurried representation. One can apply strongly monotonic reduction pairs to delete not only DPs, but also rules from $\mathcal{R}$ using the *rule removal processor* [GTS05a]. Furthermore, the *size-change termination* [LJB01] criterion has been adapted to the DP

framework by [TG05, CFGS10]. Several variants of *semantic labeling* [Zan95] are available as DP processors (cf., e.g., [Thi07, SM08, ST11b]). This list is by far not exhaustive; see, e.g., [Thi07] for further DP processors.

## 2.3 Weakly Monotone Algebras and Polynomial Interpretations

Many of the orders considered in this thesis stem from interpretations of function symbols and terms to a *weakly monotone algebra*, like *polynomial interpretations with a negative constant* [HM07] in Chapter 3, the *max-polynomial interpretations* of Chapter 4, and *arctic interpretations* [KW08] in Chapter 5. We also use standard *polynomial interpretations* over $\mathbb{N}$ [Lan79] in examples throughout this thesis. To set up the symbolic constraints for the automatic search for polynomial interpretations, we follow the parametric approach of [CMTU05].

**Definition 2.15** (Weakly Monotone Algebra [EWZ08])**.** *An $\mathcal{F}$-algebra $\mathcal{A} = (A, [\cdot]_\mathcal{A})$ consists of a non-empty carrier set $A$ and, for every $f \in \mathcal{F}$ with $ar(f) = n$, a function $[f]_\mathcal{A} : A^n \to A$. We call $[f]_\mathcal{A}$ ($= [f]_\mathcal{A}(x_1, \ldots, x_n)$) an* interpretation *of $f$.*

*Interpretations extend homomorphically from function symbols to terms, i.e., $[x]_\mathcal{A} = x$ for all $x \in \mathcal{V}$ and $[f(t_1, \ldots, t_n)]_\mathcal{A} = [f]_\mathcal{A}([t_1]_\mathcal{A}, \ldots, [t_n]_\mathcal{A})$ for all function symbols $f$ of arity $n$ and all terms $t_1, \ldots, t_n$. We often write $[f]$ for $[f]_\mathcal{A}$ and $[t]$ for $[t]_\mathcal{A}$ if $\mathcal{A}$ is clear from the context or irrelevant.*

*A function $[f] : A^n \to A$ is* monotonic *with respect to a relation $\sim \subseteq A \times A$ if for all $1 \leq i \leq n$ and all $a_1, \ldots, a_{i-1}, b, c, a_{i+1}, \ldots, a_n \in A$ where $b \sim c$, we also have:*

$$[f](a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n) \sim [f](a_1, \ldots, a_{i-1}, c, a_{i+1}, \ldots, a_n)$$

*We then call $(\mathcal{A}, \geq, >)$ a* weakly monotone algebra *iff*

- *For all $f \in \mathcal{F}$, $[f]$ is monotonic with respect to $\geq$,*
- *$>$ is well founded,*
- *$> \circ \geq \subseteq >$,*
- *$\geq \circ > \subseteq >$,*
- *$>$ is transitive, and*
- *$\geq$ is reflexive and transitive.*[13]

[13] The last three properties are not required in [EWZ08, Def. 1]. However, these additional requirements conveniently correspond to our definition of reduction pairs, which in turn simplifies proofs in Section 6, and we are not aware of any weakly monotone algebra (as in [EWZ08, Def. 1]) used in termination proving practice where these requirements do not hold.

*The relations* $\geq, > \subseteq A \times A$ *carry over to relations* $\succsim_{\mathcal{A}}, \succ_{\mathcal{A}}$ *on terms* $s, t$*, where w.l.o.g.* $\mathcal{V}ar(s) \cup \mathcal{V}ar(t) = \{x_1, \ldots, x_k\}$*, by:*

$$s \succsim_{\mathcal{A}} t \quad \textit{iff} \quad \forall x_1, \ldots, x_k \in A \ : \ [s] \geq [t]$$
$$s \succ_{\mathcal{A}} t \quad \textit{iff} \quad \forall x_1, \ldots, x_k \in A \ : \ [s] > [t]$$

*If the carrier and the relations associated to a weakly monotone algebra are clear from the context, we often only give the interpretation explicitly.*

The pair of relations $(\geq, >)$ on the algebra $\mathcal{A}$ thus gives rise to a pair of relations $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$ on terms.

**Theorem 2.16** (Weakly Monotone Algebras Give Rise to Reduction Pairs)**.** *Let* $(\mathcal{A}, \geq, >)$ *be a weakly monotone algebra. Then* $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$ *is a reduction pair.*

*Proof.* [EWZ08, Theorem 2, Part 2] and its proof directly carry over to our definition of weakly monotone algebras and the induced term relations. This adapted theorem then implies the present theorem (cf. also [ZM09, Theorem 3], which makes a corresponding statement for polynomial interpretations). □

One of the most classic (weakly) monotone algebras in termination proving for term rewriting uses $\mathbb{N}$ as carrier and the standard orders $\geq$ and $>$ for comparison. All interpretation functions are *polynomial functions* (also simply called *polynomials*). Such functions can be defined by composing the operations "+" and "$*$" on variables and constants from $\mathbb{N}$. This way, one always obtains weakly monotonic interpretations.[14] Such interpretations are also known as *polynomial interpretations* [Lan79].

**Example 2.17** (Polynomial Interpretation for Example 2.14)**.** Consider the weakly monotone algebra $(\mathcal{A}, \geq, >)$ with carrier $\mathbb{N}$ and the following polynomial interpretation

$$[\mathsf{BITS}](x_1) = x_1 \tag{2.16}$$
$$[\mathsf{HALF}](x_1) = x_1 \tag{2.17}$$
$$[\mathsf{half}](x_1) = x_1 \tag{2.18}$$
$$[\mathsf{s}](x_1) = x_1 + 1 \tag{2.19}$$
$$[\mathsf{0}] = 0 \tag{2.20}$$

With this interpretation, the constraints from Example 2.14 are satisfied. Since there are no dependency pairs left in the resulting DP problem, there cannot be any infinite chain, and we can conclude termination of the TRS from Example 2.1.

[14] In recent work [NMZ10], the authors present criteria for weak monotonicity of certain polynomial functions on $\mathbb{N}$ where coefficients may be *negative*, e.g., $[f](x) = x^2 - x$. Since this result is independent of the contributions of this thesis, for simplicity we do not use weak monotonicity of such interpretations.

Note that in general it is not decidable whether $s \succ_{\mathcal{A}} t$ or $s \succsim_{\mathcal{A}} t$ holds. For instance, if $\mathcal{A}$ uses polynomial interpretations to $\mathbb{N}$, then undecidability of $s \succ_{\mathcal{A}} t$ and $s \succsim_{\mathcal{A}} t$ follows from undecidability of Hilbert's $10^{th}$ problem [Mat70]. Therefore, for automation it is common to use decidable sufficient criteria instead. For polynomial interpretations, a widely used criterion is the *absolute positiveness criterion*.

**Theorem 2.18** (Absolute Positiveness Criterion [HJ98])**.** *Let $p$ by a polynomial which w.l.o.g. has the following shape, where $p_0, \ldots, p_n$ are integers, where all $e_{i,j} \in \mathbb{N}$, and where for all $i$ we have $e_{1i} + \ldots + e_{ni} \geq 1$.*

$$p = p_0 + p_1\, x_1^{e_{11}} \ldots\, x_n^{e_{n1}} + \cdots + p_k\, x_1^{e_{1k}} \ldots\, x_n^{e_{nk}} \tag{2.21}$$

*Instead of inequalities or equalities of the form $p > 0$, $p \geq 0$, or $p = 0$ (where $x_1, \ldots, x_n$ are implicitly universally quantified over $\mathbb{N}$), it suffices[15] to require the following constraints [HJ98], respectively:*

$$\alpha_{p>0} = (p_0 > 0 \wedge p_1 \geq 0 \wedge \ldots \wedge p_k \geq 0) \tag{2.22}$$

$$\alpha_{p\geq 0} = (p_0 \geq 0 \wedge p_1 \geq 0 \wedge \ldots \wedge p_k \geq 0) \tag{2.23}$$

$$\alpha_{p=0} = (p_0 = 0 \wedge p_1 = 0 \wedge \ldots \wedge p_k = 0) \tag{2.24}$$

**Corollary 2.19.** *If $p$ is a polynomial as in* (2.21) *where for all $i$ we have $e_{1i} + \ldots + e_{ni} = 1$ (i.e., $p$ is a "linear polynomial"), then*

- *$p > 0$ holds iff $\alpha_{p>0}$ holds,*
- *$p \geq 0$ holds iff $\alpha_{p\geq 0}$ holds, and*
- *$p = 0$ holds iff $\alpha_{p=0}$ holds.*

To see that the given polynomial interpretation from Example 2.17 indeed satisfies the term constraints from Example 2.14, consider, e.g., the constraint $\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{BITS}(\mathsf{s}(\mathsf{half}(x)))$. Using this interpretation, we obtain the corresponding polynomial constraint $x + 2 > x + 1$ or, equivalently, $1 + 0 \cdot x > 0$. To check via the absolute positiveness criterion whether this constraint holds, instead we consider the constraint $\alpha_{1+0\cdot x>0} = 1 > 0 \wedge 0 \geq 0$. Since $\alpha_{1+0\cdot x>0}$ indeed holds, we get that $1 + 0 \cdot x > 0$ holds for all $x \in \mathbb{N}$. Therefore, the given interpretation indeed satisfies $\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{BITS}(\mathsf{s}(\mathsf{half}(x)))$.

For automation of the search for a polynomial interpretation to give rise to a reduction pair which solves a set of term constraints, [CMTU05] propose a *parametric* approach. The carrier $\mathbb{N}$ and the orders $\geq$ and $>$ are fixed beforehand, whereas suitable interpretations of the function symbols still need to be found. Initially, [CMTU05] fix the shape of the

[15] Of course, $\alpha_{p>0}$ and $\alpha_{p\geq 0}$ are sufficient, but in general not necessary for $p > 0$ and $p \geq 0$. Stronger criteria are presented, e.g., in [NMZ10].

polynomial interpretations, which is often specified by the degree of the polynomial, but they do not fix the values for the *coefficients*.

**Definition 2.20** (Parametric Polynomial, Interpretation, Coefficient [CMTU05])**.** *In a* parametric (polynomial) interpretation *each $n$-ary symbol $f$ is mapped to a* parametric polynomial *of the form*

$$a_0 + a_1\, x_1^{e_{11}} \dots x_n^{e_{n1}} \;+\; \dots \;+\; a_m\, x_1^{e_{1m}} \dots x_n^{e_{nm}} \tag{2.25}$$

*Here, the $e_{ij}$ are actual numbers, so the degree and the shape of the polynomials are fixed beforehand. In contrast, the coefficients $a_i$ are left open (i.e., they are* parametric *or* variable coefficients*).*

In practice, linear polynomial interpretations are a rather widely used choice for this setting.

**Example 2.21** (Parametric Interpretation for Example 2.14)**.** For the term constraints from Example 2.14, we could use a parametric interpretation $[\cdot]_{\mathcal{B}}$ with linear polynomials $[\mathsf{BITS}](x_1) = B_1\, x_1 + B_0$, $[\mathsf{HALF}](x_1) = H_1\, x_1 + H_0$, $[\mathsf{half}](x_1) = h_1\, x_1 + h_0$, $[\mathsf{s}](x_1) = s_1\, x_1 + s_0$, and $[\mathsf{0}] = z_0$. Here, $B_0, B_1, H_0, \dots$ are the parameters of the interpretation.

Extending a parametric interpretation to terms and term constraints then works exactly the same way as for a concrete interpretation.

**Example 2.22** (Example 2.21 continued)**.** In case of the term constraints (2.13), (2.14), and (2.15), we obtain the following polynomial constraints:

$$h_0 + h_1 z_0 \geq z_0 \tag{2.26}$$

$$h_0 + h_1 s_0 + h_1 s_1 z_0 \geq z_0 \tag{2.27}$$

$$h_0 + h_1 s_0 + h_1 s_1 s_0 + h_1 s_1^2 x \geq s_0 + s_1 h_0 + s_1 h_1 x \tag{2.28}$$

The term constraints (2.10), (2.11), and (2.12) yield the following constraints:

$$H_0 + H_1 s_0 + H_1 s_1 s_0 + H_1 s_1^2 x > H_0 + H_1 x \tag{2.29}$$

$$B_0 + B_1 s_0 + B_1 s_1 s_0 + B_1 s_1^2 x > H_0 + H_1 x \tag{2.30}$$

$$B_0 + B_1 s_0 + B_1 s_1 s_0 + B_1 s_1^2 x > B_0 + B_1 s_0 + B_1 s_1 h_0 + B_1 s_1 h_1 x \tag{2.31}$$

Note that only the variable $x$ is (implicitly) universally quantified.

Also absolute positiveness is applicable in the parametric setting.

**Example 2.23** (Example 2.22 continued)**.** Towards applying the absolute positiveness criterion, we simplify the constraints from Example 2.22 and obtain

$$h_0 + h_1 z_0 - z_0 \geq 0 \tag{2.32}$$
$$h_0 + h_1 s_0 + h_1 s_1 z_0 - z_0 \geq 0 \tag{2.33}$$
$$h_0 + h_1 s_0 + h_1 s_1 s_0 - s_0 - s_1 h_0 \; + \; (h_1 s_1^2 - s_1 h_1)\, x \geq 0 \tag{2.34}$$
$$H_1 s_0 + H_1 s_1 s_0 \; + \; (H_1 s_1^2 - H_1)\, x > 0 \tag{2.35}$$
$$B_0 + B_1 s_0 + B_1 s_1 s_0 - H_0 \; + \; (B_1 s_1^2 - H_1)\, x > 0 \tag{2.36}$$
$$B_1 s_1 s_0 - B_1 s_1 h_0 \; + \; (B_1 s_1^2 - B_1 s_1 h_1)\, x > 0 \tag{2.37}$$

Having transformed the constraints into the shape required in Theorem 2.18, we now replace all constraints $p > 0$ and $p \geq 0$ by $\alpha_{p>0}$ and $\alpha_{p\geq 0}$, respectively. This way, the universally quantified variables (here: $x$) are eliminated. We obtain the following conjunction:

$$h_0 + h_1 z_0 - z_0 \geq 0 \quad \wedge \tag{2.38}$$
$$h_0 + h_1 s_0 + h_1 s_1 z_0 - z_0 \geq 0 \quad \wedge \tag{2.39}$$
$$h_0 + h_1 s_0 + h_1 s_1 s_0 - s_0 - s_1 h_0 \geq 0 \quad \wedge \tag{2.40}$$
$$h_1 s_1^2 - s_1 h_1 \geq 0 \quad \wedge \tag{2.41}$$
$$H_1 s_0 + H_1 s_1 s_0 > 0 \quad \wedge \tag{2.42}$$
$$H_1 s_1^2 - H_1 \geq 0 \quad \wedge \tag{2.43}$$
$$B_0 + B_1 s_0 + B_1 s_1 s_0 - H_0 > 0 \quad \wedge \tag{2.44}$$
$$B_1 s_1^2 - H_1 \geq 0 \quad \wedge \tag{2.45}$$
$$B_1 s_1 s_0 - B_1 s_1 h_0 > 0 \quad \wedge \tag{2.46}$$
$$B_1 s_1^2 - B_1 s_1 h_1 \geq 0 \tag{2.47}$$

In order to make sure that only non-negative solutions are found, we explicitly introduce suitable side constraints on the parametric variables:

$$B_0 \geq 0 \wedge B_1 \geq 0 \wedge H_0 \geq 0 \wedge H_1 \geq 0 \wedge h_0 \geq 0 \wedge h_1 \geq 0 \wedge s_0 \geq 0 \wedge s_1 \geq 0 \wedge z_0 \geq 0 \tag{2.48}$$

So to solve a set of term constraints we have to show the *satisfiability* of such *Diophantine constraints.*[16] Definition 2.24 introduces their syntax and semantics.

**Definition 2.24** (Diophantine Constraints)**.** *Let $\mathcal{A}$ be a set of Diophantine variables. The set of* polynomials *$\mathcal{P}$ is the smallest set with*

[16] In honor of the ancient Greek mathematician *Diophantus of Alexandria*, formulas over *integer* arithmetic (in)equalities are often also referred to as *Diophantine* constraints.

- $\mathcal{A} \subseteq \mathcal{P}$ *and* $\mathbb{N} \subseteq \mathcal{P}$
- *If* $\{p,\ q\} \subseteq \mathcal{P}$ *then* $\{p+q,\ p-q,\ p*q\} \subseteq \mathcal{P}$

*The set of* Diophantine constraints *(also called* (quantifier-free) SMT-NIA formulas*)* $\mathcal{C}$ *is the smallest set with*

- $\{\mathit{true}, \mathit{false}\} \subseteq \mathcal{C}$
- *If* $\{p,\ q\} \subseteq \mathcal{P}$ *then* $\{p \geq q,\ p > q,\ p = q\} \subseteq \mathcal{C}$
- *If* $\{\alpha,\ \beta\} \subseteq \mathcal{C}$ *then* $\{\neg\alpha,\ \alpha \wedge \beta,\ \alpha \vee \beta,\ \alpha \to \beta,\ \alpha \leftrightarrow \beta,\ \alpha \oplus \beta\} \subseteq \mathcal{C}$

*A Diophantine interpretation* $\mathcal{D}$ *is a mapping* $\mathcal{D} : \mathcal{A} \to \mathbb{Z}$*. It can be extended to polynomials by defining* $\mathcal{D}(n) = n$ *for all* $n \in \mathbb{Z}$*,* $\mathcal{D}(p+q) = \mathcal{D}(p) + \mathcal{D}(q)$*,* $\mathcal{D}(p-q) = \mathcal{D}(p) - \mathcal{D}(q)$*, and* $\mathcal{D}(p*q) = \mathcal{D}(p) * \mathcal{D}(q)$*. It can also be extended to Diophantine constraints as follows (i.e., we then have* $\mathcal{D} : \mathcal{C} \to \{0,1\}$*, where* 0 *stands for "false" and* 1 *stands for "true"). As usual,* $\mathcal{D}$ *is called a* model *of a constraint* $\alpha$ *iff* $\mathcal{D}(\alpha) = 1$*.*

- $\mathcal{D}(\mathit{true}) = 1$*,* $\mathcal{D}(\mathit{false}) = 0$
- $\mathcal{D}(p \geq q) = 1$ *if* $\mathcal{D}(p) \geq \mathcal{D}(q)$ *and* $\mathcal{D}(p \geq q) = 0$*, otherwise*
- $\mathcal{D}(p > q) = 1$ *if* $\mathcal{D}(p) > \mathcal{D}(q)$ *and* $\mathcal{D}(p > q) = 0$*, otherwise*
- $\mathcal{D}(p = q) = 1$ *if* $\mathcal{D}(p) = \mathcal{D}(q)$ *and* $\mathcal{D}(p = q) = 0$*, otherwise*
- $\mathcal{D}(\neg\alpha) = 1$ *if* $\mathcal{D}(\alpha) = 0$ *and* $\mathcal{D}(\neg\alpha) = 0$*, otherwise,*

  *and similarly for the other Boolean connectives, where* $\oplus$ *is exclusive-or*

For example, let $a \in \mathcal{A}$ and let $\mathcal{D}$ with $\mathcal{D}(a) = 2$. Then $\mathcal{D}(2{*}a) = \mathcal{D}(2){*}\mathcal{D}(a) = 2{*}2 = 4$ and $\mathcal{D}(1+a) = 3$. Thus, $\mathcal{D}(2*a > 1+a) = 1$, since $4 > 3$.

Similarly, the constraint (2.40) is satisfied by the interpretation $\mathcal{D}$ with $\mathcal{D}(h_1) = \mathcal{D}(s_0) = \mathcal{D}(s_1) = 1$ and $\mathcal{D}(h_0) = 0$. This Diophantine interpretation instantiates the parametric polynomial interpretation $[\cdot]_{\mathcal{B}}$ from Example 2.21 with $[\mathsf{half}]_{\mathcal{B}} = h_1\,x_1 + h_0$ and $[\mathsf{s}]_{\mathcal{B}} = s_1\,x_1 + s_0$ to the concrete polynomial interpretation $[\cdot]_{\mathcal{A}}$ from Example 2.17 with $[\mathsf{half}]_{\mathcal{A}} = x_1$ and $[\mathsf{s}]_{\mathcal{A}} = x_1 + 1$ (i.e., we also write[17] $\mathcal{D}([\cdot]_{\mathcal{B}}) = [\cdot]_{\mathcal{A}}$ and $\mathcal{D}(\mathcal{B}) = \mathcal{A}$).

This way, we can use an assignment of the parametric coefficients (i.e., Diophantine variables) to natural numbers that satisfies the above constraints to instantiate the parameters of the parametric interpretation $[\cdot]_{\mathcal{B}}$. Any such assignment then yields a concrete interpretation $[\cdot]_{\mathcal{A}}$ such that the given term constraints hold for the corresponding reduction pair $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$.

[17] $\mathcal{D}$ only instantiates parametric coefficients like $h_0, h_1, s_0, s_1$. For variables $x_i$ which are not parametric coefficients (i.e., not Diophantine variables) we define $\mathcal{D}(x_i) = x_i$. Thus $\mathcal{D}(h_1\,x_1 + h_0) = 1{*}x_1 + 0 = x_1$.

Note that in Example 2.23 we obtain polynomial constraints which are *non-linear* over the parameters of our polynomial interpretation. This is the case even though the original parametric interpretation and the resulting polynomial constraints are linear over the (universally quantified) variables. The reason is that nesting of function symbols in general leads to non-linear constraints because multiplication of (parametric) coefficients takes place when the interpretation of a term is obtained.

This way, the search for a reduction pair on terms that satisfies certain term constraints is reduced to showing satisfiability of Diophantine constraints, i.e., SMT-NIA formulas. While also in general satisfiability of such constraints in quantifier-free non-linear integer arithmetic is undecidable (again, this is due to undecidability of Hilbert's $10^{th}$ problem [Mat70]), several powerful (albeit, of course, necessarily incomplete) techniques have been developed over the last years to solve such SMT-NIA problems.

For instance, in [FGM+07, Fuh07] we propose an (incomplete, but sound) encoding from satisfiability of Diophantine constraints (and hence SMT-NIA) to the satisfiability problem for propositional logic (the SAT problem). Our implementation of this encoding in the termination tool AProVE coupled with the SAT solver MiniSAT [ES04] (resulting in the SMT solver AProVE NIA) subsequently scored highest among all competing tools in the category QF_NIA (i.e., quantifier-free non-linear integer arithmetic) at the *SMT Competition 2011 (SMTCOMP '11)*, the annual international competition of SMT solvers.[18] A similar encoding is also given in [EWZ08].

In [BLN+09, BLO+12], the authors present an (incomplete, but sound) encoding from SMT-NIA to SMT-LIA, which has also been applied successfully for solving problem instances that arise in the setting of finding termination proofs for term rewriting.

[18] See also `http://www.smtcomp.org/2011/` for details.

# 3 Polynomials with Negative Constants

While the class of polynomial interpretations is already very useful in practice, it can be convenient to use also polynomials whose *constant addend is negative*. Hirokawa and Middeldorp present this class of interpretation functions in recent work [HM07]. As demonstrated by the tools TTT [HM07] and AProVE [GST06] in the termination competitions, such polynomials (in connection with the DP method) are very helpful in practice. To ensure well-definedness on the carrier set $\mathbb{N}$, Hirokawa and Middeldorp propose using the maximum of such a polynomial and the value 0 as an interpretation. Thus, for comparing two term interpretations, we also need to take the max-operator into account. To deal with expressions which may also contain $\max(\cdot, 0)$, they propose *polynomial approximations* for the interpretations as sufficient criteria for comparisons. While these approximations can be computed efficiently for a concrete interpretation, [HM07] does not present any satisfactory automation to find such approximations for a *parametric* interpretation. Due to the presence of "max", the parametric approach for generation of Diophantine constraints (i.e., SMT-NIA formulas) presented by [CMTU05] is not directly applicable here. We tackle this problem in the present chapter of this thesis by a suitable encoding to an SMT problem for non-linear integer arithmetic. Our experimental results using a SAT-based implementation as an SMT back-end in the termination prover AProVE [GST06] indicate performance improvements by orders of magnitude.

In Section 3.1 we review the setting of polynomial interpretations with a negative constant [HM07] as weakly monotone algebras for termination analysis. Section 3.2 then deals with means of automation by polynomial approximations [HM07] and present a novel SMT-NIA encoding for these approximations. Based on these approximations, in Section 3.3 we also present a syntactic criterion stating that for certain function symbols a negative constant does not increase power for given input term constraints. Moreover, we prove completeness of this criterion (i.e., it is never harmful to consider only non-negative constants for these symbols). Hence, this criterion allows to prune infeasible parts of the search space for the SMT-NIA encoding further, leading to smaller encodings and additional improvements for solving time. In Section 3.4 we provide empirical results confirming the practical usefulness of the contributions of this chapter. Section 3.5 then concludes this chapter.

## 3.1 Polynomial Interpretations with a Negative Constant

**Example 3.1** (Bits, version 2 [AG01])**.** To motivate the usefulness of polynomial interpretations with a negative constant, consider the following modification of Example 2.1, which is taken from [AG01, Example 4.28]. In this TRS $\mathcal{R}$ the rules (2.5) and (2.6) from Example 2.1 are replaced by the single rule (3.1).

$$\mathsf{half}(\mathsf{0}) \to \mathsf{0} \tag{2.1}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{0})) \to \mathsf{0} \tag{2.2}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{half}(x)) \tag{2.3}$$
$$\mathsf{bits}(\mathsf{0}) \to \mathsf{0} \tag{2.4}$$
$$\mathsf{bits}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \tag{3.1}$$

We obtain the following dependency pairs (where (3.2) and (3.3) are new):

$$\mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{HALF}(x) \tag{2.7}$$
$$\mathsf{BITS}(\mathsf{s}(x)) \to \mathsf{HALF}(x) \tag{3.2}$$
$$\mathsf{BITS}(\mathsf{s}(x)) \to \mathsf{BITS}(\mathsf{half}(\mathsf{s}(x))) \tag{3.3}$$

So the initial DP problem is $(\{(2.7), (3.2), (3.3)\}, \{(2.1), (2.2), (2.3), (2.4), (3.1)\})$. In a first step, we can remove the dependency pairs (2.7), (3.2) using the reduction pair processor with the polynomial interpretation $[\cdot]$ where $[\mathsf{BITS}](x_1) = [\mathsf{HALF}](x_1) = [\mathsf{half}](x_1) = x_1$, $[\mathsf{s}](x_1) = x_1 + 1$, and $[\mathsf{0}] = 0$. Afterwards, we only need to show finiteness of the DP problem $(\{(3.3)\}, \{(2.1), (2.2), (2.3), (2.4), (3.1)\})$.

Following Theorem 2.13, we need to find a ($\mathcal{C}_\varepsilon$-compatible) reduction pair $(\succsim, \succ)$ such that the following term constraints hold:

$$\mathsf{BITS}(\mathsf{s}(x)) \succ \mathsf{BITS}(\mathsf{half}(\mathsf{s}(x))) \tag{3.4}$$
$$\mathsf{half}(\mathsf{0}) \succsim \mathsf{0} \tag{2.13}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{0})) \succsim \mathsf{0} \tag{2.14}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succsim \mathsf{s}(\mathsf{half}(x)) \tag{2.15}$$

However, now no linear polynomial interpretation over $\mathbb{N}$ can be used to solve these term constraints. To see this, consider a linear polynomial interpretation with $[\mathsf{BITS}] = B_0 + B_1\, x_1$, $[\mathsf{half}] = h_0 + h_1\, x_1$, $[\mathsf{s}] = s_0 + s_1\, x_1$, and $[\mathsf{0}] = z_0$ (cf. Example 2.21).

From (3.4) we obtain the polynomial constraint

$$B_0 + B_1 s_0 + B_1 s_1 x > B_0 + B_1 h_0 + B_1 h_1 s_0 + B_1 h_1 s_1 x \tag{3.5}$$

equivalent to:

$$B_1 s_0 - B_1 h_0 - B_1 h_1 s_0 + (B_1 s_1 - B_1 h_1 s_1)\, x > 0 \tag{3.6}$$

The absolute positiveness criterion (which in this case is also a necessary criterion, cf. Corollary 2.19) yields:

$$B_1 s_0 - B_1 h_0 - B_1 h_1 s_0 > 0 \quad \wedge \tag{3.7}$$

$$B_1 s_1 - B_1 h_1 s_1 \geq 0 \tag{3.8}$$

From the term constraints (2.13), (2.14), and (2.15) we again obtain the constraints on the parameters (2.38), (2.39), (2.40), and (2.41) that also need to be satisfied:

$$h_0 + h_1 z_0 - z_0 \geq 0 \quad \wedge \tag{2.38}$$

$$h_0 + h_1 s_0 + h_1 s_1 z_0 - z_0 \geq 0 \quad \wedge \tag{2.39}$$

$$h_0 + h_1 s_0 + h_1 s_1 s_0 - s_0 - s_1 h_0 \geq 0 \quad \wedge \tag{2.40}$$

$$h_1 s_1^2 - s_1 h_1 \geq 0 \tag{2.41}$$

From (3.7) we can conclude that $B_1 s_0 > B_1 h_1 s_0$ must already hold. This is only possible if $B_1 s_0 \geq 1$, which requires $B_1 \geq 1$ and $s_0 \geq 1$. Moreover, we necessarily have $h_1 = 0$ because otherwise $B_1 s_0 > B_1 h_1 s_0$ cannot hold. This simplifies (3.7) to $B_1 s_0 > B_1 h_0$. Since $B_1 \geq 1$, this reduces to $s_0 > h_0$.

However, with $h_1 = 0$ the constraint (2.40) simplifies to $h_0 \geq s_0 + s_1 h_0$, which necessitates $h_0 \geq s_0$. Thus we have $s_0 > h_0 \geq s_0$, which is a contradiction.

Note that [GTSF06, Theorem 26] presents an improved version of the reduction pair processor which states that it suffices to consider the *usable rules w.r.t. the (implicit) argument filtering of the reduction pair* [GTSF06, Definition 21]. In our example, this means that if $B_1 = 0$, then we do not have to consider the constraints (2.38), (2.39), (2.40), and (2.41) anymore since then the $\mathsf{half}$-rules are not usable. However, with $B_1 = 0$ the constraint (3.7) simplifies to the contradiction $0 > 0$. Hence, a linear polynomial interpretation in combination with the improved reduction pair processor from [GTSF06, Theorem 26] would not help here either.[19]

Here, the hard term constraint is (3.4). The problem is that we essentially need to show that $\mathsf{s}(x) \succ \mathsf{half}(\mathsf{s}(x))$ holds, where $\mathsf{half}(\mathsf{s}(x))$ is syntactically greater than $\mathsf{s}(x)$.

One possible solution for this problem is to use a polynomial interpretation with a *negative constant* [HM07] (i.e., in (2.25) we have $a_0 < 0$) for the function symbol $\mathsf{half}$. All other coefficients still have to be natural numbers. Indeed, with $[\mathsf{half}](x_1) = x_1 - 1$

[19] See also [Zan09, Example 3.5] for similar reasoning showing that the reduction pair processor cannot be successfully applied on a certain problem if only linear polynomial interpretations on $\mathbb{N}$ are used.

and with $[\mathsf{s}](x) = x + 1$, the term constraint $\mathsf{s}(x) \succ \mathsf{half}(\mathsf{s}(x))$ leads to the polynomial constraint $x + 1 > x$, which holds for all $x \in \mathbb{N}$.

However, we still need to ensure well-definedness of such an interpretation on the carrier set $\mathbb{N}$. The problem is that a polynomial whose constant addend is negative can take negative values, e.g., if all variables are instantiated by 0. Therefore, we consider the maximum of such a polynomial and the value 0. This way, we always obtain a function that is both weakly monotonic and well defined. For instance, here the interpretation for $\mathsf{half}$ is modified to $[\mathsf{half}](x_1) = \max(x_1 - 1, 0)$.[20] The term constraint $\mathsf{s}(x) \succ \mathsf{half}(\mathsf{s}(x))$ then leads to the constraint $\max(x+1, 0) > \max(\max(x+1, 0) - 1, 0)$, i.e., we again have $x + 1 > x$.

**Example 3.2** (Solving Bits, version 2)**.** With a reduction pair that is based on an interpretation to $\mathbb{N}$ with $[\cdot]_{\mathcal{A}}$ given by $[\mathsf{BITS}]_{\mathcal{A}}(x_1) = x_1$, $[\mathsf{half}]_{\mathcal{A}}(x_1) = \max(x_1 - 1, 0)$, $[\mathsf{s}]_{\mathcal{A}}(x_1) = x_1 + 1$, and $[\mathsf{0}]_{\mathcal{A}} = 0$, we can solve all of the term constraints (3.4), (2.13), (2.14), and (2.15). This way, the reduction pair processor transforms the DP problem

$$(\{(3.3)\}, \{(2.1), (2.2), (2.3), (2.4), (3.1)\})$$

to the DP problem

$$(\varnothing, \{(2.1), (2.2), (2.3), (2.4), (3.1)\}).$$

Since the first component of this DP problem is empty, we can conclude that it is finite and thus also that the TRS from Example 3.1 is indeed terminating.

The next definition formalizes the notion of a polynomial interpretation with negative constants.

**Definition 3.3** (Polynomial Interpretation with Negative Constants)**.** *Let $\mathcal{F}$ be a signature. We call $\mathcal{A} = (\mathbb{N}, [\cdot]_{\mathcal{A}})$ a* polynomial interpretation with negative constants *if for all $f \in \mathcal{F}$, we have*

$$[f]_{\mathcal{A}}(x_1, \ldots, x_n) = \max(p_0, 0) \tag{3.9}$$

*for a polynomial $p_0 = a_0 + a_1\ x_1^{e_{11}} \ldots x_n^{e_{n1}} + \ldots + a_m\ x_1^{e_{1m}} \ldots x_n^{e_{nm}}$ where $a_0 \in \mathbb{Z}$, $a_1, \ldots, a_m, e_{11}, \ldots, e_{nm} \in \mathbb{N}$, and where for all $i$ we have $e_{1i} + \ldots + e_{ni} \geq 1$.*[21]

The following corollary states that we may indeed use polynomial interpretations with negative constants to obtain reduction pairs as ingredients for the reduction pair processor of Theorem 2.13.

[20] Strictly speaking, because of the max-operator such interpretations are not polynomials anymore. However, we follow [HM07] and by slight abuse of terminology nonetheless refer to this class of interpretations as "polynomial interpretations".

[21] We identify $\max(p, 0)$ with $p$ if $p$ cannot take negative values.

**Corollary 3.4** (Polynomial Interpretations with Negative Constants Give Rise to Weakly Monotone Algebras [HM07]). *Let $\mathcal{A} = (\mathbb{N}, [\cdot]_\mathcal{A})$ be a polynomial interpretation with negative constants as in Definition 3.3. Then $(\mathcal{A}, \geq, >)$ is a weakly monotone algebra.*

## 3.2 SMT-Based Automation

So now every interpretation for a function symbol $f$ has the shape $\max(p_0, 0)$ where $p_0$ is a polynomial with a possibly negative constant and non-negative coefficients in the non-constant part. Thus, previously existing criteria for comparing two term interpretations are not applicable anymore. The reason is that because of the max-operator, the interpretations are not polynomials any longer. Still, in this setting "max" is used only in a rather restricted way (one of the arguments is always the number 0). Therefore, in [HM07] Hirokawa and Middeldorp propose a sufficient criterion for showing that $\ell \succsim_\mathcal{A} r$ or $\ell \succ_\mathcal{A} r$ hold if the weakly monotone algebra $(\mathcal{A}, \geq, >)$ uses polynomial interpretations over $\mathbb{N}$ where negative constants may occur. For this purpose, they use *approximations* for $[\ell]_\mathcal{A}$ and $[r]_\mathcal{A}$ which are polynomials. This way, existing criteria for comparing term interpretations can be used again.

For the automatic synthesis of suitable interpretations, we are interested in *parametric* polynomial interpretations with variable coefficients. To find values for the coefficients for interpretations that do not use the max-operator, inequalities like $[\ell] \geq [r]$ are transformed into Diophantine constraints by building $\alpha_{[\ell]-[r]\geq 0}$ etc., cf. Corollary 2.19. Here, we simply require all coefficients of the polynomial $[\ell]-[r]$ to be non-negative. However, now $[\ell]-[r]$ contains the max-operator (i.e., it is no longer a polynomial). Thus, it is unclear how to transform $[\ell] \geq [r]$ into Diophantine constraints.

To solve this problem, let us first regard *concrete* polynomial interpretations (where the coefficients are actual numbers). In [HM07], Hirokawa and Middeldorp present an approach to transform inequalities like $[\ell] \geq [r]$ into ordinary polynomial inequalities without the max-operator. The idea is to define an under-approximation $[\cdot]^{left}$ and an over-approximation $[\cdot]^{right}$ which are "proper" polynomials that do not contain "max" anymore. Then instead of $[\ell] \geq [r]$ one requires $[\ell]^{left} \geq [r]^{right}$, which can be checked using techniques like the absolute positiveness criterion of Theorem 2.18.

**Definition 3.5** ($[\cdot]^{left}$ and $[\cdot]^{right}$ for Concrete Interpretations [HM07]). *For every polynomial $p$ we denote its* constant part *by $con(p)$ and the* non-constant part *$p - con(p)$ by $ncon(p)$, i.e., for a polynomial $p$ as in Definition 3.3, we have $con(p) = a_0$ and $ncon(p) = a_1\ x_1^{e_{11}} \dots x_n^{e_{n1}} + \dots + a_m\ x_1^{e_{1m}} \dots x_n^{e_{nm}}$. For any concrete polynomial interpretation with negative constants $[\cdot]_\mathcal{A}$ and any term $t$, we define the polynomials $[t]_\mathcal{A}^{left}$*

*and* $[t]^{right}_{\mathcal{A}}$ *as follows:*[22]

$$[t]^{left} = \begin{cases} t & \text{if } t \text{ is a variable} \\ 0 & \text{if } t = f(t_1,\ldots,t_n),\ ncon(p_1) = 0,\ \text{and } 0 > con(p_1) \\ p_1 & \text{if } t = f(t_1,\ldots,t_n),\ \text{otherwise} \end{cases}$$

$$[t]^{right} = \begin{cases} t & \text{if } t \text{ is a variable} \\ ncon(p_2) & \text{if } t = f(t_1,\ldots,t_n) \text{ and } 0 > con(p_2) \\ p_2 & \text{if } t = f(t_1,\ldots,t_n),\ \text{otherwise} \end{cases}$$

*where we have*

- $[f](x_1,\ldots,x_n) = \max(p_0(x_1,\ldots,x_n),0)$ *for a polynomial* $p_0$ *with a possibly negative constant and non-negative coefficients in the non-constant part, where* $p_0$ *must be weakly monotonic on* $\mathbb{Z}$*,*
- $p_1 = p_0([t_1]^{left},\ldots,[t_n]^{left})$*, and*
- $p_2 = p_0([t_1]^{right},\ldots,[t_n]^{right})$.

**Corollary 3.6** (Constant Part of a Polynomial)**.** *Let* $p(x_1,\ldots,x_n)$ *be a polynomial in the variables* $x_1,\ldots,x_n$*. Then* $con(p(x_1,\ldots,x_n)) = p(0,\ldots,0)$.

As shown in [HM07], we have $[t]^{left} \le [t] \le [t]^{right}$ for all terms $t$. Moreover, if the polynomial interpretation has no negative constants, then we have $[t]^{left} = [t] = [t]^{right}$. For the polynomial interpretation with $[\mathsf{half}]_{\mathcal{A}} = \max(x_1 - 1, 0)$, we obtain

$$[\mathsf{half}(x)]^{left}_{\mathcal{A}} = x - 1 \qquad [\mathsf{half}(x)]_{\mathcal{A}} = \max(x-1,0) \qquad [\mathsf{half}(x)]^{right}_{\mathcal{A}} = x \tag{3.10}$$

The reason is that for both $i \in \{1,2\}$, with $[\mathsf{half}](x) = \max(x-1,0)$ we have $p_i = x-1$ and thus $ncon(p_i) = x$ and $con(p_i) = -1$. This example also indicates that this approximation is incomplete. Obviously, $\mathsf{half}(x) \gtrsim_{\mathcal{A}} \mathsf{half}(x)$ holds since $\gtrsim_{\mathcal{A}}$ is reflexive, yet we have $[\mathsf{half}(x)]^{left}_{\mathcal{A}} \not\geq [\mathsf{half}(x)]^{right}_{\mathcal{A}}$. However, as also indicated by our experiments, this sufficient criterion for $\ell \gtrsim r$ is still very useful in practice.

**Example 3.7** (Solving Bits, version 2 using $[\cdot]^{left}$ and $[\cdot]^{right}$)**.** Using the interpretation from Example 3.2, we obtain $[\ell]^{left} > [r]^{right}$ for the dependency pair (3.3) and $[\ell]^{left} \ge [r]^{right}$ for all rules $\ell \to r \in \{(2.1),(2.2),(2.3)\}$. Thus, the proof step with the reduction pair processor in Example 3.2 can now easily be verified automatically using the approximations $[\cdot]^{left}$ and $[\cdot]^{right}$.

The disadvantage of Definition 3.5 is that one can only compute $[t]^{left}$ and $[t]^{right}$ for

[22] If $\mathcal{A}$ is clear from the context, we again omit the subscript $\mathcal{A}$.

concrete polynomial interpretations.[23] However, if one wants to find the coefficients of the polynomial interpretations automatically, then it would be better to start with *parametric* polynomial interpretations again where the coefficients $a_i$ in Definition 3.3 are left open (i.e., they are *variable coefficients*).

In our example, we would use a parametric interpretation $[\cdot]_{\mathcal{B}}$ with $[\mathsf{half}]_{\mathcal{B}}(x_1) = \max(a\,x_1 + \boldsymbol{b}, 0)$. Here, $a$ may only be instantiated by natural numbers, whereas we denote parameters like $\boldsymbol{b}$ that may be instantiated by integers in **bold** face. However, to compute $[\mathsf{half}(x)]^{left}_{\mathcal{B}}$ and $[\mathsf{half}(x)]^{right}_{\mathcal{B}}$ we would have to decide whether $ncon(p_1) = a\,x$ and $con(p_i) = \boldsymbol{b}$ are equal to or less than 0, respectively. This of course depends on the instantiation of the parametric coefficients $a$ and $\boldsymbol{b}$.

Therefore, we now modify Definition 3.5 to make it suitable for parametric polynomial interpretations. The idea is to introduce new Diophantine variables[24] $\boldsymbol{b}^{\boldsymbol{left}}_{\boldsymbol{t}}$ and $b^{right}_t$ for any term $t$ to denote the constant parts of $[t]^{left}$ and $[t]^{right}$, respectively. Note that $[t]^{left}$ is an under-approximation of $[t]$, so $\boldsymbol{b}^{\boldsymbol{left}}_{\boldsymbol{t}}$ can also take negative values. Similarly, $[t]^{right}$ is an over-approximation of $[t]$, so it suffices to consider non-negative values for $b^{right}_t$ (the constant part of $[t]^{right}$ is always non-negative, cf. Definition 3.5). Then we still need to create Diophantine constraints $\alpha^{left}_t$ and $\alpha^{right}_t$ which guarantee that $\boldsymbol{b}^{\boldsymbol{left}}_{\boldsymbol{t}}$ and $b^{right}_t$ are instantiated correctly. To this end, we express the conditions $ncon(p_1) = 0$ and $0 > con(p_i)$ from Definition 3.5 as Diophantine constraints.

**Definition 3.8** ($[\cdot]^{left}$ and $[\cdot]^{right}$ for Parametric Interpretations)**.** *For any parametric polynomial interpretation $[\cdot]_{\mathcal{A}}$ such that every concretization $[\cdot]_{\mathcal{D}(\mathcal{A})}$ is weakly monotonic on $\mathbb{Z}$ and for any term $t$, we define:*

- *If $t$ is a variable, then $[t]^{left} = t$, $[t]^{right} = t$, $\alpha^{left}_t = true$, and $\alpha^{right}_t = true$.*
- *If $t = f(t_1, \ldots, t_n)$, then*[25] $[t]^{left} = ncon(p_1) + \boldsymbol{b}^{\boldsymbol{left}}_{\boldsymbol{t}}$, $[t]^{right} = ncon(p_2) + b^{right}_t$,

$$\begin{aligned}\alpha^{left}_t \;=\; \alpha^{left}_{t_1} \wedge \ldots \wedge \alpha^{left}_{t_n} \wedge (\;\; & \alpha_{ncon(p_1)=0} \wedge 0 > con(p_1) \;\rightarrow\; \boldsymbol{b}^{\boldsymbol{left}}_{\boldsymbol{t}} = 0) \\ & \wedge (\neg(\alpha_{ncon(p_1)=0} \wedge 0 > con(p_1)) \;\rightarrow\; \boldsymbol{b}^{\boldsymbol{left}}_{\boldsymbol{t}} = con(p_1))\end{aligned}$$

$$\begin{aligned}\alpha^{right}_t = \;\; \alpha^{right}_{t_1} \wedge \ldots \wedge \alpha^{right}_{t_n} \wedge (\;\; & 0 > con(p_2) \;\rightarrow\; b^{right}_t = 0) \\ & \wedge (\neg(0 > con(p_2)) \;\rightarrow\; b^{right}_t = con(p_2))\end{aligned}$$

*Here, both $p_1$ and $p_2$ are defined as in Definition 3.5, and $\alpha_{ncon(p_1)=0}$ is defined as in Theorem 2.18.*

[23] Thus, previous implementations for polynomial interpretations with negative constants like T$_{\text{T}}$T and AProVE simply *test* several choices for the coefficients. More sophisticated algorithms for *systematically finding* coefficients like [CMTU05] only work for non-negative coefficients.

[24] Note that it can be helpful to use different ranges for different Diophantine variables. In particular, it is recommendable to use a larger range for the fresh variables $\boldsymbol{b}^{\boldsymbol{left}}_{\boldsymbol{t}}$ and $b^{right}_t$, since they stand for the values of complex polynomials $con(p_i)$ which contain sums and multiplications of many other Diophantine variables.

[25] Note that according to Definition 3.5, $[t]^{left} = ncon(p_1)$ if $ncon(p_1) = 0$ and $0 > con(p_1)$.

For $[\mathsf{half}]_{\mathcal{B}}(x_1) = \max(a\,x_1 + \boldsymbol{b}, 0)$, $t = \mathsf{half}(x)$, and $[t]_{\mathcal{B}}$, we have $ncon(p_0) = a\,x$, $con(p_0) = \boldsymbol{b}$, and:

$$[\mathsf{half}(x)]^{left}_{\mathcal{B}} = a\,x + \boldsymbol{b^{left}_t} \qquad \text{and} \qquad [\mathsf{half}(x)]^{right}_{\mathcal{B}} = a\,x + b^{right}_t \tag{3.11}$$

$$\alpha^{left}_t = ((a = 0 \wedge 0 > \boldsymbol{b}) \rightarrow \boldsymbol{b^{left}_t} = 0) \ \wedge \ (\neg(a = 0 \wedge 0 > \boldsymbol{b}) \rightarrow \boldsymbol{b^{left}_t} = \boldsymbol{b}) \tag{3.12}$$

$$\alpha^{right}_t = \qquad ((0 > \boldsymbol{b}) \rightarrow b^{right}_t = 0) \ \wedge \ (\neg(0 > \boldsymbol{b}) \rightarrow b^{right}_t = \boldsymbol{b}) \tag{3.13}$$

Theorem 3.9 shows that Definition 3.8 extends Definition 3.5 to parametric interpretations correctly.

**Theorem 3.9** (Correspondence of Definition 3.5 and 3.8)**.** *Let $\mathcal{D}$ be a Diophantine interpretation (which may only map* **bold** *variables also to negative numbers). Let $[\cdot]_{\mathcal{B}}$ be a parametric polynomial interpretation, and let $t$ be a term. Then $\mathcal{D}(\alpha^{left}_t) = 1$ implies $\mathcal{D}([t]^{left}_{\mathcal{B}}) = [t]^{left}_{\mathcal{D}(\mathcal{B})}$ and $\mathcal{D}(\alpha^{right}_t) = 1$ implies $\mathcal{D}([t]^{right}_{\mathcal{B}}) = [t]^{right}_{\mathcal{D}(\mathcal{B})}$.*

*Proof.* We use structural induction on $t$ and only prove the part $\mathcal{D}([t]^{left}_{\mathcal{B}}) = [t]^{left}_{\mathcal{D}(\mathcal{B})}$. The part $\mathcal{D}([t]^{right}_{\mathcal{B}}) = [t]^{right}_{\mathcal{D}(\mathcal{B})}$ is proved in an analogous way.

In this proof, we write "for $t$ we have $p_{1\mathcal{A}}\ldots$" to denote that with the interpretation $[\cdot]_{\mathcal{A}}$, the polynomial $p_1$ of Definition 3.5 or Definition 3.8 for $t$ is $p_{1\mathcal{A}}$ (analogously for $p_2$).

Thus, assume $\mathcal{D}(\alpha^{left}_t) = 1$.

If $t$ is a variable, then we have

$$\begin{array}{rcll} \mathcal{D}([t]^{left}_{\mathcal{B}}) & = & \mathcal{D}(t) & \text{by Definition 3.8} \\ & = & t & \text{cf. Footnote 17} \\ & = & [t]^{left}_{\mathcal{D}(\mathcal{B})} & \end{array}$$

Next, we regard the case $t = f(t_1, \ldots, t_n)$. For $t$, we have $p_{1\mathcal{B}} = p_0([t_1]^{left}_{\mathcal{B}}, \ldots, [t_n]^{left}_{\mathcal{B}})$ and $p_{1\mathcal{D}(\mathcal{B})} = \mathcal{D}(p_0)([t_1]^{left}_{\mathcal{D}(\mathcal{B})}, \ldots, [t_n]^{left}_{\mathcal{D}(\mathcal{B})})$, where $[f]_{\mathcal{B}}(x_1, \ldots, x_n) = \max(p_0(x_1, \ldots, x_n), 0)$.

First consider the subcase $ncon(p_{1\mathcal{D}(\mathcal{B})}) = 0$, and $0 > con(p_{1\mathcal{D}(\mathcal{B})})$. As $\mathcal{D}(\alpha^{left}_t) = 1$, we also have $\mathcal{D}(\alpha^{left}_{t_i}) = 1$ for all $i \in \{1, \ldots, n\}$. So the induction hypothesis implies $\mathcal{D}([t_i]^{left}_{\mathcal{B}}) = [t_i]^{left}_{\mathcal{D}(\mathcal{B})}$. Hence for $t$ we have

$$\begin{array}{rcll} \mathcal{D}(p_{1\mathcal{B}}) & = & \mathcal{D}(p_0([t_1]^{left}_{\mathcal{B}}, \ldots, [t_n]^{left}_{\mathcal{B}})) & \\ & = & \mathcal{D}(p_0)(\mathcal{D}([t_1]^{left}_{\mathcal{B}}), \ldots, \mathcal{D}([t_n]^{left}_{\mathcal{B}})) & \\ & = & \mathcal{D}(p_0)([t_1]^{left}_{\mathcal{D}(\mathcal{B})}, \ldots, [t_n]^{left}_{\mathcal{D}(\mathcal{B})}) & \text{by the induction hypothesis} \\ & = & p_{1\mathcal{D}(\mathcal{B})} & \end{array}$$

Therefore, we also have

$$\mathcal{D}(ncon(p_{1\mathcal{B}})) = ncon(p_{1\mathcal{D}(\mathcal{B})}) \qquad \text{and} \qquad \mathcal{D}(con(p_{1\mathcal{B}})) = con(p_{1\mathcal{D}(\mathcal{B})}).$$

This implies[26]

$$ncon(p_{1\mathcal{D}(\mathcal{B})}) = 0 \quad \text{iff} \quad \mathcal{D}(ncon(p_{1\mathcal{B}})) = 0 \quad \text{iff} \quad \mathcal{D}(\alpha_{ncon(p_{1\mathcal{B}})=0}) = 1$$

and

$$0 > con(p_{1\mathcal{D}(\mathcal{B})}) \quad \text{iff} \quad \mathcal{D}(0 > con(p_{1\mathcal{D}(\mathcal{B})})) = 1.$$

Therefore, $\mathcal{D}(\alpha_t^{left}) = 1$ implies $\mathcal{D}(\boldsymbol{b_t^{left}}) = 0$. Thus,

$$\begin{aligned}
\mathcal{D}([t]_{\mathcal{B}}^{left}) &= \mathcal{D}(ncon(p_{1\mathcal{B}}) + \boldsymbol{b_t^{left}}) \\
&= \mathcal{D}(ncon(p_{1\mathcal{B}})) + \mathcal{D}(\boldsymbol{b_t^{left}}) \\
&= ncon(p_{1\mathcal{D}(\mathcal{B})}) + 0 \\
&= 0 \\
&= [t]_{\mathcal{D}(\mathcal{B})}^{left}
\end{aligned}$$

Finally, for $t = f(t_1, \ldots, t_n)$ we regard the remaining subcase where $ncon(p_{1\mathcal{D}(\mathcal{B})}) \neq 0$ or $0 \leq con(p_{1\mathcal{D}(\mathcal{B})})$. Similar to the previous subcase, one can show that $\mathcal{D}(\alpha_t^{left}) = 1$ implies

$$\mathcal{D}(\boldsymbol{b_t^{left}}) \;=\; \mathcal{D}(con(p_{1\mathcal{B}})) \;=\; con(p_{1\mathcal{D}(\mathcal{B})}).$$

Hence,

$$\begin{aligned}
\mathcal{D}([t]_{\mathcal{B}}^{left}) &= \mathcal{D}(ncon(p_{1\mathcal{B}}) + \boldsymbol{b_t^{left}}) \\
&= \mathcal{D}(ncon(p_{1\mathcal{B}})) + \mathcal{D}(\boldsymbol{b_t^{left}}) \\
&= ncon(p_{1\mathcal{D}(\mathcal{B})}) + con(p_{1\mathcal{D}(\mathcal{B})}) \\
&= p_{1\mathcal{D}(\mathcal{B})} \\
&= [t]_{\mathcal{D}(\mathcal{B})}^{left}
\end{aligned}$$

□

For example, let $\mathcal{D}$ be a Diophantine interpretation which turns the parametric polynomial interpretation $[\cdot]_{\mathcal{B}}$ into the concrete interpretation $[\cdot]_{\mathcal{A}}$, where we have $\mathcal{D}(a) = 1$ and $\mathcal{D}(\boldsymbol{b}) = -1$. Then indeed, $\mathcal{D}([\mathsf{half}]_{\mathcal{B}}) = \mathcal{D}(\max(a\,x_1 + \boldsymbol{b}, 0)) = \max(x_1 - 1, 0) = [\mathsf{half}]_{\mathcal{A}}$. To satisfy the Diophantine constraints $\alpha_t^{left}$ and $\alpha_t^{right}$ in (3.12) and (3.13), we must have $\mathcal{D}(\boldsymbol{b_t^{left}}) = -1$ and $\mathcal{D}(b_t^{right}) = 0$. Then by (3.10) and (3.11), we indeed obtain

$$\begin{array}{lllllll}
\mathcal{D}([\mathsf{half}(x)]_{\mathcal{B}}^{left}) & = & \mathcal{D}(ax + \boldsymbol{b_t^{left}}) & = & x - 1 & = & [\mathsf{half}(x)]_{\mathcal{A}}^{left} \\
\mathcal{D}([\mathsf{half}(x)]_{\mathcal{B}}^{right}) & = & \mathcal{D}(ax + b_t^{right}) & = & x & = & [\mathsf{half}(x)]_{\mathcal{A}}^{right}
\end{array}$$

To summarize, we now proceed as follows to automate the search for a polynomial interpretation with negative constants for a set of term constraints $\ell \succ r$ or $\ell \succsim r$, as needed for the reduction pair processor:

[26] Note that by (2.24), $\alpha_{p=0}$ requires that all coefficients of $p$ must be 0. Thus, we indeed have $\mathcal{D}(\alpha_{p=0}) = \mathcal{D}(p = 0)$ for all Diophantine interpretations $\mathcal{D}$.

(i) Fix a parametric polynomial interpretation $[\cdot]_{\mathcal{B}}$ and transform the inequalities $\ell \succ r$ or $\ell \succsim r$ into $[\ell]^{left} - [r]^{right} > 0$ or $[\ell]^{left} - [r]^{right} \geq 0$, respectively. Add the conjunction of all corresponding constraints $\alpha_\ell^{left}$ and $\alpha_r^{right}$, and add constraints to ensure that non-bold Diophantine variables are only instantiated with non-negative numbers.

(ii) Replace $[\ell]^{left} - [r]^{right} > 0$ by $\alpha_{[\ell]^{left} - [r]^{right}} > 0$ and $[\ell]^{left} - [r]^{right} \geq 0$ by $\alpha_{[\ell]^{left} - [r]^{right}} \geq 0$, respectively.

(iii) Use an SMT-NIA solver to determine a solution $\mathcal{D}$ for the resulting Diophantine constraint. If the SMT solver finds a satisfying Diophantine interpretation, instantiate the parametric polynomial interpretation $[\cdot]_{\mathcal{B}}$ to a concrete interpretation $[\cdot]_{\mathcal{D}(\mathcal{B})}$. The corresponding weakly monotone algebra then induces a reduction pair that solves the initial term constraints.

Given a DP problem $(\mathcal{P}, \mathcal{R})$, in practice one often searches for a reduction pair that orients at least one rule from $\mathcal{P}$ strictly and all other rules from $\mathcal{P}$ and the usable rules $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ weakly. This can be expressed by the following constraint on terms, which [CGST12] calls the *usable rule constraint*:

$$\bigwedge_{\ell \to r \,\in\, \mathcal{P} \cup \mathcal{U}_{\mathcal{R}}(\mathcal{P})} \ell \succsim r \;\wedge \bigvee_{\ell \to r \,\in\, \mathcal{P}} \ell \succ r \tag{3.14}$$

For the case of using a weakly monotonic algebra, these term constraints become the following constraint over the algebra:

$$\bigwedge_{\ell \to r \,\in\, \mathcal{P} \cup \mathcal{U}_{\mathcal{R}}(\mathcal{P})} [\ell] \geq [r] \;\wedge \bigvee_{\ell \to r \,\in\, \mathcal{P}} [\ell] > [r] \tag{3.15}$$

In the setting of this chapter, we obtain the following sufficient criterion:

$$\bigwedge_{\ell \to r \,\in\, \mathcal{P} \cup \mathcal{U}_{\mathcal{R}}(\mathcal{P})} [\ell]^{left} \geq [r]^{right} \;\wedge \bigvee_{\ell \to r \,\in\, \mathcal{P}} [\ell]^{left} > [r]^{right} \tag{3.16}$$

In practice, one usually provides the SMT-NIA solver with additional information on the search space for the Diophantine variables. For instance, for solving term constraints in termination proving with polynomial interpretations, one usually only searches for values of the coefficients which are close to 0. This claim is substantiated by our experiments in [FGM+07] which indicate that for linear polynomial interpretations on $\mathbb{N}$, searching for coefficients which exceed the value 6 is hardly beneficial in practice. Thus, it pays off to restrict the search space in order to benefit from knowledge about the application domain (here: solving term constraints) which is hard to recover for the SMT solver or even gets lost in the encoding process altogether.

In the next section, we present a criterion which, given a set of term constraints, indicates for which symbols negative constants do not lead to any additional power in combination with the approximations from Definition 3.5. This way, especially transformational approaches to solving SMT-NIA instances such as the SAT encoding of [FGM+07, EWZ08] or the SMT-LIA encoding of [BLO+12] benefit. The reason is that the increased search space does not need to be represented explicitly on symbolic level (i.e., the encodings become smaller), and the actual search starts earlier.

## 3.3 A Necessary Criterion for Negative Constants

Consider again the term constraint (3.4)

$$\mathsf{BITS}(\mathsf{s}(x)) \succ \mathsf{BITS}(\mathsf{half}(\mathsf{s}(x))) \tag{3.4}$$

from Example 3.1. The reason that we can successfully apply the interpretation $[\cdot]_{\mathcal{A}}$ with a negative constant in $[\mathsf{half}]_{\mathcal{A}}(x_1) = \max(x_1 - 1, 0)$ from Example 3.2 is that the argument of $\mathsf{half}$ (i.e., $\mathsf{s}(x)$) is always interpreted as a value which is $> 0$ (i.e., $[\mathsf{s}(x)]_{\mathcal{A}} = x + 1$). This way, the interpretation of the argument $\mathsf{s}(x)$ compensates for the negative constant of $[\mathsf{half}]_{\mathcal{A}}(x_1) = \max(x_1 - 1, 0)$. Therefore, and since moreover the sum of $[\mathsf{s}(x)]_{\mathcal{A}} = x + 1$ and the negative constant $-1$ from $[\mathsf{half}]_{\mathcal{A}}(x_1)$ is non-negative, for the term $t = \mathsf{half}(\mathsf{s}(x))$ the approximation $[t]^{right}_{\mathcal{A}} = (x + 1) - 1 = x$ is identical to the actual interpretation $[t]_{\mathcal{A}} = \max(\max(x + 1, 0) - 1, 0) = \max(\,(x + 1)\, - 1, 0) = \max(x, 0) = x$.

In contrast, consider again the term constraints from Example 2.14:

$$\mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{HALF}(x) \tag{2.10}$$
$$\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{HALF}(x) \tag{2.11}$$
$$\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{BITS}(\mathsf{s}(\mathsf{half}(x))) \tag{2.12}$$
$$\mathsf{half}(\mathsf{0}) \succsim \mathsf{0} \tag{2.13}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{0})) \succsim \mathsf{0} \tag{2.14}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succsim \mathsf{s}(\mathsf{half}(x)) \tag{2.15}$$

Note that here we have no right-hand side of a term constraint where $\mathsf{half}$ is applied to a term that is always interpreted as a positive value. The only occurrences of $\mathsf{half}$ are in (2.12) and (2.15) in the subterm $\mathsf{half}(x)$. Since $x$ can also take the value 0, also the approximation $[\mathsf{half}(x)]^{right}_{\mathcal{A}} = x$ cannot benefit from the negative constant of $[\mathsf{half}]_{\mathcal{A}}(x_1) = \max(x_1 - 1, 0)$. In contrast, for the left-hand side of a term constraint, it is beneficial to use interpretations which map to high values. So we might as well use an interpretation $[\mathsf{half}](x_1) = x_1$ for the term constraints from Example 2.14 (which is the interpretation for $\mathsf{half}$ that is used in Example 2.17).

Also for the automation via parametric interpretations of Definition 3.8, it would be good if the search space could be restricted right from the start such that, e.g., for the constraints from Example 2.14, one would not try to search for negative values for the constant addend of [half] in the first place. This way, we can achieve an additional speed-up for the automation by omitting an infeasible part of the search space. This is especially important since the current state of the art of SMT-solving for non-linear integer arithmetic uses encoding-based approaches like bit-blasting to SAT [FGM+07, EWZ08] or translations to SMT-LIA [BLO+12]. In both cases, each additional value for some variable that must be considered for the search space leads to an increase of the size of the problem instance and also to an increase in encoding time. So even if the back-end solving engine should recognize early during the exploration of the search space that negative values are infeasible for certain parameters of the interpretation, we would nonetheless unnecessarily lose time while constructing the input for the solver.

Generalizing from these observations, we can now state a general criterion for applicability of negative constants. It is based on the notion of *potentially negative symbols* for a term $t$, i.e., symbols where an interpretation with a negative constant could be helpful.

**Definition 3.10** (Potentially Negative Symbols)**.** *For a term $t$ we define its* potentially negative symbols $\mathcal{N}(t)$ *as*

- $\mathcal{N}(x) = \varnothing$ *for every variable $x$ and*
- $\mathcal{N}(f(t_1, \ldots, t_n)) = \begin{cases} \bigcup_{1 \le i \le n} \mathcal{N}(t_i) \cup \{f\} & \text{if } \mathcal{F}(\{t_1, \ldots, t_n\}) \setminus \{f\} \neq \varnothing \\ \bigcup_{1 \le i \le n} \mathcal{N}(t_i) & \text{otherwise} \end{cases}$

*For a set of terms $T$ its* potentially negative symbols *are $\mathcal{N}(T) = \bigcup_{t \in T} \mathcal{N}(t)$.*

Note that we do not consider a symbol $f$ to be potentially negative for $t$ only because of a subterm $s = f(t_1, \ldots, t_n)$ of $t$ where only the symbol $f$ itself and variables occur in $s$. For example, consider a term $t = \mathsf{g}(\mathsf{f}(\mathsf{f}(x)))$. Here we get $\mathcal{N}(x) = \mathcal{N}(\mathsf{f}(x)) = \mathcal{N}(\mathsf{f}(\mathsf{f}(x))) = \varnothing$. Moreover, we get $\mathcal{N}(t) = \{\mathsf{g}\}$ since $\mathcal{F}(\{\mathsf{f}(\mathsf{f}(x))\}) \setminus \{\mathsf{g}\} = \{\mathsf{f}\} \neq \varnothing$.

Now consider the subterm $s = \mathsf{f}(\mathsf{f}(x))$. If we use a polynomial interpretation $[\cdot]_{\mathcal{A}}$ with a negative constant for the interpretation of $\mathsf{f}$, i.e., $[\mathsf{f}]_{\mathcal{A}}(x_1) = \max(p_0(x_1), 0)$ where $con(p_0) < 0$, then we also have $[\mathsf{f}(x)]_{\mathcal{A}}^{right} = ncon(\mathsf{f}([x]_{\mathcal{A}}^{right}))$ and thus $con([\mathsf{f}(x)]_{\mathcal{A}}^{right}) = con(ncon(\mathsf{f}([x]_{\mathcal{A}}^{right}))) = 0$. Together with the negative constant $con(p_0)$ for $\mathsf{f}$, we have $con(p_0([\mathsf{f}(x)]_{\mathcal{A}}^{right})) < 0$, which yields $con([s]_{\mathcal{A}}^{right}) = 0$ again. Thus, here we could as well use an interpretation $[\cdot]_{\mathcal{B}}$ where $[\mathsf{f}]_{\mathcal{B}}(x_1) = \max(ncon(p_0(x_1)), 0)$, i.e., an interpretation where the constant for $\mathsf{f}$ is 0. This would yield $[s]_{\mathcal{A}}^{right} = [s]_{\mathcal{B}}^{right} = 0$.

Based on this notion of potentially negative symbols, we now state a criterion for considering negative constants for certain function symbols, and we prove that it is a *necessary* criterion. In other words, we do not lose any power if we restrict the search

space for the constant of the interpretation for these function symbols to non-negative numbers.

Consider an automation for polynomial interpretations with negative constants where the approximations of Definition 3.5 are used to check if a term constraint $\ell \mathrel{(\succsim)} r$ holds ($\ell \succsim r$ and $\ell \succ r$ can be treated analogously). For term constraints $\ell_1 \mathrel{(\succsim)} r_1 \wedge \ldots \wedge \ell_n \mathrel{(\succsim)} r_n$ it then suffices to consider negative constants for symbols $f \in \mathcal{N}(\{r_1, \ldots, r_n\})$. Formally:

**Theorem 3.11** (Necessary Criterion for Negative Constants)**.** *Let $[\cdot]_\mathcal{A}$ be a polynomial interpretation with negative constants where Definition 3.5 is applicable (i.e., for each $f \in \mathcal{F}$, we have $[f]_\mathcal{A}(x_1, \ldots, x_n) = \max(p_f(x_1, \ldots, x_n), 0)$ where $p_f(x_1, \ldots, x_n)$ is weakly monotonic on $\mathbb{Z}$).*

*Moreover, let $[\ell_1]^{left}_\mathcal{A} \mathrel{(\geq)} [r_1]^{right}_\mathcal{A} \wedge \ldots \wedge [\ell_n]^{left}_\mathcal{A} \mathrel{(\geq)} [r_n]^{right}_\mathcal{A}$ hold. Let $[\cdot]_\mathcal{B}$ be defined as follows:*

- *$[f]_\mathcal{B} = ncon(p_f)$, if $f \notin \mathcal{N}(\{r_1, \ldots, r_n\})$, $[f]_\mathcal{A} = \max(p_f, 0)$, and $0 > con(p_f)$,*
- *$[f]_\mathcal{B} = [f]_\mathcal{A}$, otherwise.*

*Here $p_f(x_1, \ldots, x_n)$ is a polynomial with a possibly negative constant. Then also the polynomial constraint $[\ell_1]^{left}_\mathcal{B} \mathrel{(\geq)} [r_1]^{right}_\mathcal{B} \wedge \ldots \wedge [\ell_n]^{left}_\mathcal{B} \mathrel{(\geq)} [r_n]^{right}_\mathcal{B}$ holds.*

*Proof.* As in the proof of Theorem 3.9, we write "for $t$ we have $p_{1\mathcal{A}} \ldots$" to denote that with the interpretation $[\cdot]_\mathcal{A}$, the polynomial $p_1$ of Definition 3.5 or Definition 3.8 for $t$ is $p_{1\mathcal{A}}$ (analogously for $p_2$).

Without loss of generality, assume that $[\ell_1]^{left}_\mathcal{A} \geq [r_1]^{right}_\mathcal{A} \wedge \ldots \wedge [\ell_n]^{left}_\mathcal{A} \geq [r_n]^{right}_\mathcal{A}$ holds. This is not a restriction since on $\mathbb{N}$ we can express $x > y$ equivalently as $x \geq y + 1$. Let $[\cdot]_\mathcal{A}, [\cdot]_\mathcal{B}$ be defined as above. To prove the theorem, we now show the following stronger claims:

(i) For *all* $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have
   - $[t]^{left}_\mathcal{B} \geq [t]^{left}_\mathcal{A}$ and
   - $ncon([t]^{left}_\mathcal{B}) = 0$ iff $ncon([t]^{left}_\mathcal{A}) = 0$.

(ii) For all subterms $r$ of any term in $\{r_1, \ldots, r_n\}$, we have $[r]^{right}_\mathcal{A} = [r]^{right}_\mathcal{B}$

We prove Claim (i) by induction over the structure of $t$. If $t$ is a variable, we have $[t]^{left}_\mathcal{B} = t \geq t = [t]^{left}_\mathcal{A}$. Moreover, we have both $ncon([t]^{left}_\mathcal{B}) = t \neq 0$ and $ncon([t]^{left}_\mathcal{A}) = t \neq 0$.

Now let $t = f(t_1, \ldots, t_n)$. Then by the induction hypothesis we have for $1 \leq i \leq n$: $[t_i]^{left}_\mathcal{B} \geq [t_i]^{left}_\mathcal{A}$ and $ncon([t_i]^{left}_\mathcal{B}) = 0$ iff $ncon([t_i]^{left}_\mathcal{A}) = 0$. There are two cases to consider:

(a) $[f]_\mathcal{A}(x_1, \ldots, x_n) = [f]_\mathcal{B}(x_1, \ldots, x_n) = \max(p_0(x_1, \ldots, x_n), 0)$

(b) $[f]_\mathcal{A}(x_1, \ldots, x_n) = \max(p_0(x_1, \ldots, x_n), 0), [f]_\mathcal{B}(x_1, \ldots, x_n) = ncon(p_0(x_1, \ldots, x_n))$, and $0 > con(p_0)$

We only give the proof for Case (a); the proof for Case (b) is analogous.

So let $[f]_{\mathcal{A}}(x_1,\ldots,x_n) = [f]_{\mathcal{B}}(x_1,\ldots,x_n) = \max(p_0(x_1,\ldots,x_n),0)$.
By the induction hypothesis and by monotonicity of $p_0$ on $\mathbb{Z}$, we have:

$$p_{1\mathcal{B}} = p_0([t_1]^{left}_{\mathcal{B}},\ldots,[t_n]^{left}_{\mathcal{B}}) \geq p_0([t_1]^{left}_{\mathcal{A}},\ldots,[t_n]^{left}_{\mathcal{A}}) = p_{1\mathcal{A}} \tag{3.17}$$

We now show that also the following property of $p_1$ holds:

$$ncon(p_{1\mathcal{B}}) = 0 \quad \text{iff} \quad ncon(p_{1\mathcal{A}}) = 0 \tag{3.18}$$

In the following, we call a function $p(x_1,\ldots,x_n)$ *independent from its* $i^{th}$ *argument* iff for all $v_1,\ldots,v_n,w$ we have $p(v_1,\ldots,v_i,\ldots,v_n) = p(v_1,\ldots,w,\ldots,v_n)$. Now to see that (3.18) holds, note that:

$ncon(p_{1\mathcal{A}}) = 0$
iff $\bigwedge_{1\leq i\leq n} ncon([t_i]^{left}_{\mathcal{A}}) = 0 \vee p_0(x_1,\ldots,x_n)$ is independent from its $i^{th}$ argument
iff $\bigwedge_{1\leq i\leq n} ncon([t_i]^{left}_{\mathcal{B}}) = 0 \vee p_0(x_1,\ldots,x_n)$ is indep. from its $i^{th}$ arg. (by ind. hyp.)
iff $ncon(p_{1\mathcal{B}}) = 0$

By (3.18) it suffices to consider the following two cases:

- $ncon(p_{1\mathcal{A}}) \neq 0$ and $ncon(p_{1\mathcal{B}}) \neq 0$
- $ncon(p_{1\mathcal{A}}) = 0$ and $ncon(p_{1\mathcal{B}}) = 0$

If we have $ncon(p_{1\mathcal{A}}) \neq 0$ and $ncon(p_{1\mathcal{B}}) \neq 0$, then we also have $[t]^{left}_{\mathcal{B}} = p_{1\mathcal{B}} \geq p_{1\mathcal{A}} = [t]^{left}_{\mathcal{A}}$, $ncon([t]^{left}_{\mathcal{A}}) \neq 0$, and $ncon([t]^{left}_{\mathcal{B}}) \neq 0$.

So now consider $ncon(p_{1\mathcal{A}}) = 0$ and $ncon(p_{1\mathcal{B}}) = 0$. If $0 \leq con(p_{1\mathcal{A}})$, then we have

$$\begin{aligned}
[t]^{left}_{\mathcal{A}} &= p_{1\mathcal{A}} \\
&\leq p_{1\mathcal{B}} && \text{by (3.17)} \\
&= [t]^{left}_{\mathcal{B}} && \text{(3.17) and } con(p_{1\mathcal{A}}) \geq 0 \text{ imply } con(p_{1\mathcal{B}}) \geq 0
\end{aligned}$$

This also implies $ncon([t]^{left}_{\mathcal{A}}) = 0$ and $ncon([t]^{left}_{\mathcal{B}}) = 0$.

If however $0 > con(p_{1\mathcal{A}})$, we have $[t]^{left}_{\mathcal{A}} = 0$. If also $0 > con(p_{1\mathcal{B}})$, we get $[t]^{left}_{\mathcal{B}} = 0 \geq 0 = [t]^{left}_{\mathcal{A}}$. If $0 \leq con(p_{1\mathcal{B}})$, we get $[t]^{left}_{\mathcal{B}} = p_{1\mathcal{B}} = con(p_{1\mathcal{B}}) \geq 0 = [t]^{left}_{\mathcal{A}}$. Both for $0 > con(p_{1\mathcal{B}})$ and for $0 \leq con(p_{1\mathcal{B}})$, we thus also have $ncon([t]^{left}_{\mathcal{A}}) = 0$ and $ncon([t]^{left}_{\mathcal{B}}) = 0$.

We prove Claim (ii) by induction over the structure of the term $r$. If $r$ is a variable, we have $[r]^{right}_{\mathcal{A}} = r = [r]^{right}_{\mathcal{B}}$.

Now consider $r = f(t_1, \ldots, t_n)$. There are two cases: First, consider $[f]_{\mathcal{A}}(x_1, \ldots, x_n) = [f]_{\mathcal{B}}(x_1, \ldots, x_n) = \max(p_0(x_1, \ldots, x_n), 0)$. By the induction hypothesis we have:

$$p_{2\mathcal{A}} = p_0([t_1]^{right}_{\mathcal{A}}, \ldots, [t_n]^{right}_{\mathcal{A}}) = p_0([t_1]^{right}_{\mathcal{B}}, \ldots, [t_n]^{right}_{\mathcal{B}}) = p_{2\mathcal{B}} \tag{3.19}$$

If $0 \leq con(p_{2\mathcal{B}})$, then we have

$$\begin{aligned} [r]^{right}_{\mathcal{B}} &= p_{2\mathcal{B}} \\ &= p_{2\mathcal{A}} && \text{by (3.19)} \\ &= [r]^{right}_{\mathcal{A}} \end{aligned}$$

Now let $0 > con(p_{2\mathcal{B}})$. Then we have $[r]^{right}_{\mathcal{B}} = ncon(p_{2\mathcal{B}}) = ncon(p_{2\mathcal{A}}) = [r]^{right}_{\mathcal{A}}$.

Thus, let $f \notin \mathcal{N}(\{r_1, \ldots, r_n\})$, $0 > con(p_0)$, $[f]_{\mathcal{A}}(x_1, \ldots, x_n) = \max(p_0(x_1, \ldots, x_n), 0)$, and $[f]_{\mathcal{B}}(x_1, \ldots, x_n) = ncon(p_0)(x_1, \ldots, x_n)$.

Since $f \notin \mathcal{N}(\{r_1, \ldots, r_n\})$ and since $r$ is a subterm of a term from $\{r_1, \ldots, r_n\}$, we can conclude that $r \in \mathcal{T}(\{f\}, \mathcal{V})$, i.e., $r$ only contains variables and the function symbol $f$.

To complete the proof, we will use the following statement:

$$\begin{aligned} &\text{Let } q_1, \ldots, q_n, p \text{ be polynomials with } con(q_1) = \ldots = con(q_n) = 0. \\ &\text{Then } con(p(q_1, \ldots, q_n)) = con(p(x_1, \ldots, x_n)). \end{aligned} \tag{3.20}$$

To see that (3.20) holds, let $q_1, \ldots, q_n, p$ be polynomials with $con(q_1) = \ldots = con(q_n) = 0$. We then have:

$$\begin{aligned} & con(p(q_1, \ldots, q_n)) \\ &= p(q_1, \ldots, q_n)(0, \ldots, 0) && \text{by Corollary 3.6} \\ &= p(q_1(0, \ldots, 0), \ \ldots \ , q_n(0, \ldots, 0)) \\ &= p(con(q_1), \ldots, con(q_n)) \\ &= p(0, \ldots, 0) \\ &= con(p) && \text{by Corollary 3.6} \end{aligned}$$

Using (3.20), we now show the following auxiliary statement by induction:

$$\text{For all } u \in \mathcal{T}(\{f\}, \mathcal{V}), \text{ we have } con([u]^{right}_{\mathcal{A}}) = 0. \tag{3.21}$$

If $u$ is a variable, we have $con([u]^{right}_{\mathcal{A}}) = con(u) = 0$. So let $u = f(u_1, \ldots, u_n)$. Then the induction hypothesis of (3.21) states $con([u_1]^{right}_{\mathcal{A}}) = \ldots = con([u_n]^{right}_{\mathcal{A}}) = 0$. Thus, we can apply the statement (3.20) and get $con(p_0([u_1]^{right}_{\mathcal{A}}, \ldots, [u_n]^{right}_{\mathcal{A}})) = con(p_0(x_1, \ldots, x_n)) < 0$. Thus, we have $[u]^{right}_{\mathcal{A}} = ncon(p_0([u_1]^{right}_{\mathcal{A}}, \ldots, [u_n]^{right}_{\mathcal{A}}))$ and $con([u]^{right}_{\mathcal{A}}) = 0$.

Having proved (3.21), we continue with the induction step for Claim (ii). As $r \in \mathcal{T}(\{f\}, \mathcal{V})$, by (3.21) we have $con([r]_{\mathcal{A}}^{right}) = 0$. Thus:

$$
\begin{aligned}
&[r]_{\mathcal{A}}^{right} \\
&= ncon([r]_{\mathcal{A}}^{right}) \\
&= ncon(p_0([t_1]_{\mathcal{A}}^{right}, \ldots, [t_n]_{\mathcal{A}}^{right})) \\
&= ncon(p_0([t_1]_{\mathcal{A}}^{right}, \ldots, [t_n]_{\mathcal{A}}^{right})) + con(p_0([t_1]_{\mathcal{A}}^{right}, \ldots, [t_n]_{\mathcal{A}}^{right})) \\
&\quad - con(p_0([t_1]_{\mathcal{A}}^{right}, \ldots, [t_n]_{\mathcal{A}}^{right})) \\
&= p_0([t_1]_{\mathcal{A}}^{right}, \ldots, [t_n]_{\mathcal{A}}^{right}) - con(p_0([t_1]_{\mathcal{A}}^{right}, \ldots, [t_n]_{\mathcal{A}}^{right})) \\
&= (p_0 - con(p_0))([t_1]_{\mathcal{A}}^{right}, \ldots, [t_n]_{\mathcal{A}}^{right}) && \text{using (3.20) with:} \\
& && \text{for all } i,\ t_i \in \mathcal{T}(\{f\}, \mathcal{V}), \text{ so by (3.21) } con([t_i]_{\mathcal{A}}^{right}) = 0 \\
&= (ncon(p_0))([t_1]_{\mathcal{A}}^{right}, \ldots, [t_n]_{\mathcal{A}}^{right}) \\
&= (ncon(p_0))([t_1]_{\mathcal{B}}^{right}, \ldots, [t_n]_{\mathcal{B}}^{right}) && \text{by induction hypothesis} \\
&= [r]_{\mathcal{B}}^{right}
\end{aligned}
$$

□

Note that as a special case, it is *always* safe to permit only non-negative constant addends for constant function symbols, even if one does not use the approximations from Definition 3.5. The reason is that a constant symbol has no arguments, and $\max(c, 0)$ always takes the value 0 if $c$ is negative.

It is also worth noting that in Theorem 3.11, the restriction to the approximations of Definition 3.5 is necessary for completeness. This is demonstrated by the following example.

**Example 3.12** (Completeness of Theorem 3.11 Requires Approximations)**.** Consider again Example 3.1, where we now also add the rule $\mathsf{half}(x) \to \mathsf{g}(x)$ to the TRS. Similar to Example 3.1, we then need to solve the following term constraints to prove termination:

$$\mathsf{BITS}(\mathsf{s}(x)) \succ \mathsf{BITS}(\mathsf{half}(\mathsf{s}(x))) \tag{3.4}$$
$$\mathsf{half}(\mathsf{0}) \succsim \mathsf{0} \tag{2.13}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{0})) \succsim \mathsf{0} \tag{2.14}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succsim \mathsf{s}(\mathsf{half}(x)) \tag{2.15}$$
$$\mathsf{half}(x) \succsim \mathsf{g}(x) \tag{3.22}$$

If we extend the interpretation $[\cdot]_{\mathcal{A}}$ from Example 3.2 to an interpretation $[\cdot]_{\mathcal{A}'}$ to interpret $\mathsf{g}$ by $[\mathsf{g}]_{\mathcal{A}'}(x_1) = \max(x_1 - 1, 0)$, the term constraint (3.22) yields the constraint $\max(x - 1, 0) \geq \max(x - 1, 0)$. Since this constraint obviously holds ($\geq$ is reflexive), we can conclude termination also of this extended TRS.

But the criterion from Theorem 3.11 would not consider $\mathsf{g}$ to be a potentially negative symbol and thus would not allow to use the negative constant $-1$ as part of an interpretation for $\mathsf{g}$. This shows that using this criterion, in general we lose interpretations that would actually have solved our term constraints.

However, with the approximations from Definition 3.5, we obtain $[\mathsf{half}(x)]^{left}_{\mathcal{A}'} = x - 1 \not\geq x = [\mathsf{g}(x)]^{right}_{\mathcal{A}'}$. Indeed, now there is no possible extension of $[\cdot]_{\mathcal{A}}$ to the new symbol $\mathsf{g}$ that would allow us to solve this term constraint with these approximations. The reason is that $[\mathsf{half}(x)]^{left}_{\mathcal{A}} = x - 1$ can also become negative (for $x = 0$), but $[\mathsf{g}(x)]^{right}$ is always non-negative, regardless of the interpretation used for $\mathsf{g}$.

Note that Theorem 3.11 would not allow negative constants for the interpretation of $\mathsf{g}$ in the first place. So with the approximations of Definition 3.5, we indeed do not lose any power if we use the criterion from Theorem 3.11.

## 3.4 Experiments

We implemented our new SMT-based approach for polynomial interpretations with negative constants in the termination prover AProVE [GST06]. To solve the arising SMT-NIA problems, we used our implementation of our SAT-based approach from [FGM+07] with the MiniSAT solver [ES04] as back-end. To convert formulas to CNF, we applied SAT4J's [LP10] implementation of Tseitin's algorithm [Tse68]. As our implementation of our SAT-based approach from [FGM+07] scored highest among all the competing tools at the SMT Competition 2011 in the category *QF_NIA*, we believe that it is a natural choice for this setting. This way, in our experiments it is a SAT solver like MiniSAT that makes the choice of the interpretation that should be used.

To evaluate our new SAT-based implementation of polynomial interpretations with negative constants (AProVE-SAT), we compare it with the non-SAT-based implementations in the termination tools AProVE 1.2 and T$_{\text{T}}$T [HM07]. To investigate the effect of negative constants, we moreover compare with the results for conventional polynomial interpretations without negative constants (cf. in Section 2.3) reported in [FGM+07]. As for AProVE-SAT, the solving back-end of this configuration is the SAT solver MiniSAT. Therefore, we name this configuration AProVE-SAT $\geq 0$.

We tested the tools on all 865 TRSs from the TPDB, version 3.2.[27] This is the collection of examples used in the International Termination Competition of 2006.[28] For our experiments to compare the different search procedures for polynomial interpretations with a negative constants, the tools were run on an AMD Athlon 64 at 2.2 GHz. To measure the effect of the different implementations for polynomial interpretations with negative

[27] This version of the data base is available from `http://www.lri.fr/~marche/tpdb/`.

[28] These experiments were conducted already for an earlier version of this chapter published in [FGM+07]. Therefore, we report on our results with this previous version of the TPDB.

| | AProVE-SAT | | | AProVE 1.2 | | | TTT | | | AProVE-SAT $\geq 0$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Range | Yes | TO | Time | Yes | TO | Time | Yes | TO | Time | Yes | TO | Time |
| 1 | 440 | 0 | 98.0 | 441 | 22 | 1863.7 | 341 | 106 | 7307.3 | 421 | 0 | 45.5 |
| 2 | 479 | 1 | 305.4 | 460 | 126 | 8918.3 | 360 | 181 | 12337.3 | 431 | 0 | 91.8 |
| 3 | 483 | 4 | 1092.4 | 434 | 221 | 15570.9 | 361 | 247 | 16927.7 | 434 | 0 | 118.6 |

Figure 3.13: Empirical Results for Different Implementations of Negative Polynomials

constants, we configured all tools to use only a basic version of the DP method and no other termination technique.

For each example, we imposed a time limit of 60 seconds (corresponding to the way tools are evaluated in the annual competition). In Figure 3.13, the columns "*Yes*" and "*TO*" show the number of TRSs for which proving termination with the given configuration succeeds or times out. Finally, "*Time*" gives the total time in seconds needed for analyzing all 865 examples.[29] The column "*Range*" specifies the range of the coefficients of polynomials (i.e., if the "*Range*" is $n$, then we only searched for coefficients from $\{0, \ldots, n\}$ and constants from $\{-n, \ldots, n\}$). For the configuration AProVE-SAT $\geq 0$, we searched for values from $\{0, \ldots, n\}$ also for the constants.

The SAT-based configuration for the search for polynomial orders with a negative constant is much faster and substantially more powerful than the non-SAT-based ones.[30] Even for larger ranges, only few timeouts occur, whereas increasing the range with the previous approaches leads to a drastic increase in the number of timeouts.

Another interesting point is the comparison of power and speed of the approaches based on polynomials with a negative constant and on standard polynomials without negative

[29] For these experiments we executed AProVE in *batch mode*, i.e., we started up AProVE only once, performed runs on several TRS for just-in-time-compilation, and only then performed the measurements for all TRSs from the TPDB in the same session. This way, we make sure that we do not measure the time for startup of the Java Virtual Machine and AProVE and also just-in-time compilation. Since this overhead amounts to at least 0.9 seconds per TRS, i.e., more than 770 seconds for the whole TPDB, runtime of some of the configurations in Figure 3.13 would be affected by more than an order of magnitude, which is hardly non-negligible. This is also interesting since we compare to tools which are statically compiled to native machine code in advance such as TTT and thus do not suffer from this overhead.

[30] Note that for range 1, AProVE 1.2 discovers a termination proof where AProVE-SAT does not. This is the case for the example `TRS/AProVE_06/identity.xml` (prior to a recent reorganization of the TPDB, this TRS could be found in the TPDB as `TRS/Thiemann/identity.trs`). Here AProVE 1.2 uses an interpretation with $[\mathsf{half}](x_1) = \max(x_1 - 1, 0)$ and $[\mathsf{g}](x_1, x_2) = \max(x_2 - 1, 0)$ to solve the term constraint $\mathsf{half}(x) \gtrsim \mathsf{g}(\mathsf{h}, x)$. With this interpretation, this yields the constraint $\max(x - 1, 0) \geq \max(x - 1, 0)$. The search procedure of AProVE 1.2 recognizes that we have identical expressions on both sides of the (non-strict) inequality and concludes that the inequality holds. In contrast, here approaches like AProVE-SAT that only use the approximations from [HM07] (cf. Definition 3.5) would obtain the polynomial constraint $x - 1 \geq x$, which obviously does not hold. This indicates that incompleteness of the approximations can also surface in practical examples. However, the fact that only a single example from the whole TPDB is affected shows that in practice, incompleteness of the approximations is only rarely noticeable, at least when compared to AProVE 1.2. Moreover, this particular example is easily solved by AProVE also, e.g., with our SMT-NIA-based implementation of *rational polynomial orders* [Luc05, FNO+08].

| | AProVE-SAT | | | AProVE-SAT + Theorem 3.11 | | |
|---|---|---|---|---|---|---|
| Range | Yes | TO | Time | Yes | TO | Time |
| 1 | 561 | 14 | 3627.5 *(2333.3)* | 561 | 14 | 3482.7 *(2188.5)* |
| 2 | 620 | 18 | 4182.2 *(2888.0)* | 620 | 16 | 3900.5 *(2606.3)* |
| 3 | 626 | 25 | 5209.9 *(3915.7)* | 625 | 23 | 4576.9 *(3282.7)* |

Figure 3.14: Empirical Effects of Theorem 3.11

constants, respectively. Here, our results indicate a significant increase in power by admitting negative constants. In the most powerful configuration with negative constants, over 10.1 % more examples can be solved compared to the corresponding configuration where only non-negative constants are considered. At the same time, the increases in runtime are only moderate and timeouts are still few.

To experiment with our implementation and for further details on the above experiments, please see `http://aprove.informatik.rwth-aachen.de/eval/SATPOLO/`.

Finally, we recently conducted experiments to assess the impact of Theorem 3.11. Here, we used an Intel Xeon 5140 at 2.33 GHz, and we executed AProVE on the 1438 TRSs from the TPDB (version 8.0.1) that are used for the "TRS standard" category. The results are given in Figure 3.14. Since we did not run AProVE in batch mode for this experiment, it is sensible to disregard the constant overhead of approx. 0.9 seconds induced by startup of the Java Virtual Machine and of AProVE. Therefore, here we give both the measures runtimes and in parentheses also the runtimes where the overhead of 0.9 seconds per example has been subtracted. This way, one can compare both the measured raw data for the runtimes and also the processed runtimes where the effect of the constant overhead has been removed.

We observe a reduction in runtime after startup of up to 16.2 % for range 3. This shows that our novel complete criterion from Theorem 3.11 to prune the search space beforehand also has a noticeable impact in a practical setting.[31]

## 3.5 Summary and Outlook

In this chapter we have discussed automation of the search for weakly monotone algebras based on polynomial interpretations with negative constants. The basis of this encoding are the polynomial approximations of Definition 3.5 introduced by [HM07], In Definition 3.8, we have presented an SMT-NIA encoding for these polynomial approximations (and proved its correctness in Theorem 3.9) which then allows an SMT solver such as our award-winning SAT-based approach from [FGM+07] to find a suitable inter-

[31] For range 3, the example `TRS/Transformed_CSR_04/PEANO_complete_noand_iGM.xml` was proved terminating by AProVE-SAT without this criterion within 56.3 seconds, whereas the termination proof using AProVE-SAT + Theorem 3.11 took 67.8 seconds, leading to a timeout.

pretation. This way, for the first time we provide a systematic search procedure for this class of interpretations. Our experimental results indicate performance improvements by orders of magnitude over the state of the art. Additionally, in Theorem 3.11 we present a novel necessary criterion for negative constants in the setting of [HM07], which is helpful for restricting the search space for parametric interpretations. Also here, our experiments indicate additional notable performance improvements.

Note that using an SMT-NIA encoding instead of a (more low-level) SAT encoding allows for additional flexibility. Currently, SAT encodings like our work from [FGM+07] (cf. also [EWZ08]) are a state-of-the-art technique for solving SMT-NIA problems. However, with the rapid progress in SMT solving, it is quite likely that over the next years, alternative solving techniques for SMT-NIA are developed which do not use this particular SAT encoding. Indeed, for instance the paper [BLO+12] presents an encoding from SMT-NIA to SMT-LIA which allows to use off-the-shelf SMT-LIA solvers to tackle SMT-NIA problems. By giving an encoding to SMT-NIA instead of a classical SAT encoding for propositional logic, we thus also make the search for polynomial interpretations with a negative constant accessible to SMT-LIA solvers.

As future work, it would be interesting to extend our parametric approach to encompass also the—in theory more powerful—criterion of AProVE 1.2 for comparing terms using a given polynomial interpretation with negative constants (cf. Footnote 30).

Moreover, one could implement an approach without approximations, which could be based on a finite case analysis. For instance, for an (implicitly universally quantified) constraint $\varphi(x, y) = \max(2\,x + y - 2, 0) \geq q(x, y)$ for some polynomial $q$, we can equivalently write $\varphi(0, 0) \wedge \varphi(0, 1) \wedge \varphi(x_1 + 1, y_1) \wedge \varphi(x_2, y_2 + 2)$ (i.e., we enumerate the finitely many cases where $2\,x + y - 2 \neq \max(2\,x + y - 2, 0)$. Partial evaluation then yields the constraint $0 \geq q(0, 0) \wedge 0 \geq q(0, 1) \wedge 2\,x_1 + y_1 \geq q(x_1 + 1, y_1) \wedge 2\,x_2 + y_2 \geq q(x_2, y_2 + 2)$, which does not contain max anymore and can be evaluated using standard techniques. Of course, this reasoning would still need to be lifted to the parametric setting. We conjecture that the approach using Definition 3.8 still outperforms the one just sketched, but it would nevertheless be interesting to investigate how much power is lost by the approximations of Definition 3.5 compared to an approach that does not use incomplete approximations.

While in the present chapter, we have also used the max-operator, this has only been the case in the special setting $\max(\cdot, 0)$. In Chapter 4 we investigate a general integration of the max-operator into polynomial interpretations for weakly monotone algebras that can be used for automated termination analysis.

Moreover, in related work Koprowski and Waldmann propose *Arctic Interpretations below Zero* [KW08, KW09], where the max-operator is applied in the context of max-plus-algebras. This is also a class of interpretations with negative numbers as ingredients that can be used in weakly monotone algebras. We discuss this setting and satisfiability-based means of automation further in Chapter 5.

In their IsaFoR [TS09] formalization of the technique of this chapter in the proof assistant Isabelle [NPW02] for certification, Sternagel and Thiemann [ST10] extend the theoretical setting to carriers like the non-negative rational or real numbers and also to *matrix interpretations* [EWZ08] where some parts of the constant vector may be negative. Our contributions to automation given in this chapter should carry over directly to the extensions proposed by Sternagel and Thiemann.

## Acknowledgments

We thank Daniel Le Berre for helpful comments on an earlier version of this chapter.

# 4 Maximal Termination

As the previous chapter shows, using the max-operator in combination with polynomial interpretations can lead to notable increases in power for termination analysis of TRSs via weakly monotone algebras.

In a related application, Marion and Péchoux [MP09] propose a combination of polynomials with the max-operator for use as *sup-interpretations*. Such interpretations can be applied to yield upper bounds for the worst-case *size complexity* of the functions defined by TRSs. Here size complexity denotes the size of the output of a function with respect to the size of its input. However, they do not discuss automation.

Also in applications such a termination analysis of heap-based imperative programming languages like Java Bytecode, automated tools like COSTA [AAC+08] and Julia [SMP10] rely on abstractions of heap structures to their *path length* [AAG+07, SMP10]. This essentially means that for an object which has pointers to several other objects on the heap, the *maximum* of the abstractions of these other objects to natural numbers is used as an ingredient for the abstraction (i.e., the interpretation) of the current object. This way, one can interpret an object as the maximum number of consecutive dereferencing steps possible from this object.

These observations and applications of the max-operator in analysis of TRSs and, more generally, programs motivate investigation of a more flexible integration of the max-operator with polynomial interpretations for termination analysis of TRSs. Here we permit arbitrary combinations of polynomials and the max-operator (i.e., max-*polynomials*), e.g., we could use $p + \max(q, \max(r, s))$ where $p, q, r, s$ are polynomials.

In this chapter we set up the class of weakly monotone algebras with max-polynomial interpretations in Section 4.1. In Section 4.2 we discuss means of automation of max-polynomials via encodings to SMT-NIA, which can then be further reduced to SAT or to SMT-LIA, as in the previous chapters. We also discuss a possible application of the setting of this chapter to the polynomial interpretations with a negative constant, as discussed in Chapter 3. Section 4.3 motivates the need for suitable heuristics for the *shape* of a max-polynomial interpretation. For this, we give a family of terms showing that $n$ occurrences of the max-operator in an interpretation of a term can lead to $2^n$ symbolic constraints. To render max-polynomial interpretations applicable in practice, we present heuristics for *shapes* of parametric max-polynomial interpretations which are based on the structure of the rewrite rules that are supposed to be oriented. We also

provide an alternative way of dealing with the max-operator, which can be helpful in some cases. In Section 4.4 we present experimental results highlighting the impact of our contribution, and in Section 4.5 we conclude.

## 4.1 Max-Polynomial Interpretations

**Example 4.1.** Consider the TRS SUBST from [HL86] and [Zan03, Ex. 6.5.42], which is also available in the TPDB as `TRS/Zantema_05/z10.xml`:

$$\lambda(x) \circ y \to \lambda(x \circ (1 \star (y \circ \uparrow))) \tag{4.1}$$
$$(x \star y) \circ z \to (x \circ z) \star (y \circ z) \tag{4.2}$$
$$(x \circ y) \circ z \to x \circ (y \circ z) \tag{4.3}$$
$$\mathsf{id} \circ x \to x \tag{4.4}$$
$$1 \circ \mathsf{id} \to 1 \tag{4.5}$$
$$\uparrow \circ \mathsf{id} \to \uparrow \tag{4.6}$$
$$1 \circ (x \star y) \to x \tag{4.7}$$
$$\uparrow \circ (x \star y) \to y \tag{4.8}$$

The dependency pairs for this TRS are:

$$\lambda(x) \circ^\sharp y \to x \circ^\sharp (1 \star (y \circ \uparrow)) \tag{4.9}$$
$$\lambda(x) \circ^\sharp y \to y \circ^\sharp \uparrow \tag{4.10}$$
$$(x \star y) \circ^\sharp z \to x \circ^\sharp z \tag{4.11}$$
$$(x \star y) \circ^\sharp z \to y \circ^\sharp z \tag{4.12}$$
$$(x \circ y) \circ^\sharp z \to x \circ^\sharp (y \circ z) \tag{4.13}$$
$$(x \circ y) \circ^\sharp z \to y \circ^\sharp z \tag{4.14}$$

**Example 4.2.** For the DP problem corresponding to Example 4.1, we use the reduction pair $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$ based on the weakly monotone algebra $(\mathcal{A}, \geq, >)$ with carrier $\mathbb{N}$ and with $[\cdot]_{\mathcal{A}}$ as follows:[32]

$$[\lambda]_{\mathcal{A}} = x_1 + 1 \qquad [\star]_{\mathcal{A}} = \max(x_1, x_2)$$
$$[\circ]_{\mathcal{A}} = [\circ^\sharp]_{\mathcal{A}} = x_1 + x_2 \qquad [1]_{\mathcal{A}} = [\mathsf{id}]_{\mathcal{A}} = [\uparrow]_{\mathcal{A}} = 0$$

Then all (usable) rules and dependency pairs are weakly decreasing (w.r.t. $\succsim_{\mathcal{A}}$). Furthermore, the DPs (4.9) and (4.10) are strictly decreasing (w.r.t. $\succ_{\mathcal{A}}$) and can be removed by

[32] Since not only the operations "+" and "∗", but also "max" are weakly monotonic on $\mathbb{N}$, the algebras considered in this chapter are all weakly monotone.

Theorem 2.13. Afterwards, we use the following interpretation $[\cdot]_{\mathcal{B}}$ where the remaining DPs are strictly decreasing and the rules are still weakly decreasing:

$$[\circ^\sharp]_{\mathcal{B}} = x_1 \qquad [\star]_{\mathcal{B}} = \max(x_1, x_2) + 1$$
$$[\circ]_{\mathcal{B}} = x_1 + x_2 + 1 \qquad [\lambda]_{\mathcal{B}} = [\mathsf{1}]_{\mathcal{B}} = [\mathsf{id}]_{\mathcal{B}} = [\uparrow]_{\mathcal{B}} = 0$$

Note that including the function "max" as a building block for interpretations along with "+", "∗", numbers from $\mathbb{N}$, and variables still leads to a weakly monotone algebra, such that for this example the reduction pair processor is indeed applicable with the given interpretations.

Note that termination of SUBST cannot be proved with Theorem 2.13 using reduction pairs based on linear polynomial interpretations. Thus, this example shows the usefulness of polynomial interpretations with "max". Previously, only restricted forms of such interpretations were available in termination tools. For example, already in 2004, T$_\text{T}$T used (non-monotonic) interpretations like $\max(x_1 - x_2, 0)$ (cf. [HM07]), but no tool offered arbitrary interpretations with polynomials and "max" like $\max(x_1, x_2) + 1$.

While SUBST's original termination proof was very complicated [HL86], easier proofs were developed later, using the techniques of *distribution elimination* [Zan94] or *semantic labeling* [Zan03]. Indeed, the only tool that could prove termination of SUBST automatically prior to this research (TPA [Kop06]) used semantic labeling as part of its proof.[33] In contrast, Example 4.2 now shows that there is an even simpler proof without semantic labeling.

## 4.2 SMT-Based Automation

The most efficient implementations to search for polynomial interpretations are based on encodings to SMT-NIA instances [CMTU05], which for solving then usually are translated further to SAT or SMT-LIA instances [FGM$^+$07, EWZ08, BLO$^+$12]. In Chapter 3 we have presented an extension of this approach to interpretations of the form $\max(p - n, 0)$ where $p$ is a polynomial with natural coefficients and $n \in \mathbb{N}$. Thus, in Chapter 3 we permit interpretations like $\max(x_1 - 1, 0)$, but not interpretations like $\max(x_1, x_2)$ (as needed in Example 4.1). In contrast, here our goal is to be able to search for *arbitrary* interpretations using polynomials and the max-operator using SMT-NIA solvers.[34] Compared to the

[33] For the semantic labeling, TPA uses only a (small) fixed set of functions, including certain fixed polynomials and the function "max". So in contrast to our automation in Section 4.2, TPA does not use more general parametric combinations of polynomials and "max". Moreover, TPA employs a labeling over an infinite carrier which leads to a TRS with infinitely many rules. This poses additional difficulties for the integration of the method into existing termination tools designed to analyze problems with only finite explicit representations.

[34] Of course, in an analogous way, one can also integrate the *minimum* function and indeed, we did this in our implementations. We present more details on using the minimum function in Section 4.3.

approach presented in Chapter 3, this poses an additional challenge since it is not clear how one could adapt, e.g., the polynomial approximations presented in Definition 3.5 to *arbitrary* combinations of "max" with polynomials without losing too much power.

**Definition 4.3** (max-polynomial)**.** *Let $\mathcal{V}$ be the set of variables. The set of* max-polynomials $\mathbb{P}_M$ *over a set of numbers $M$ is the smallest set such that*

- $M \subseteq \mathbb{P}_M$ *and* $\mathcal{V} \subseteq \mathbb{P}_M$
- *if* $p, q \in \mathbb{P}_M$*, then* $p + q \in \mathbb{P}_M$*,* $p * q \in \mathbb{P}_M$*, and* $\max(p, q) \in \mathbb{P}_M$

Concretely, we focus on the case where $M = \mathbb{N}$. So here we map every function symbol to a max-polynomial over $\mathbb{P}_\mathbb{N}$. Obviously, then $(\succsim_\mathcal{A}, \succ_\mathcal{A})$ is a $\mathcal{C}_\varepsilon$-compatible reduction pair (since also max is a weakly monotonic function on $\mathbb{N}$).

As in the previous chapters, to find such interpretations automatically we start with a *parametric* max-polynomial interpretation. It maps each function symbol to a max-polynomial over a set $\mathcal{A}$ of *parametric* coefficients. In other words, we have to determine the degree and the shape of the max-polynomial in advance of the search, but the actual coefficients are left open. For example, for the TRS of Example 4.1 we could use a parametric polynomial interpretation $[\cdot]_\mathcal{A}$ where $[\star]_\mathcal{A} = \max(a_1\, x_1 + a_2\, x_2,\ a'_1\, x_1 + a'_2\, x_2)$, $[\uparrow]_\mathcal{A} = b$, $[\circ]_\mathcal{A} = x_1 + x_2$, etc.[35] Here, $a_1, a_2, a'_1, a'_2, b$ are parametric coefficients.

Now to apply the reduction pair processor of Theorem 2.13, we have to find an instantiation of the parametric coefficients satisfying the following condition. Then all dependency pairs that are strictly decreasing (i.e., $[s] \geq [t] + 1$, which on $\mathbb{N}$ is equivalent to $[s] > [t]$) can be removed (cf. (3.14)).

$$\bigwedge_{\ell \to r \,\in\, \mathcal{P} \cup \mathcal{U}_\mathcal{R}(\mathcal{P})} [\ell]_\mathcal{A} \geq [r]_\mathcal{A} \ \wedge \bigvee_{s \to t \,\in\, \mathcal{P}} [\ell]_\mathcal{A} \geq [r]_\mathcal{A} + 1 \tag{4.15}$$

Here, all rules in $\mathcal{P} \cup \mathcal{U}_\mathcal{R}(\mathcal{P})$ are variable-renamed to have pairwise different variables. The expressions $[s]_\mathcal{A}$, $[t]_\mathcal{A}$, etc. are again max-polynomials over $\mathcal{A}$. So with the interpretation $[\cdot]_\mathcal{A}$ above, to make the last rule of Example 4.1 weakly decreasing (i.e., $\uparrow \circ (x \star y) \succsim_\mathcal{A} y$) we obtain the inequality $[\uparrow \circ (x \star y)]_\mathcal{A} \geq [y]_\mathcal{A}$:

$$b + \max(a_1\, x + a_2\, y,\ a'_1\, x + a'_2\, y) \geq y \tag{4.16}$$

We have to find an instantiation of the parametric coefficients $a_1, a_2, \ldots$ such that (4.16) holds for *all* instantiations of the variables $x$ and $y$. In other words, the variables from $\mathcal{V}$ occurring in such inequalities are again universally quantified.

[35] Here $\circ$'s interpretation is fixed beforehand to simplify the presentation. Our implementations use heuristics to determine when to use an interpretation with "max", cf. Section 4.3.

## Transformation Rules

Several techniques have been proposed to transform such inequalities further in order to remove such universally quantified variables [HJ98] (cf. Theorem 2.18). However, the existing techniques only operate on inequalities without "max". Therefore, we now present new inference rules to eliminate "max" from such inequalities.

Our inference rules operate on *conditional* constraints of the form

$$p_1 \geq q_1 \wedge \ldots \wedge p_n \geq q_n \;\Rightarrow\; p \geq q \tag{4.17}$$

Here, $n \geq 0$ and $p_1, \ldots, p_n, q_1, \ldots, q_n$ are polynomials with parametric coefficients without "max". In contrast, $p$, $q$ are max-polynomials with parametric coefficients.

The first inference rule eliminates an inner occurrence of "max" from the inequality $p \geq q$. If $p$ or $q$ have a sub-expression $\max(p', q')$ where $p'$ and $q'$ do not contain "max", then we can replace this sub-expression by $p'$ or $q'$ when adding the appropriate condition $p' \geq q'$ or $q' \geq p' + 1$, respectively.

I. **Eliminating "max"**

$$\begin{array}{llllll} \varphi & \Rightarrow & \ldots \max(p', q') \ldots & & & \\ \hline \varphi \wedge p' \geq q' & \Rightarrow & \ldots & p' & \ldots & \wedge \\ \varphi \wedge q' \geq p' + 1 & \Rightarrow & \ldots & q' & \ldots & \end{array}$$

if $p'$ and $q'$ do not contain "max" and where $\varphi = p_1 \geq q_1 \wedge \ldots \wedge p_n \geq q_n$

Obviously, by repeated application of inference rule (I), all occurrences of "max" can be removed. In our example, the constraint (4.16) is transformed into the following new constraint that does not contain "max" anymore.

$$a_1\, x + a_2\, y \geq a'_1\, x + a'_2\, y \;\Rightarrow\; b + a_1\, x + a_2\, y \geq y \qquad \wedge \tag{4.18}$$

$$a'_1\, x + a'_2\, y \geq a_1\, x + a_2\, y + 1 \;\Rightarrow\; b + a'_1\, x + a'_2\, y \geq y \tag{4.19}$$

Since the existing methods for eliminating universally quantified variables only work for *unconditional* inequalities, the next inference rule eliminates the conditions $p_i \geq q_i$ from a constraint of the form (4.17).[36] To this end, we introduce two new parametric polynomials $\overline{p}$ and $\overline{q}$ (that do not contain "max"). The polynomial $\overline{q}$ over the variables $x_1, \ldots, x_n$ is used to "measure" the polynomials $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$, respectively, in the premise of (4.17). Similarly, the unary polynomial $\overline{p}$ over $x_1$ measures the polynomials $p$ and $q$ in the conclusion of (4.17). We write $\overline{q}(p_1, \ldots, p_n)$ to denote the result of instantiating the variables $x_1, \ldots, x_n$ in $\overline{q}$ by $p_1, \ldots, p_n$, etc.

[36] Such conditional polynomial constraints also occur in other applications, such as the termination analysis of logic programs. Indeed, [NDGS11] uses a rule similar to inference rule (II) in the tool Polytool for termination analysis of logic programs. However, Polytool only applies classical polynomial interpretations without "max".

| II. **Eliminating Conditions** | |
|---|---|
| $\dfrac{p_1 \geq q_1 \wedge \ldots \wedge p_n \geq q_n \;\Rightarrow\; p \geq q}{\overline{p}(p) - \overline{p}(q) \geq \overline{q}(p_1, \ldots, p_n) - \overline{q}(q_1, \ldots, q_n)}$ | if $\overline{q}$ and $\overline{p}$ do not contain "max", $\overline{p}$ is strongly monotonic, and $\overline{q}$ is weakly monotonic |

Here, the monotonicity conditions mean that $x > y \Rightarrow \overline{p}(x) > \overline{p}(y)$ must hold and similarly that $x_1 \geq y_1 \wedge \ldots \wedge x_n \geq y_n \Rightarrow \overline{q}(x_1, \ldots, x_n) \geq \overline{q}(y_1, \ldots, y_n)$.

To see why Rule (II) is sound, let $\overline{p}(p) - \overline{p}(q) \geq \overline{q}(p_1, \ldots, p_n) - \overline{q}(q_1, \ldots, q_n)$ hold and assume that there is an instantiation $\sigma$ of all variables in the polynomials with numbers that refutes $p_1 \geq q_1 \wedge \ldots \wedge p_n \geq q_n \;\Rightarrow\; p \geq q$. Now $p_1\sigma \geq q_1\sigma \wedge \ldots \wedge p_n\sigma \geq q_n\sigma$ implies $\overline{q}(p_1, \ldots, p_n)\sigma \geq \overline{q}(q_1, \ldots, q_n)\sigma$ by weak monotonicity of $\overline{q}$. Hence, $\overline{p}(p)\sigma - \overline{p}(q)\sigma \geq 0$. Since the instantiation $\sigma$ is a counterexample to our original constraint, we have $p\sigma \not\geq q\sigma$ and thus $p\sigma < q\sigma$. But then strict monotonicity of $\overline{p}$ would imply $\overline{p}(p)\sigma - \overline{p}(q)\sigma < 0$, which gives a contradiction.

If we choose[37] the parametric polynomials $\overline{p} = c\,x_1$ and $\overline{q} = d\,x_1$ for (4.18) and $\overline{p} = c'\,x_1$ and $\overline{q} = d'\,x_1$ for (4.19), then (4.18) and (4.19) are transformed into the following unconditional inequalities. (Note that we also have to add the inequalities $c \geq 1$ and $c' \geq 1$ to ensure that $\overline{p}$ is strongly monotonic.)

$$c \cdot (b + a_1\,x + a_2\,y) - c \cdot y \geq d \cdot (a_1\,x + a_2\,y) - d \cdot (a'_1\,x + a'_2\,y) \qquad \wedge \qquad (4.20)$$

$$c' \cdot (b + a'_1\,x + a'_2\,y) - c' \cdot y \geq d' \cdot (a'_1\,x + a'_2\,y) - d' \cdot (a_1\,x + a_2\,y + 1) \qquad (4.21)$$

Of course, such inequalities can be transformed into inequalities with 0 on their right-hand side. For example, (4.20) is transformed to

$$(c\,a_1 - d\,a_1 + d\,a'_1)\;x \;+\; (c\,a_2 - c - d\,a_2 + d\,a'_2)\;y \;+\; c\,b \;\geq\; 0 \qquad (4.22)$$

Thus, we again have to ensure non-negativeness of "polynomials" over variables like $x, y$, where the "coefficients" are polynomials over the parametric variables like $c\,a_1 - d\,a_1 + d\,a'_1$. For this purpose, we make use of the absolute positiveness criterion presented in Theorem 2.18 (cf. [HJ98]) and require that all these "coefficients" are $\geq 0$. This way, we can again eliminate all universally quantified variables like $x, y$, and (4.22) is transformed into the Diophantine constraint

$$c\,a_1 - d\,a_1 + d\,a'_1 \geq 0 \qquad \wedge \qquad c\,a_2 - c - d\,a_2 + d\,a'_2 \geq 0 \qquad \wedge \qquad c\,b \geq 0$$

Formulated as an inference rule for constraints on max-polynomials, the absolute positiveness criterion is expressed as follows:

[37] In this setting a good heuristic is to choose $\overline{q} = b_1x_1 + \ldots + b_nx_n$ where all $b_i$ are from $\{0, 1\}$ and $\overline{p} = a \cdot x_1$ where $1 \leq a \leq \max(\Sigma_{i=1}^{n} b_i,\ 1)$.

| III. **Eliminating Universally Quantified Variables** | |
|---|---|
| $p_0 + p_1\, x_1^{e_{11}} \dots x_n^{e_{n1}} + \dots + p_k\, x_1^{e_{1k}} \dots x_n^{e_{nk}} \geq 0$ | if the $p_i$ neither contain "max" |
| $p_0 \geq 0 \;\wedge\; p_1 \geq 0 \;\wedge \dots \wedge\; p_k \geq 0$ | nor any variable from $\mathcal{V}$ |

To search for suitable values for the parametric coefficients that satisfy the resulting Diophantine constraints, in practice we fix an upper bound for these values.

Moreover, for soundness we need to make sure that the parametric coefficients $a$ are instantiated by natural numbers only. Therefore, for all occurring parametric coefficients $a$ we also need to provide the information $a \geq 0$ to the SMT-NIA solver. In our example, the constraints resulting from the initial inequality (4.16) are satisfied, e.g., by $a_1 = 1$, $a_2 = 0$, $a'_1 = 0$, $a'_2 = 1$, $b = 0$, $c = 1$, $d = 1$, $c' = 1$, $d' = 0$.[38] With these values, the parametric interpretation $\max(a_1\, x_1 + a_2\, x_2,\; a'_1\, x_1 + a'_2\, x_2)$ for $\star$ is turned into the concrete interpretation $\max(x_1, x_2)$.

### Negative Constants

Actually, we can be even more general with the shapes of max-polynomials used for interpretations than in Definition 4.3. For instance, we can also allow max-polynomial interpretations to use *negative constants*, as long as the overall interpretation of a function symbol is always non-negative. This way, we can also use max-polynomial interpretations to implement polynomial interpretations with a negative constant as in Chapter 3 and possibly render the method more efficient or more powerful than the automation of the approximations of Definition 3.5 via Definition 3.8 described in Chapter 3. As our experiments show, however, this approach has severe performance issues in practice (cf. Section 4.4). Therefore, we recommend the dedicated approach from Chapter 3 for this special setting.

## 4.3 Shape Heuristics and Optimizations

### Exponential Blowup

Max-polynomials conveniently provide further expressive power over standard polynomials. However, as the following example shows, too unrestricted occurrences of "max" in an interpretation may cause severe problems in the initial phase of their automation.

**Example 4.4** (Exponential Blowup for Handling "max" Automatically)**.** Consider the following family of terms $T = \{t_i \mid i \in \mathbb{N}\}$ over a signature $\mathcal{F}$, where $\mathsf{a} \in \mathcal{F}$ and $ar(\mathsf{a}) = 2$:

- $t_0 = y$,
- $t_{n+1} = \mathsf{a}(x_{n+1}, t_n)$.

[38] To ease readability, we write $a_1 = 1$ instead of $\mathcal{D}(a_1) = 1$, etc.

Note that the term $t_n$ has size $2n+1$. Consider moreover a max-polynomial interpretation with $[\mathsf{a}](x_1, x_2) = \max(x_1, x_2)$. A term constraint $t_n \gtrsim y$ then becomes $[t_n] \geq [y]$, i.e., $\max(x_n, \max(x_{n-1}, \ldots, \max(x_1, y)\ldots)) \geq y$. Then repeated application of inference rule (I) (possibly in combination with (II)) from Section 4.2 to eliminate all occurrences of max from $[t_n]$ yields $2^n$ conditional constraints. (This follows with a direct inductive argument using $[t_{n+1}] = \max(x_{n+1}, [t_n])$.)

Although this example is rather artificial, it illustrates that an unrestricted application of even very simple shapes of max-polynomial interpretations can lead to an exponential blowup already on symbolical level, before the SMT solver is even invoked.

Moreover, while also the underlying term shape of Example 4.4 may look very artificial, in fact it is rather common to have TRSs over a signature that only contains constants and a single binary function symbol. This class of TRSs is also known as *applicative TRSs (ATRSs)*. For instance, in applicative notation, the TRS rule (2.6), i.e., $\mathsf{bits}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{bits}(\mathsf{s}(\mathsf{half}(x))))$, would be represented as $\mathsf{a}(\mathsf{bits}, \mathsf{a}(\mathsf{s}, \mathsf{a}(\mathsf{s}, x))) \to \mathsf{a}(\mathsf{s}, \mathsf{a}(\mathsf{bits}, \mathsf{a}(\mathsf{s}, \mathsf{a}(\mathsf{half}, x))))$. ATRSs are frequently used to express programs with higher-order functions via first-order TRSs, and the current TPDB (version 8.0.2) contains almost 200 ATRSs.

## Heuristics for Interpretation Shapes

Another burden that comes with the increased flexibility of max-polynomials is that it is less clear which shape to use for the parametric interpretations than with classical polynomial interpretations. For instance, even if one only wishes to use an interpretation like $\sum a_{i_k,j_k} \max(x_{i_k}, x_{j_k})$ for some $i_k$ and $j_k$, it is a priori not clear *which* $i_k$ and $j_k$ are suitable candidates. For an $n$-ary function symbol $f$, we already obtain $\frac{n^2-n}{2}$ possible expressions $\max(x_i, x_j)$ (here we already take into account that we can safely require $i < j$ since $\max(x, y) = \max(y, x)$ and $\max(x, x) = x$).

Due to these issues, one should use suitable *heuristics* to determine for *which* function symbols of a set of term constraints and in what way one should use "max" (and, analogously, "min").

A common theme for the following heuristics is that for function symbols $f$ with $ar(f) = n$, we always regard parametric interpretations of the shape

$$\begin{aligned} &[f](x_1, \ldots, x_n) \\ &= f_0 + f_1\, x_1 + \ldots + f_n\, x_n + \sum a_{i_k,j_k}\, \max(x_{i_k}, x_{j_k}) + \sum b_{i'_\ell,j'_\ell}\, \min(x_{i'_\ell}, x_{j'_\ell}) \end{aligned} \tag{4.23}$$

Here all $f_i, a_{i,j}, b_{i,j}$ are parametric coefficients that are only used in the interpretation of the symbol $f$. In other words, we use linear polynomial interpretations where we additionally allow applications of the maximum (or minimum) function to two variables

as addends (multiplied with a parametric coefficient).

The following heuristic detects cases where a rule *duplicates* a variable below a binary constructor symbol:

**Heuristic 4.5** (Duplication Heuristic)**.** *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem, let $c$ be a binary constructor symbol. If $\mathcal{P} \cup \mathcal{R}$ contains a rule[39] $\ell \to C[c(t_1, t_2)]$ for some context $C$ where $\mathcal{V}ar(t_1) \cap \mathcal{V}ar(t_2) \neq \varnothing$, then include $a_{1,2} \cdot \max(x_1, x_2)$ as an addend for the parametric max-polynomial $[c]$.*

The intuition for this heuristic is that interpretations like $[\mathsf{c}](x_1, x_2) = \max(x_1, x_2)$ allow to solve a term constraint like $x \gtrsim \mathsf{c}(x, x)$ since the resulting max-polynomial constraint $x \geq \max(x, x)$ holds over $\mathbb{N}$. At the same time, we do not need to *filter* an argument $i$ of $\mathsf{c}$ by the interpretation (i.e., use an interpretation that is *independent* of $x_i$). This can become relevant for solving other term constraints at the same time.[40]

As an example for the applicability of Heuristic 4.5 consider again the TRS SUBST from Example 4.1 and, specifically, its rule:

$$(x \star y) \circ z \to (x \circ z) \star (y \circ z) \tag{4.2}$$

Zantema [Zan94] identifies rule (4.2) as a *distribution rule* for $\star$, i.e., the rule has the shape $C[a(x_1, \ldots, x_n)] \to a(C[x_1], \ldots, C[x_n])$ for a non-trivial context $C$ where $\star$ corresponds to the symbol $a$. As part of his termination proof, Zantema uses a specialized technique for distribution rules called *distribution elimination* [Zan94] to remove the distribution rule for $\star$ from the TRS. Also in our proof, the symbol $\star$ gets a special treatment, i.e., it is the only symbol where we use the max-operator as a building block for its interpretation. Indeed, Heuristic 4.5 recognizes the symbol $\star$ as a candidate for using $\max(x_1, x_2)$ as part of its interpretation because of rule (4.2).

The next heuristic is based on identifying *projection rules*:

**Heuristic 4.6** (Variable Projection Heuristic)**.** *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem, let $f$ be a function symbol of arity $n$. If $\mathcal{P} \cup \mathcal{R}$ contains two rules $C_1[f(s_1, \ldots, s_{i-1}, y, s_{i+1}, \ldots, s_n)] \to y$ and $C_2[f(t_1, \ldots, t_{j-1}, z, t_{j+1}, \ldots, t_n)] \to z$ for variables $y$ and $z$ where $i < j$, then include $a_{i,j} \cdot \max(x_i, x_j)$ as an addend for the parametric max-polynomial $[f]$.*

Here the intuition is that if a function symbol $f$ can be projected to several different arguments $x_i$ and $x_j$, then it could be beneficial to satisfy both

- $f(s_1, \ldots, s_{i-1}, y, s_{i+1}, \ldots, s_n) \gtrsim y$ and

[39] Of course, one could formulate the heuristics analogously for sets of *term constraints* instead of TRSs.

[40] In [FNO+08, Heuristic 19] we present a similar heuristic for applying polynomial interpretations with non-negative *rational* coefficients like $\frac{1}{2}$. Also here, the intuition is that interpretations like $[\mathsf{c}](x_1, x_2) = \frac{1}{2}\,x_1 + \frac{1}{2}\,x_2$ allow to solve a term constraint $x \gtrsim \mathsf{c}(x, x)$. With this interpretation we obtain $x \geq \frac{1}{2}\,x + \frac{1}{2}\,x$, which holds over the non-negative rational numbers. Again, this interpretation does not filter any argument of $\mathsf{c}$.

- $f(t_1, \ldots, t_{j-1}, z, t_{j+1}, \ldots, t_n) \succsim z$

using an interpretation $[f]$ that is as small as possible. The smallest such max-polynomial $p$ with $p \geq x_i$ and $p \geq x_j$ is $p = \max(x_i, x_j)$.

There are several typical functions defined by rewrite rules where this pattern is very helpful. For instance, the function $\mathsf{max}$ which computes the maximum of two natural numbers (in term notation) is usually defined via the following rules:

$$\mathsf{max}(x, \mathsf{0}) \to x \tag{4.24}$$

$$\mathsf{max}(\mathsf{0}, y) \to y \tag{4.25}$$

$$\mathsf{max}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{s}(\mathsf{max}(x, y)) \tag{4.26}$$

For typical uses of symbols like $\mathsf{0}$ and $\mathsf{s}$ for modeling natural numbers in TRSs, often the interpretations $[\mathsf{0}] = 0$ and $[\mathsf{s}](x_1) = x_1 + 1$ turn out to be beneficial for the termination proof. In this case, an interpretation $[\mathsf{max}](x_1, x_2) = \max(x_1, x_2)$ indeed is the smallest possible interpretation such that $\mathsf{max}(x, \mathsf{0}) \succsim x$ and $\mathsf{max}(\mathsf{0}, y) \succsim y$ hold. Due to the rules (4.24) and (4.25), Heuristic 4.6 would recognize that $\mathsf{max}$ is a suitable candidate for $\max(x_1, x_2)$ as an ingredient of its interpretation.

Similarly, one often observes a function symbol $\mathsf{if}$ to model conditional expressions in TRSs. The function can be defined, e.g., by the following two rules:

$$\mathsf{if}(\mathsf{true}, x, y) \to x \tag{4.27}$$

$$\mathsf{if}(\mathsf{false}, x, y) \to y \tag{4.28}$$

Also here, an interpretation $[\mathsf{if}](x_1, x_2, x_3) = \max(x_2, x_3)$ is frequently useful. For instance, in [AEF+08] we prove termination of the leading example [AEF+08, Example 1], which contains the rules (4.27) and (4.28), using such interpretations for $\mathsf{if}$.[41]

As mentioned earlier, our implementation also offers the possibility to use "min" in a way which is completely analogous to "max". Also here, it is sensible to use a dedicated heuristic to select in what way one should include "min" in a parametric interpretation. For this purpose, we propose the following criterion.

**Heuristic 4.7** (Ground Projection Heuristic)**.** *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem, let $f$ be a function symbol of arity $n$. If $\mathcal{P} \cup \mathcal{R}$ contains rules $C_1[f(s_1, \ldots, s_{i-1}, r_1, s_{i+1}, \ldots, s_n)] \to r_1$ and $C_2[f(t_1, \ldots, t_{j-1}, r_2, t_{j+1}, \ldots, t_n)] \to r_2$ for constructor ground terms $r_1, r_2$ where $i < j$ and if each term $s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n, t_1, \ldots, t_{j-1}, t_{j+1}, \ldots, t_n$ contains variables, then include $b_{i,j} \cdot \min(x_i, x_j)$ as an addend for the parametric max-polynomial $[f]$.*

[41] Our paper [AEF+08] deals with proving termination of *context-sensitive* term rewriting [Luc98, Luc02]. This strategy restriction of term rewriting can conveniently be used to model the usual behavior of $\mathsf{if}(t_1, t_2, t_3)$: one may evaluate $t_1$, but not $t_2$ or $t_3$.

While in practice less frequently needed for termination proofs than "max", also "min" can be beneficial as an ingredient for max-polynomial interpretations. Heuristic 4.7 detects cases where a function symbol projects to a *ground* argument at different argument positions. This is the case for the usual implementation of the function $\mathsf{min}$, which computes the minimum of two terms representing natural numbers:

$$\mathsf{min}(x, \mathsf{0}) \to \mathsf{0} \tag{4.29}$$

$$\mathsf{min}(\mathsf{0}, y) \to \mathsf{0} \tag{4.30}$$

$$\mathsf{min}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{s}(\mathsf{min}(x, y)) \tag{4.31}$$

Heuristic 4.7 uses the rules (4.29) and (4.30) to detect that $\mathsf{min}$ might benefit from $\min(x_1, x_2)$ as a building block of its interpretation.

In addition to using Heuristic 4.5, Heuristic 4.6, and Heuristic 4.7, in AProVE we only allow "max" or "min" as part of our interpretation if we are not working on an ATRS (or a DP problem with a corresponding signature). Moreover, we delete conditional constraints where we can show that the premise is inconsistent or that it entails the conclusion for all possible values of the parametric coefficients. Likewise, we also drop valid atoms from constraint premises.

To see the effect of the heuristics in practice, consider the following example:

**Example 4.8.** The TRS `TRS/Secret_06_TRS/tpa06.xml` from the TPDB consists of the above rules (4.24), (4.25), (4.26), (4.29), (4.30), (4.31) for $\mathsf{max}$ and $\mathsf{min}$ and the following rules:

$$\mathsf{p}(\mathsf{s}(x)) \to x \tag{4.32}$$

$$\begin{aligned}\mathsf{f}(\mathsf{s}(x), \mathsf{s}(y), \mathsf{s}(z)) \to \mathsf{f}(\,&\mathsf{max}(\mathsf{s}(x), \mathsf{max}(\mathsf{s}(y), \mathsf{s}(z))),\ \mathsf{p}(\mathsf{min}(\mathsf{s}(x), \mathsf{max}(\mathsf{s}(y), \mathsf{s}(z)))),\\ &\mathsf{min}(\mathsf{s}(x), \mathsf{min}(\mathsf{s}(y), \mathsf{s}(z)))\,)\end{aligned} \tag{4.33}$$

$$\mathsf{f}(\mathsf{0}, y, z) \to \mathsf{max}(y, z) \tag{4.34}$$

$$\mathsf{f}(x, \mathsf{0}, z) \to \mathsf{max}(x, z) \tag{4.35}$$

$$\mathsf{f}(x, y, \mathsf{0}) \to \mathsf{max}(x, y) \tag{4.36}$$

Using max-polynomial interpretations, AProVE succeeds in finding a termination proof automatically. In the final proof step of the modular termination proof for this challenge example,[42] the reduction pair processor needs to solve the following term constraints:

[42] This termination proof involves also several applications of the *rewriting dependency pairs processor* [GTSF06, Thi07], which performs symbolic evaluation on the right-hand sides of dependency pairs. This explains why the constraint (4.33) does not correspond to any dependency pair in $\mathcal{DP}(\mathcal{R})$ where $\mathcal{R}$ is the original TRS `TRS/Secret_06_TRS/tpa06.xml`.

$$\mathsf{F}(\mathsf{s}(x),\mathsf{s}(y),\mathsf{s}(z)) \succ \mathsf{F}(\,\mathsf{s}(\mathsf{max}(x,\mathsf{max}(y,z))),\ \mathsf{min}(x,\mathsf{max}(y,z)),\ \mathsf{s}(\mathsf{min}(x,\mathsf{min}(y,z)))\,) \tag{4.37}$$

$$\mathsf{max}(x,\mathsf{0}) \succsim x \tag{4.38}$$

$$\mathsf{max}(\mathsf{0},y) \succsim y \tag{4.39}$$

$$\mathsf{max}(\mathsf{s}(x),\mathsf{s}(y)) \succsim \mathsf{s}(\mathsf{max}(x,y)) \tag{4.40}$$

$$\mathsf{min}(x,\mathsf{0}) \succsim \mathsf{0} \tag{4.41}$$

$$\mathsf{min}(\mathsf{0},y) \succsim \mathsf{0} \tag{4.42}$$

$$\mathsf{min}(\mathsf{s}(x),\mathsf{s}(y)) \succsim \mathsf{s}(\mathsf{min}(x,y)) \tag{4.43}$$

AProVE finds the following interpretation to solve these term constraints: $[\mathsf{F}](x_1,x_2,x_3) = x_2 + x_3$, $[\mathsf{max}](x_1,x_2) = \max(x_1,x_2)$, $[\mathsf{min}](x_1,x_2) = \min(x_1,x_2)$, $[\mathsf{s}](x_1) = x_1 + 1$, and $[\mathsf{0}] = 0$. Thus, AProVE can also successfully apply max-polynomial interpretations if more complicated expressions with "max" and "min" are involved.

## Containing the Blowup

In case of *nested* occurrences of the max-function in the conditional constraints, it turns out that we are not obliged to suffer the exponential blowup.

**Example 4.9** (Example 4.4 Revisited)**.** Consider again the family of terms $T$ from Example 4.4. Recall that with the interpretation $[\cdot]$ the term constraint $t_n \succsim y$ becomes $[t_n] \geq [y]$, i.e.:

$$\max(x_n, \max(x_{n-1}, \ \ldots\ , \max(x_1, y)\ldots)) \geq y$$

We can express this (conditional) max-polynomial constraint equivalently using a max-operator of *higher arity*[43] (here we use the arity $n+1$) and write:

$$\max(x_n, x_{n-1}, \ \ldots\ , x_1, y) \geq y$$

An adaption of inference rule (I) then yields only the following $n+1$ conditional constraints which do not contain "max" anymore:

$$x_1 \geq x_2 \wedge x_1 \geq x_3 \wedge \ldots \wedge x_1 \geq x_n \wedge x_1 \geq y \Rightarrow x_1 \geq y \tag{4.44}$$

$$x_2 \geq x_1 \wedge x_2 \geq x_3 \wedge \ldots \wedge x_2 \geq x_n \wedge x_2 \geq y \Rightarrow x_2 \geq y \tag{4.45}$$

[43] Following Definition 4.3, "max" can only be used with exactly 2 arguments, but one can consider $\max(p_1,p_2,\ldots,p_{n-1},p_n)$ to be an abbreviation for $\max(p_1,\max(p_2,\ldots,\max(p_{n-1},p_n)\ldots))$. Since "max" is associative (and commutative), this does not cause any problems.

$$\ldots \tag{4.46}$$

$$x_n \geq x_1 \wedge x_n \geq x_2 \wedge \ldots \wedge x_n \geq x_{n-1} \wedge x_n \geq y \Rightarrow x_n \geq y \tag{4.47}$$

$$y \geq x_1 \wedge y \geq x_2 \wedge \ldots \wedge y \geq x_n \wedge y \geq x_n \Rightarrow y \geq y \tag{4.48}$$

Here the intuition is that for finding out which value an expression $\max(p_1, \ldots, p_r)$ takes, it suffices to know which $p_i$ is the maximum element, i.e., at least as great as all other $p_j$. In contrast, we can ignore the explicit case analysis for the two cases $p_j \geq p_k$ and $p_k \geq p_j$ with $j \neq i$ and $k \neq i$ since the conclusion is in both cases the same.

The following inference rule (IV) now allows us to *merge* two occurrences of "max" where one of them is a direct argument of the other. This rule that we have also used in Example 4.9 is in principle always applicable.

IV. **Merging "max"**

$$\frac{\ldots \Rightarrow \ldots \max(p_1, \ldots, \max(q_1, \ldots, q_n), \ldots, p_r) \ldots}{\ldots \Rightarrow \ldots \max(p_1, \ldots, q_1, \ldots, q_n, \ldots, p_r) \ldots}$$

In general, the nesting of two nested occurrences of "max" in conditional constraints does not have to be *direct* (i.e., other arithmetic operators can occur between two occurrences of "max". Thus, we need the following additional inference rule (V) to normalize max-polynomials accordingly.

V. **Distributing "+" and "∗"**

$$\frac{\ldots \Rightarrow \ldots \max(p_1, \ldots, p_r) \circ \max(q_1, \ldots, q_n) \ldots}{\ldots \Rightarrow \ldots \max(p_1 \circ q_1, \ldots, p_1 \circ q_n, \ \ldots \ , p_r \circ q_1, \ldots, p_r \circ q_n) \ldots}$$

if $\circ \in \{+, *\}$ and if $p_1, \ldots, p_r$, $q_1, \ldots, q_n$ do not contain "max" and cannot become negative; here we identify $\max(p)$ and $p$

For instance, inference rule (V) allows to replace $\max(x, y) + 3$ by $\max(x + 3, y + 3)$, and it allows to replace $\max(x, 2) * \max(y, z)$ by $\max(xy, xz, 2y, 2z)$. Note that inference rule (V) requires for soundness that $p_1, \ldots, p_r, q_1, \ldots, q_n$ cannot take any negative values: Consider the max-polynomial $-1 * \max(2, 3)$. Arithmetic evaluation yields the value $-3$. If we applied inference rule (V) instead, we would obtain $\max(-1 * 2, -1 * 3)$, which would evaluate to a different result, viz. $-2$. The problem here is that "∗" is not weakly monotonic if negative numbers are considered as arguments.[44]

However, if we work on $\mathbb{P}_{\mathbb{N}}$, then $p_1, \ldots, p_r, q_1, \ldots, q_n$ can never take negative values since $\mathbb{N}$ does not contain negative numbers, and we do not allow the "−"-operator in $\mathbb{P}_M$.

Now we can introduce the inference rule (VI), which is the adaption of inference rule (I) for "max" of higher arities that we have used in Example 4.9.

[44] In contrast, for the operator "+", inference rule (V) would actually be applicable also for negative values since also in this case "+" is weakly monotonic.

VI. **Eliminating High-Arity "max"**

| $\varphi$ | $\Rightarrow$ | $\ldots \max(p_1, \ldots, p_n) \ldots$ | | | if $p_1, \ldots, p_n$ do not contain "max" |
|---|---|---|---|---|---|
| $\varphi \wedge p_1 \geq p_2 \wedge p_1 \geq p_3 \wedge \ldots \wedge p_1 \geq p_n$ | $\Rightarrow$ | $\ldots$ | $p_1$ | $\ldots$ $\wedge$ | |
| $\varphi \wedge p_2 \geq p_1 \wedge p_2 \geq p_3 \wedge \ldots \wedge p_2 \geq p_n$ | $\Rightarrow$ | $\ldots$ | $p_2$ | $\ldots$ $\wedge$ | |
| $\ldots$ | | | | | |
| $\varphi \wedge p_n \geq p_1 \wedge p_n \geq p_2 \wedge \ldots \wedge p_n \geq p_{n-1}$ | $\Rightarrow$ | $\ldots$ | $p_n$ | $\ldots$ | |

Note that inference rule (VI) is not a proper generalization of inference rule (I). Inference rule (I) transforms, e.g., $\max(x, y) \geq x$ to $(x \geq y \Rightarrow x \geq x) \wedge (y \geq x{+}1 \Rightarrow y \geq x)$ (here the premises describe disjoint cases), whereas inference rule (VI) transforms $\max(x, y) \geq x$ to $(x \geq y \Rightarrow x \geq x) \wedge (y \geq x \Rightarrow y \geq x)$ (here both premises hold for $x = y$).

Note also that the additional inference rules (IV) – (VI) in general do not solve the problem that a term constraint is transformed into exponentially many conditional constraints, as shown by the following example.

**Example 4.10** (Exponential Blowup for Handling "max", version 2)**.** Consider the following family of terms $T' = \{t'_i \mid i \in \mathbb{N}\}$ over a signature $\mathcal{F}$ where $\mathsf{a}, \mathsf{cons}, \mathsf{nil} \in \mathcal{F}$, $ar(\mathsf{a}) = ar(\mathsf{cons}) = 2$, and $ar(\mathsf{nil}) = 0$:

- $t'_0 = y$,
- $t'_{n+1} = \mathsf{cons}(\, \mathsf{a}(x_{n+1,1},\, x_{n+1,2}),\, t'_n)$.

Note that the term $t'_n$ has size $4n + 1$. Consider moreover a max-polynomial interpretation with $[\mathsf{a}](x_1, x_2) = \max(x_1, x_2)$, $[\mathsf{cons}](x_1, x_2) = x_1 + x_2$, and $[\mathsf{nil}] = 42$. A term constraint $t'_n \gtrsim \mathsf{nil}$ then becomes $[t'_n] \geq [\mathsf{nil}]$, i.e., $\max(x_{n,1}, x_{n,2}) + \max(x_{n-1,1}, x_{n-1,2}) + \ldots + \max(x_{1,1}, x_{1,2}) + y \geq 42$. Then repeated application of inference rule (I) (possibly in combination with (II)) from Section 4.2 to eliminate all occurrences of "max" from $[t'_n]$ again yields $2^n$ conditional constraints since $n$ occurrences of "max" must be eliminated. To render inference rule (VI) applicable, we need to apply rule (V) first so that "+" is moved inside "max". This, however, leads to a max-polynomial on the left-hand side where "max" has $2^n$ arguments of the shape $x_{n,i_n} + \ldots + x_{1,i_1} + y$ and all $2^n$ combinations of $i_1, \ldots, i_n \in \{1, 2\}$ occur. Then inference rule (VI) also yields $2^n$ conditional constraints without "max".

As Example 4.10 indicates, even with the new inference rules (IV) – (VI) we obtain an exponential blowup already when interpreting a single function symbols as the maximum of two of its arguments and otherwise using only linear polynomials. The problem is that for interpreting a term $t'_n$, here one needs to deal with sums of several expressions with "max" at their root. This invokes the costly inference rule (V), which distributes "+" over "max".

Thus, depending on the desired shape of the parametric interpretation, the inference rules (IV) – (VI) may be more suitable than the inference rule (I). However, in general

these inference rules solve the complexity issues of this automation only for rather specialized parametric shapes of interpretations, and in most cases one should still apply suitable heuristics to obtain parametric interpretations.

## 4.4 Experiments

Our results have been implemented in the systems AProVE [GST06] and T$_{\text{T}}$T$_2$ [KSZM09]. While AProVE and T$_{\text{T}}$T$_2$ were already the two most powerful termination provers for TRSs at the International Termination Competition in 2007 [MZ07], our contributions increase the power of both tools considerably without affecting their efficiency. More precisely, when using a time limit of 60 seconds per example, AProVE and T$_{\text{T}}$T$_2$ can now automatically prove termination of 15 additional examples from the Termination Problem Data Base, version 4.0. Several of these examples had not been proved terminating by any tool at the competitions before. To run the AProVE implementation via a web-interface and for further details, we refer to `http://aprove.informatik.rwth-aachen.de/eval/maxpolo/`.[45]

Moreover, we also conducted experiments for polynomial interpretations with a negative constant, as described at the end of Section 4.2, on the 1438 examples from the TPDB, version 8.0.2 which are used for the category "TRS standard" in the annual Termination Competition. We used a setting similar to that of Section 3.4 (i.e., we used a basic version of the DP method and no other orders), and we ran the experiments on an Intel Xeon 5140 at 2.33 GHz. With parametric interpretations of the shape $[f](x_1, \ldots, x_n) = \max(a_1\, x_1 + \ldots + a_n\, x_n + \boldsymbol{b}, 0)$ and parameter values from $\{-1, 0, 1\}$ where negative values are allowed only for $\boldsymbol{b}$ (and as an optimization only if $n > 0$), we could only show termination for two additional examples: `TRS/AProVE_06/identity.xml` and `TRS/Transformed_CSR_04/ExProp7_Luc06_GM.xml`.

In both cases, the approach from Chapter 3 fails, whereas the implementation based on the setting of this chapter succeeds. However, in both cases already the old search procedure from AProVE 1.2 with its slightly more general criteria can easily prove termination (cf. Footnote 30, which discusses this in detail for `TRS/AProVE_06/identity.xml`). For `TRS/Transformed_CSR_04/ExProp7_Luc06_GM.xml`, in its normal configuration AProVE manages to prove termination quickly by repeated applications of strongly monotonic polynomial orders to remove rules [Lan79, FGM$^+$07], a switch to innermost termination (which suffices to conclude full termination here), the dependency pair transformation [AG00], an application of the dependency graph processor [GTS05a], removal of non-usable rules [GTS05a], and finally a termination proof via the size-change termination principle [LJB01, TG05].

[45] While this evaluation web page also reports on results for *non-monotonic* reduction pairs, the 15 additional examples on the TPDB are all due to the contributions of this chapter.

While in this experiment, AProVE could show termination using the approximations of Definition 3.8 for 561 examples with 14 timeouts after 60 seconds (total measured runtime: 3482.7 seconds), with our implementation of the setting of the present chapter AProVE could only prove 471 examples as terminating and 440 timeouts occurred (total measured runtime: 33858.4 seconds). These results indicate that for polynomials with a negative constants the specialized approximation-based automation of Chapter 3 is much more suitable in practice than an automation following the more general approach of this chapter. The reason is that in Chapter 3, we benefit from the special shape of the used interpretations. Therefore, this particular setting allows us to work with an approximation-based approach that uses only polynomials without "max" and at the same time to lose only little power due to the approximations.

## 4.5 Summary and Outlook

In this chapter we have shown how to use polynomial interpretations with "max" in weakly monotone algebras for termination proofs with DPs and developed a method to encode the resulting search problems into SMT-NIA. Here, we have used a translation via conditional polynomial constraints to standard unconditional polynomial constraints without "max" . Moreover, we have shown that in general an unrestricted introduction of "max" for parametric interpretations leads to an exponential blowup in the number of constraints to be solved. We have also implemented polynomial interpretations with a negative constant [HM07] as discussed in Chapter 3.

To deal with the exponential blowup, we have provided syntax-based shape heuristics to tailor parametric max-polynomial interpretations for the needs of a given set of term constraints. This way, "max" is used where it is needed, yet the resulting search problems can still be handled with ease. Our experimental evaluation then shows two things: (i) Modern termination tools like AProVE and T$_{\text{T}}$T$_2$ benefit notably from our automation of max-polynomial interpretations, and (ii) for the special case of polynomial interpretations with a negative constant (which also use "max"), our dedicated SMT-NIA encoding from Definition 3.8, which makes use of the special shape of the interpretations, is significantly more suitable than the more widely applicable automation of the present chapter.

Note that our approach captures a large class of interpretations, which allows for a high degree of flexibility also for settings beyond termination analysis of conventional rewriting. In particular, in follow-up work we apply max-polynomial interpretations for termination analysis of *integer term rewriting* [FGP$^+$09]. Here we introduce term rewriting with built-in integers together with corresponding pre-defined operations and provide dedicated methods based on max-polynomial interpretations to $\mathbb{Z}$ for termination analysis of integer term rewriting. In the mean time, integer term rewriting has been applied successfully as a dedicated translation target language for termination analysis of programming languages

like Java Bytecode [OBEG10, BOG11]. Hence, the contributions of this chapter also have a direct impact on automated termination analysis for Java Bytecode.

For future work, one direction is to lift the approach to rational or real max-polynomial interpretations (cf., e.g., our work [FNO+08] on automation of rational polynomial interpretations [Luc05] or the paper [ZM10] for interpretations to the reals). Similarly, one could also extend max-interpretations to the setting of matrix interpretations (e.g., by a component-wise extension).

For a restricted setting, which is however quite amenable to automation, related work [KW08, KW09] already provides a variant of max-interpretations for matrix interpretations. There, Koprowski and Waldmann consider *arctic interpretations*, i.e., interpretations to max/plus algebras based on the *arctic semi-ring*, where also negative numbers can be used. In Chapter 5 we look in more detail into this setting, review the current state of the art in SAT encodings to automate arctic interpretations, and propose a novel SAT encoding for arctic interpretations.

# 5 SAT Encodings for Arctic Termination Revisited

In recent work [KW08, KW09], Koprowski and Waldmann also propose to make use of the max-operation as building block for weakly monotone algebras. Here they embed the max-operation into a more restricted setting than that of Chapter 4. Instead of allowing combinations of "max" with *arbitrary* polynomials, they allow only the "+"-function and constants as possible additional building blocks for interpretation functions. Moreover, they impose the shape $\max(c_1 + x_1, \ldots, c_n + x_n, c_0)$ while at the same time adding the value $-\infty$ to the carrier of the weakly monotone algebra. Based on this setting, they additionally provide both an extension to *arctic matrix interpretations* and to *arctic matrix interpretations with negative constants* (named *below zero* interpretations in [KW08, KW09]).[46]

To solve the constraints arising from parametric interpretations in the arctic setting, Koprowski and Waldmann [KW08, KW09] give a SAT encoding which is based on a *binary* encoding of numbers. In this chapter, we propose an alternative SAT encoding based on a *unary* representation of numbers. Our experiments indicate that this—asymptotically larger—representation of the constraints as a SAT problem leads to notable performance improvements with the current state of the art in SAT solving, as exemplified, e.g., by the solver MiniSAT [ES04].

In Chapter 5.1 we present the setting for this chapter. Here we also contribute a novel transformation that allows for more flexible interpretations with negative values than earlier work [KW08, KW09]. Chapter 5.2 presents the binary SAT encoding based on [KW08, KW09] that we use as a reference encoding. In Chapter 5.3 we then present our novel *unary* SAT encoding to search for arctic interpretations. Chapter 5.4 discusses related work, both with respect to arctic interpretations and with respect to the approach underlying our unary SAT encoding. In Chapter 5.5 we provide an empirical evaluation of our novel encoding in several settings to demonstrate the practical impact of our contribution, and in Chapter 5.6 we conclude.

[46] See also Footnote 6 for the origin of the term "arctic" in the context of algebras.

## 5.1 Arctic Interpretations

This section sets up the scenario for constraint-based termination analysis using arctic algebras. As in the previous chapters, a parametric approach *à la* [CMTU05] gives rise to existential constraints over the carrier of the algebra. The presentation of this section is based on [KW08, KW09].

### Arctic Arithmetic

In the setting of arctic algebras, we consider the carriers $\mathbb{A}_{\mathbb{N}} = \mathbb{N} \cup \{-\infty\}$ and $\mathbb{A}_{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty\}$. As building block of our interpretations, we use the operation "$\oplus$" defined as follows:

- $x \oplus y = \max(x, y)$, if both $x, y \neq -\infty$,
- $x \oplus y = x$, if $y = -\infty$, and
- $x \oplus y = y$, otherwise.

Moreover, we use the operation "$\otimes$" defined as follows:

- $x \otimes y = x + y$, if both $x, y \neq -\infty$
- $x \otimes y = -\infty$, otherwise.

Thus, $-\infty$ and 0 are the neutral elements of the $\oplus$-operation and the $\otimes$-operation, respectively. Since "$\oplus$" and "$\otimes$" provide a direct generalization of the standard operations "max" and "+" to the additional carrier element $-\infty$, algebras like $(\mathbb{A}_{\mathbb{N}}, \oplus, \otimes)$ and $(\mathbb{A}_{\mathbb{Z}}, \oplus, \otimes)$ are also known as *max/plus algebras*. Since these algebras also satisfy the semi-ring properties, they are also called *max/plus semi-rings* or *arctic semi-rings*. Because of their role in arctic semi-rings, the operations "$\oplus$" and "$\otimes$" are often referred to as *arctic addition* and *arctic multiplication*, respectively.[47] Note that for the operations "$\oplus$" and "$\otimes$" the usual arithmetic laws for addition and multiplication hold, i.e., both "$\oplus$" and "$\otimes$" are associative and commutative, and by $x \otimes (y \oplus z) = x \otimes y \oplus x \otimes z$ we also have distributivity.

**Example 5.1** (Arctic Arithmetic)**.** With the above operations, the following equalities hold:

- $(2 \oplus 3) \otimes (4 \oplus 5) = 3 \otimes 5 = 8$
- $-\infty \otimes 3 \oplus 1 = -\infty \oplus 1 = 1$
- $(-\infty \oplus 3) \otimes 1 = 3 \otimes 1 = 4$

[47] Therefore, we let "$\otimes$" bind stronger than "$\oplus$".

As in [KW08, KW09], we *compare* values from an arctic algebra as follows:

- $x \geq y$ iff $y = -\infty$ or $x, y \in \mathbb{N}$ (or $\mathbb{Z}$) with $x \geq_{\mathbb{N}/\mathbb{Z}} y$
- $x \gg y$ iff $y = -\infty$ or $x, y \in \mathbb{N}$ (or $\mathbb{Z}$) with $x >_{\mathbb{N}/\mathbb{Z}} y$

Note that $\gg$ is neither a strict ordering nor well founded since we have $-\infty \gg -\infty$.

## Arctic Matrix Interpretations

As interpretations $[\cdot]$ for function symbols, [KW08, KW09] propose linear functions on *vectors* of arctic numbers with $d \geq 1$ entries. The value $d$ is also called the *dimension* of the interpretation. These interpretations have the following shape:

$$[f](x_1, \ldots, x_n) = F_1 \otimes x_1 \ \oplus \ \ldots \ \oplus \ F_n \otimes x_n \ \oplus \ F_0 \tag{5.1}$$

Here, $x_1, \ldots, x_n$ and $F_0$ are (column) vectors of $d$ entries, and $F_1, \ldots, F_n$ are $d \times d$-*matrices*. This is also the reason why such interpretations are commonly known as *arctic matrix interpretations*.[48] Depending on the setting, the entries of $F_0, F_1, \ldots, F_n$ are elements of $\mathbb{A}_{\mathbb{N}}$ or $\mathbb{A}_{\mathbb{Z}}$. We call an arctic number $x$ *finite* iff $x \neq -\infty$, and we call $x$ *positive* iff $x \geq 0$. A vector $F$ with arctic entries $a_i$ is called *finite* iff $a_1$ is finite, and it is called *positive* iff $a_1$ is positive. A matrix $G$ with arctic entries $b_{i,j}$ is called *finite* iff $b_{1,1}$ is finite.

Note that so far we have only defined the operations "$\oplus$" and "$\otimes$" for arctic numbers, not matrices or vectors. The extension of these operations from numbers to matrices (and vectors) is completely analogous to standard matrix addition and multiplication. For two $k \times n$-matrices $M$ and $N$ with entries $a_{i,j}$ and $b_{i,j}$, respectively, the entries $c_{i,j}$ of the $k \times n$-matrix $M \oplus N$ are defined as $c_{i,j} = a_{i,j} \oplus b_{i,j}$, i.e., "$\oplus$" extends component-wise to matrices. For a $k \times m$-matrix $M$ with entries $a_{i,j}$ and a $m \times n$-matrix $N$ with entries $b_{i,j}$, the entries $c_{i,j}$ of the $k \times n$-matrix $M \otimes N$ are defined as $c_{i,j} = a_{i,1} \otimes b_{1,j} \oplus \ldots \oplus a_{i,m} \otimes b_{m,j}$.

For the comparison operations $\geq$ and $\gg$ we use component-wise extensions to matrices and vectors. By [KW08, Lemma 8], we get the following criterion for comparison of two linear functions $f$ and $g$ on arctic vectors with $f(x_1, \ldots, x_n) = F_1 \otimes x_1 \ \oplus \ \ldots \ \oplus \ F_n \otimes x_n \ \oplus \ F_0$ and $g(x_1, \ldots, x_n) = G_1 \otimes x_1 \ \oplus \ \ldots \ \oplus \ G_n \otimes x_n \ \oplus \ G_0$.

$$\begin{array}{ll} & F_1 \geq G_1 \wedge \ \ldots \ \wedge \ F_n \geq G_n \ \wedge \ F_0 \geq G_0 \\ \text{implies} & \forall x_1, \ldots, x_n : f(x_1, \ldots, x_n) \ \geq \ g(x_1, \ldots, x_n) \end{array} \tag{5.2}$$

$$\begin{array}{ll} & F_1 \gg G_1 \wedge \ \ldots \ \wedge \ F_n \gg G_n \ \wedge \ F_0 \gg G_0 \\ \text{implies} & \forall x_1, \ldots, x_n : f(x_1, \ldots, x_n) \ \gg \ g(x_1, \ldots, x_n) \end{array} \tag{5.3}$$

[48] For matrix interpretations with standard addition and multiplication, cf., e.g., [HW06, EWZ08] for the carrier $\mathbb{N}^d$ and [GHW07, ALN09, ZM10] for the carriers $\mathbb{Q}^d_{\geq 0}$ and $\mathbb{R}^d_{\geq 0}$.

As stated in [KW08], this criterion for comparison of linear arctic functions is the "arctic counter-part" of the absolute positiveness criterion [HJ98] used for comparison of standard polynomial functions (cf. Theorem 2.18). Note that in (5.3) we require a decrease with $\gg$ in *all* inequalities. The reason for this is that if we used $\geq$ instead of $\gg$ (as in Theorem 2.18), e.g., for $f(x) = 0 \otimes x \oplus 2$ and $g(x) = 0 \otimes x \oplus 1$ we would get the constraints $0 \geq 0$ and $2 \gg 1$, which are satisfied. However, for $x = 3$, we would get $f(3) = 0 \otimes 3 \oplus 2 = 3 \not\gg 3 = 0 \otimes 3 \oplus 1 = g(3)$. As [KW08, KW09] mention, this is because in general "$\oplus$" requires a strict increase in *both* arguments in order to cause a strict increase in the result. An exception is the case that one of the arguments is $-\infty$. Here indeed $-\infty \oplus x \gg -\infty \oplus y$ always holds if $x \gg y$ holds (which is also the explanation that [KW08, KW09] give for defining $-\infty \gg -\infty$).

Of course, we can use these arctic variants of absolute positiveness also in the parametric setting. The resulting *arctic constraints* (over *arctic variables*) and their semantics are then defined completely analogously to Diophantine constraints (over Diophantine variables) and their semantics in Definition 2.24.

## Weakly Monotone Arctic Algebras

Based on these preliminaries, [KW08, KW09] present two families of weakly monotone algebras with arctic numbers.

**Theorem 5.2** (Arctic Algebras Above Zero [KW08, Theorem 12])**.** *Let $d \in \mathbb{N}$, let $\mathcal{A} = (\mathbb{N} \times \mathbb{A}_{\mathbb{N}}^{d-1}, [\cdot]_{\mathcal{A}})$ be an $\mathcal{F}$-algebra. Moreover, for all $f \in \mathcal{F}$ let*

$$[f](x_1, \ldots, x_n) \quad = \quad F_1 \otimes x_1 \ \oplus \ \ldots \ \oplus \ F_n \otimes x_n \ \oplus \ F_0$$

*as in* (5.1) *where all entries of $F_0, F_1, \ldots, F_n$ are elements of $\mathbb{A}_{\mathbb{N}}$ and where at least one of $F_0, F_1, \ldots, F_n$ is finite. Then $(\mathcal{A}, \geq, \gg)$ is a weakly monotone algebra.*[49][50]

**Example 5.3** (Bits version 1, Revisited)**.** Consider again the term constraints from Example 2.14:[51]

$$\mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{HALF}(x) \tag{2.10}$$

$$\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{HALF}(x) \tag{2.11}$$

$$\mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{BITS}(\mathsf{s}(\mathsf{half}(x))) \tag{2.12}$$

[49] While in this thesis we pose more requirements on weakly monotone algebras than [KW08, KW09], the statement also holds in our setting.

[50] Note that $\gg$ is well founded on $\mathbb{N} \times \mathbb{A}_{\mathbb{N}}^{d-1}$ since $\gg$ is well founded on $\mathbb{N}$.

[51] Note that also arctic interpretations are $\mathcal{C}_\varepsilon$-compatible: Using $[\mathsf{c}](x_1, x_2) = x_1 \oplus x_2$, both $\mathsf{c}(x, y) \geq x$ and $\mathsf{c}(x, y) \geq y$ are satisfied. This allows us to disregard the term constraints for the bits-rules here.

$$\mathsf{half}(\mathsf{0}) \succsim \mathsf{0} \tag{2.13}$$

$$\mathsf{half}(\mathsf{s}(\mathsf{0})) \succsim \mathsf{0} \tag{2.14}$$

$$\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succsim \mathsf{s}(\mathsf{half}(x)) \tag{2.15}$$

We now show how to translate these term constraints to satisfiability of a constraint problem over an arctic carrier, thus automating Theorem 5.2. Analogous to Example 2.21, we use a parametric arctic interpretation of dimension 1 (which suffices for the example and eases readability) as follows: $[\mathsf{BITS}](x_1) = B_1 \otimes x_1 \oplus B_0$, $[\mathsf{HALF}](x_1) = H_1 \otimes x_1 \oplus H_0$, $[\mathsf{half}](x_1) = h_1 \otimes x_1 \oplus h_0$, $[\mathsf{s}](x_1) = s_1 \otimes x_1 \oplus s_0$, and $[\mathsf{0}] = z_0$. Here, $B_0, B_1, H_0, \ldots$ are the parameters of the interpretation, and we can use values from $\mathbb{A}_{\mathbb{N}}$ to instantiate them.

Using this interpretation, we transform the above term constraints in an analogous way to Example 2.22, and for the term constraints (2.13), (2.14), and (2.15), we obtain:

$$h_0 \oplus h_1 \otimes z_0 \geq z_0 \tag{5.4}$$

$$h_0 \oplus h_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes z_0 \geq z_0 \tag{5.5}$$

$$h_0 \oplus h_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes s_1 \otimes x \geq s_0 \oplus s_1 \otimes h_0 \oplus s_1 \otimes h_1 \otimes x \tag{5.6}$$

Here the term constraints (2.10), (2.11), and (2.12) yield the following constraints:

$$H_0 \oplus H_1 \otimes s_0 \oplus H_1 \otimes s_1 \otimes s_0 \oplus H_1 \otimes s_1 \otimes s_1 \otimes x \gg H_0 \oplus H_1 \otimes x \tag{5.7}$$

$$B_0 \oplus B_1 \otimes s_0 \oplus B_1 \otimes s_1 \otimes s_0 \oplus B_1 \otimes s_1 \otimes s_1 \otimes x \gg H_0 \oplus H_1 \otimes x \tag{5.8}$$

$$\begin{aligned} B_0 \oplus B_1 \otimes s_0 \oplus B_1 \otimes s_1 \otimes s_0 \oplus B_1 \otimes s_1 \otimes s_1 \otimes x \\ \gg B_0 \oplus B_1 \otimes s_0 \oplus B_1 \otimes s_1 \otimes h_0 \oplus B_1 \otimes s_1 \otimes h_1 \otimes x \end{aligned} \tag{5.9}$$

As in Example 2.22, here only the variable $x$ is (implicitly) universally quantified.

So far the treatment of the example is exactly analogous to standard linear polynomial interpretations over $\mathbb{N}$. However, in addition we now also need to impose that for each symbol $f$, at least one of the coefficients of its interpretation is *finite*:

$$(H_0 \neq -\infty \vee H_1 \neq -\infty) \quad \wedge \tag{5.10}$$

$$(B_0 \neq -\infty \vee B_1 \neq -\infty) \quad \wedge \tag{5.11}$$

$$(h_0 \neq -\infty \vee h_1 \neq -\infty) \quad \wedge \tag{5.12}$$

$$(s_0 \neq -\infty \vee s_1 \neq -\infty) \quad \wedge \tag{5.13}$$

$$z_0 \neq -\infty \tag{5.14}$$

In order to eliminate the universally quantified variable $x$ from the constraints (5.4) – (5.9), we apply the criteria from (5.2) and (5.3), i.e., arctic absolute positiveness. This

way, here we obtain the following conjunction.

$$h_0 \oplus h_1 \otimes z_0 \geq z_0 \quad \wedge \tag{5.15}$$
$$h_0 \oplus h_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes z_0 \geq z_0 \quad \wedge \tag{5.16}$$
$$h_0 \oplus h_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes s_0 \geq s_0 \oplus s_1 \otimes h_0 \quad \wedge \tag{5.17}$$
$$h_1 \otimes s_1 \otimes s_1 \geq s_1 \otimes h_1 \quad \wedge \tag{5.18}$$
$$H_0 \oplus H_1 \otimes s_0 \oplus H_1 \otimes s_1 \otimes s_0 \gg H_0 \quad \wedge \tag{5.19}$$
$$H_1 \otimes s_1 \otimes s_1 \gg H_1 \quad \wedge \tag{5.20}$$
$$B_0 \oplus B_1 \otimes s_0 \oplus B_1 \otimes s_1 \otimes s_0 \gg H_0 \quad \wedge \tag{5.21}$$
$$B_1 \otimes s_1 \otimes s_1 \gg H_1 \quad \wedge \tag{5.22}$$
$$B_0 \oplus B_1 \otimes s_0 \oplus B_1 \otimes s_1 \otimes s_0 \gg B_0 \oplus B_1 \otimes s_0 \oplus B_1 \otimes s_1 \otimes h_0 \quad \wedge \tag{5.23}$$
$$B_1 \otimes s_1 \otimes s_1 \gg B_1 \otimes s_1 \otimes h_1 \tag{5.24}$$

Now any model for the conjunction (5.10) – (5.24) can also be used to instantiate the parametric arctic interpretation to a concrete arctic interpretation. This concrete arctic interpretation then gives rise to a reduction pair that solves the initial term constraints. For instance, the assignment $B_0 = h_0 = -\infty$, $B_1 = H_0 = h_1 = s_0 = 0$, and $H_1 = s_1 = z_0 = 1$, solves the above constraints. This way, we obtain the concrete arctic interpretation $\mathcal{A}$ with $[\mathsf{BITS}]_{\mathcal{A}}(x_1) = 0 \otimes x_1 \oplus -\infty$, $[\mathsf{HALF}]_{\mathcal{A}}(x_1) = 1 \otimes x_1 \oplus 0$, $[\mathsf{half}]_{\mathcal{A}}(x_1) = 0 \otimes x_1 \oplus -\infty$, $[\mathsf{s}]_{\mathcal{A}}(x_1) = 1 \otimes x_1 \oplus 0$, and $[\mathsf{0}]_{\mathcal{A}} = 1$. This then yields the weakly monotone algebra $(\mathcal{A}, \geq, \gg)$ with $\mathcal{A} = (\mathbb{N}, [\cdot]_{\mathcal{A}})$ and the corresponding reduction pair $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$, which proves termination of Example 2.1 also via an arctic interpretation.

It is interesting to note that in contrast to the max-polynomial interpretations of the previous chapter, it is not necessary to perform case analyses to deal with the $\oplus$-operation (corresponding to the max-operation in Chapter 4).

The price to pay for this conveniently automatable setting, however, is that of expressivity. Note that, e.g., for dimension 1, one can use a max-polynomial interpretation to simulate the effect of an arctic interpretations $[f]_{\mathcal{A}}(x_1, \ldots, x_n) = F_1 \otimes x_1 \oplus \ldots \oplus F_n \otimes x_n \oplus F_0$ as used in Theorem 5.2, i.e., where $F_0, \ldots, F_N \in \mathbb{A}_{\mathbb{N}}$ and where $F_0 \neq -\infty \vee \ldots \vee f_n \neq -\infty$. Such a max-polynomial interpretation $[\cdot]_{\mathcal{B}}$ would have the shape $[f]_{\mathcal{B}}(x_1, \ldots, x_n) = \max(f_{1,0} + f_{1,1} \cdot x_1, \ldots, f_{n,0} + f_{n,1} \cdot x_n, f_{0,0})$. Here we set $f_{i,0} = f_{i,1} = 0$ if $F_i = -\infty$ and $f_{i,0} = F_i, f_{i,1} = 1$ otherwise. The intuition is that an arctic coefficient $F_i = -\infty$ is used to make $[\cdot]_{\mathcal{A}}$ filter away its argument $x_i$. The same effect is achieved by (classically) multiplying $x_i$ with a (classical) coefficient 0 in $[\cdot]_{\mathcal{B}}$.

However, for a *non-linear* max-polynomial interpretation like $[f](x_1, x_2) = x_1 \cdot x_2$, there is no corresponding interpretation on arctic level, where multiplication of variables cannot be simulated easily. It is not even possible to express a polynomial interpretation by a classical *sum* of variables $[f](x_1, x_2) = x_1 + x_2$ with the approach to arctic interpretations

described in this chapter (where the operation "⊗" is never applied to two variables).

Still, for the specific setting of *string rewrite systems (SRSs)*, i.e., TRSs whose signature only contains symbols with just a single argument, it turns out that these shortcomings of arctic interpretations do not matter much in practice. The reason is that here one does not have different arguments that one could combine by an arithmetic operation. Indeed, as indicated by the termination competitions that took place since the publication of [KW08], arctic interpretations have become a key technique for termination analysis of SRSs. One of the reasons for this is the impact of arctic interpretations with numbers *below zero*.

**Theorem 5.4** (Arctic Algebras Below Zero [KW08, Theorem 14])**.** *Let $d \in \mathbb{N}$, let $\mathcal{A} = (\mathbb{N} \times \mathbb{A}_{\mathbb{Z}}^{d-1}, [\cdot]_{\mathcal{A}})$ be an $\mathcal{F}$-algebra. Moreover, for all $f \in \mathcal{F}$ let*

$$[f](x_1, \ldots, x_n) \quad = \quad F_1 \otimes x_1 \ \oplus \ \ldots \ \oplus \ F_n \otimes x_n \ \oplus \ F_0$$

*as in* (5.1) *where all entries of $F_0, F_1, \ldots, F_n$ are elements of $\mathbb{A}_{\mathbb{Z}}$ and where $F_0$ is positive. Then $(\mathcal{A}, \geq, \gg)$ is a weakly monotone algebra.*

In contrast to classic polynomial interpretations with negative constants, for below zero arctic interpretations there is no need for a special treatment via approximations (cf. Chapter 3). Instead, the constraints arising from arctic interpretations below zero are almost identical to those for arctic interpretations where only non-negative numbers are considered (together with $-\infty$).

**Example 5.5** (Bits version 2, Revisited)**.** Consider again the term constraints (3.4), (2.13), (2.14), and (2.15), which arise during termination analysis of Example 3.1:

$$\mathsf{BITS}(\mathsf{s}(x)) \succ \mathsf{BITS}(\mathsf{half}(\mathsf{s}(x))) \tag{3.4}$$
$$\mathsf{half}(\mathsf{0}) \succsim \mathsf{0} \tag{2.13}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{0})) \succsim \mathsf{0} \tag{2.14}$$
$$\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succsim \mathsf{s}(\mathsf{half}(x)) \tag{2.15}$$

To solve them, now we want to use a reduction pair based on Theorem 5.2.

We again use a parametric arctic interpretation of dimension 1 as follows: $[\mathsf{BITS}]_{\mathcal{B}}(x_1) = B_1 \otimes x_1 \oplus B_0$, $[\mathsf{half}]_{\mathcal{B}}(x_1) = h_1 \otimes x_1 \oplus h_0$, $[\mathsf{s}]_{\mathcal{B}}(x_1) = s_1 \otimes x_1 \oplus s_0$, and $[\mathsf{0}]_{\mathcal{B}} = z_0$. Again, $B_0, B_1, h_0, \ldots$ are the parameters of the interpretation. In contrast to Example 5.3, we now consider values from $\mathbb{A}_{\mathbb{Z}}$ (including finite negative numbers) to instantiate these parameters.

From the term constraints (2.13), (2.14), and (2.15) we again obtain the following

constraints:

$$h_0 \oplus h_1 \otimes z_0 \geq z_0 \tag{5.4}$$

$$h_0 \oplus h_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes z_0 \geq z_0 \tag{5.5}$$

$$h_0 \oplus h_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes s_1 \otimes x \geq s_0 \oplus s_1 \otimes h_0 \oplus s_1 \otimes h_1 \otimes x \tag{5.6}$$

Likewise, from (3.4) we get

$$\begin{aligned} B_0 \oplus B_1 \otimes s_0 \oplus B_1 \otimes s_1 \otimes x \\ \gg B_0 \oplus B_1 \otimes h_0 \oplus B_1 \otimes h_1 \otimes s_0 \oplus B_1 \otimes h_1 \otimes s_1 \otimes x \end{aligned} \tag{5.25}$$

From Theorem 5.4, we get the requirement that for each symbol $f$, the constant part of its interpretation $[f]$ must be *positive*:

$$B_0 \geq 0 \wedge h_0 \geq 0 \wedge s_0 \geq 0 \wedge z_0 \geq 0 \tag{5.26}$$

Using the arctic absolute positiveness criteria from (5.2) and (5.3), we eliminate the universally quantified variable $x$ from the constraints (5.4), (5.5), (5.6), and (5.25). This transformation yields the following conjunction:

$$h_0 \oplus h_1 \otimes z_0 \geq z_0 \quad \wedge \tag{5.15}$$

$$h_0 \oplus h_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes z_0 \geq z_0 \quad \wedge \tag{5.16}$$

$$h_0 \oplus h_1 \otimes s_0 \oplus h_1 \otimes s_1 \otimes s_0 \geq s_0 \oplus s_1 \otimes h_0 \quad \wedge \tag{5.17}$$

$$h_1 \otimes s_1 \otimes s_1 \geq s_1 \otimes h_1 \quad \wedge \tag{5.18}$$

$$B_0 \oplus B_1 \otimes s_0 \gg B_0 \oplus B_1 \otimes h_0 \oplus B_1 \otimes h_1 \otimes s_0 \quad \wedge \tag{5.27}$$

$$B_1 \otimes s_1 \gg B_1 \otimes h_1 \otimes s_1 \tag{5.28}$$

Using the assignment $h_1 = -1$, $B_0 = B_1 = h_0 = z_0 = 0$, and $s_0 = s_1 = 1$, we obtain the concrete arctic interpretation $[\cdot]_{\mathcal{C}}$ where we have $[\mathsf{BITS}]_{\mathcal{C}}(x_1) = 0 \otimes x_1 \oplus 0$, $[\mathsf{half}]_{\mathcal{C}}(x_1) = -1 \otimes x_1 \oplus 0$, $[\mathsf{s}]_{\mathcal{C}}(x_1) = 1 \otimes x_1 \oplus 1$, and $[\mathsf{0}]_{\mathcal{C}} = 0$. This way, we get the weakly monotone algebra $(\mathcal{C}, \geq, \gg)$ with $\mathcal{C} = (\mathbb{N}, [\cdot]_{\mathcal{C}})$ and the corresponding reduction pair $(\succsim_{\mathcal{C}}, \succ_{\mathcal{C}})$, which can be used to conclude the termination proof for Example 3.1 via an arctic below zero interpretation.

## Shifting Negative Numbers

Koprowski and Waldmann [KW08, KW09] propose using two's complement to represent integer values in a SAT encoding. This classic encoding for CPU arithmetic allows to represent (finite) values from $\{2^{-k}, \ldots, 2^k - 1\}$ for some $k$. However, it can be desirable to

use a search space $\{\ell_1, \ldots, \ell_2\}$ with $\ell_1 < 0 < \ell_2$ where $\ell_2 > -\ell_1$ holds. For instance, in the Termination Competition of 2011, AProVE searched for below zero interpretations with parameter values from $\{-\infty, -1, 0, 1, 2, 3\}$ (among several choices). Thus, to increase flexibility regarding the set of values to be considered for the parametric interpretation, here we advocate (and use) an alternative approach, which we illustrate by an example.

If we wish to search for finite values starting, e.g., from $\ell_1 = -1$, we use a parametric interpretation like $[\mathsf{BITS}]_{\mathcal{D}}(x_1) = -1 \otimes B'_1 \otimes x_1 \oplus -1 \otimes B'_0$, $[\mathsf{half}]_{\mathcal{D}}(x_1) = -1 \otimes h'_1 \otimes x_1 \oplus -1 \otimes h'_0$, $[\mathsf{s}]_{\mathcal{D}}(x_1) = -1 \otimes s'_1 \otimes x_1 \oplus -1 \otimes s'_0$, and $[\mathsf{0}]_{\mathcal{D}} = -1 \otimes z'_0$. Here, for the parameters $B'_0, B'_1, h'_0, \ldots$ we now only need to consider values from $\mathbb{A}_{\mathbb{N}}$. Then instead of the arctic constraint (5.16), we get the following arctic constraint from the term constraint (2.14):

$$-1 \otimes h'_0 \oplus -2 \otimes h'_1 \otimes s'_0 \oplus -3 \otimes h'_1 \otimes s'_1 \otimes z'_0 \geq -1 \otimes z'_0 \tag{5.29}$$

To eliminate the finite negative constants from this constraint, we now *shift* the whole constraint by an arctic multiplication with 3, which is the arctic multiplicative inverse to the smallest finite number in the constraint (i.e., $-3$). Using distributivity, we obtain the following arctic constraint:

$$2 \otimes h'_0 \oplus 1 \otimes h'_1 \otimes s'_0 \oplus h'_1 \otimes s'_1 \otimes z'_0 \geq 2 \otimes z'_0 \tag{5.30}$$

We proceed similarly for all arctic constraints to eliminate negative numbers.

In our example, we only need to consider assignments of the parameters to arctic numbers from $\mathbb{A}_{\mathbb{N}}$, and we can find a corresponding assignment with $h'_1 = 0$, $B'_0 = B'_1 = h'_0 = z'_0 = 1$, and $s'_0 = s'_1 = 2$ to solve the overall resulting conjunction. This way, we can instantiate $[\cdot]_{\mathcal{D}}$ to get $[\cdot]_{\mathcal{C}}$.

Thus, in the following SAT encodings we only need to consider the case where we encode variables and values over a subset of $\mathbb{A}_{\mathbb{N}}$.

## 5.2 A Binary SAT Encoding for Arctic Constraints

In [KW08, KW09], Koprowski and Waldmann sketch a SAT encoding which is based on a binary representation for (finite) numbers. In this section we elaborate on a concrete SAT encoding based on this description. Based on the previous considerations on shifting negative numbers, it suffices to consider the case where only $-\infty$ and positive arctic numbers are used.

So we are looking for an arctic constraint interpretation which assigns values from $\mathbb{A}_{\mathbb{N}}$. To obtain a finite SAT encoding, we render the search space finite, fix a value $k \in \mathbb{N}$ and only consider interpretations $\{-\infty, 0, \ldots, 2^k - 1\}$.[52]

[52]Using additional constraints to prohibit certain values, we can of course also search for interpretations

Following [KW08, KW09], an arctic variable which may take values from the set $\{-\infty, 0, \ldots, 2^k - 1\}$ is encoded to a bit tuple of $k+1$ elements. So for an arctic variable $a$, in a first encoding we would get a representation by a bit tuple $\langle a_k, \ldots, a_1 \,;\, a_0\rangle$. If $a_0$ is set, the encoded value is $-\infty$ ($a_0$ is called the *infinity bit*). Otherwise, the (finite) value of $a$ is given by $2^{k-1} \cdot a_k + \ldots + 2^1 \cdot a_2 + 2^0 \cdot a_1$, i.e., the bit $a_k$ is the most significant bit in our representation.[53] Koprowski and Waldmann additionally propose that a set infinity bit should explicitly imply that all other bits are set to zero. This way, in the (binary) SAT encoding $||\cdot||_{bin}$ we obtain the following representation for arctic variables $a$:

$$||a||_{bin} = \langle a_k \wedge \neg a_0,\ \ldots,\ a_1 \wedge \neg a_0 \,;\, a_0\rangle \tag{5.31}$$

As an example, consider an arctic variable $a$ where $k = 2$, i.e., we allow values from the set $\{-\infty, 0, 1, 2, 3\}$. To ease readability, in the examples in this and also the next section we write $\beta_i$ to denote $a_i \wedge \neg a_0$ for $i \geq 1$. Then we get $||a||_{bin} = \langle \beta_2, \beta_1 \,;\, a_0\rangle$.

For arctic constants, we thus obtain the following encoding:

$$||-\infty||_{bin} = \langle 0 \,;\, 1\rangle \tag{5.32}$$

$$||0||_{bin} = \langle 0 \,;\, 0\rangle \tag{5.33}$$

$$||n||_{bin} = \langle b_m,\ \ldots,\ b_1 \,;\, 0\rangle \qquad \text{for } n > 0 \tag{5.34}$$

Here we have $b_i \in \{0, 1\}$, $n = 2^{m-1} \cdot b_m + \ldots + 2^1 \cdot b_2 + 2^0 \cdot b_1$. For $n > 0$, we require $b_m = 1$ to avoid unnecessary leading zeros. So we have, e.g., $||2||_{bin} = \langle 1, 0 \,;\, 0\rangle$.

In the following encodings of comparisons and operations for arctic expressions, let $||p||_{bin} = \langle \varphi_k,\ \ldots,\ \varphi_1 \,;\, \varphi_0\rangle$ and $||q||_{bin} = \langle \tau_k,\ \ldots,\ \tau_1 \,;\, \tau_0\rangle$. We use 0-padding for the most significant bits where necessary so that we can assume equal length of argument tuples. We also apply Boolean simplifications so that, e.g., $x \leftrightarrow 0$ becomes $\neg x$.

Unfortunately, Koprowski and Waldmann do not give details on their encoding for comparisons of values from $\mathbb{A}_{\mathbb{N}}$, so we adapt the corresponding encodings given in the papers [CLS06, FGM+07, CLS08], where comparison on $\mathbb{N}$ is encoded. This gives rise to the following encodings for arctic comparison.

$$||p \gg q||_{bin} \quad = \quad \tau_0 \ \vee\ (\neg\varphi_0 \wedge B^{>}(\langle \varphi_k,\ \ldots,\ \varphi_1\rangle, \langle \tau_k,\ \ldots,\ \tau_1\rangle)) \tag{5.35}$$

$$||p = q||_{bin} \quad = \quad (\varphi_0 \leftrightarrow \tau_0)\ \wedge\ (\varphi_1 \leftrightarrow \tau_1)\ \wedge\ \ldots\ \wedge\ (\varphi_k \leftrightarrow \tau_k) \tag{5.36}$$

$$||p \geq q||_{bin} \quad = \quad ||p \gg q||_{bin}\ \vee\ ||p = q||_{bin} \tag{5.37}$$

The encoding for comparison of formula tuples denoting (finite) natural numbers $B^{>}$ is

with values from $\{-\infty, 0, \ldots, \ell\}$ if $\ell \neq 2^k - 1$ for any $k \in \mathbb{N}$.

[53] To highlight the purpose of the infinity bit, we separate it via ";" and not via "," from its neighboring bit, cf. [KW08, KW09].

taken from [CLS06, FGM+07, CLS08], i.e., we define:

$$B^{>}(\langle\varphi_1\rangle, \langle\tau_1\rangle) = (\varphi_1 \wedge \neg\tau_1) \tag{5.38}$$

$$\begin{aligned} B^{>}(\langle\varphi_k, \ldots, \varphi_1\rangle, \langle\tau_k, \ldots, \tau_1\rangle) &= (\varphi_k \wedge \neg\tau_k) \vee ((\varphi_k \leftrightarrow \tau_k) \\ &\quad \wedge B^{>}(\langle\varphi_{k-1}, \ldots, \varphi_1\rangle, \langle\tau_{k-1}, \ldots, \tau_1\rangle)), \text{ if } k \geq 2 \end{aligned} \tag{5.39}$$

Thus, bit tuples are compared lexicographically on their finite parts, and the properties of the infinity bits are taken into account.

To compute the arctic addition operation "$\oplus$", Koprowski and Waldmann propose to use the maximum function for the finite part of an arctic value and the conjunction of the infinity bits for the infinite part. To express $p \oplus q$ in binary, we use "if $p \gg q$ then $p$ else $q$", which we can easily express on bit level. This gives rise to the following encoding:

$$||p \oplus q||_{bin} = \langle \eta \,?\, \varphi_k : \tau_k,\ \ldots,\ \eta \,?\, \varphi_1 : \tau_1\ ;\ \varphi_0 \wedge \tau_0 \rangle \tag{5.40}$$

$$\text{where} \quad \eta = ||p \gg q||_{bin} \tag{5.41}$$

Here $\alpha_1 \,?\, \alpha_2 : \alpha_3$ is a ternary propositional connective which is equivalent to $(\alpha_1 \rightarrow \alpha_2) \wedge (\neg\alpha_1 \rightarrow \alpha_3)$, i.e., "if-then-else". Note that although in the above definition the subformula $\eta$ occurs several times, it is represented only once by our implementation in AProVE. The reason is that we *share* identical subformulas, which is a technique referred to as *structural hashing*, e.g., by [ES06]. This sharing effect is preserved by the later conversion of the overall propositional formula to conjunctive normal form.

Note also that [KW08, KW09] do not give an explicit definition for the encoding of the maximum function on natural numbers. Nevertheless, we believe that our representation is a rather natural one that should also correspond sufficiently to the intended setting.

For instance, to encode $2 \oplus a$, we first need to encode $2 \gg a$ as follows:[54]

$$||2 \gg a||_{bin} = a_0 \vee (\neg 0 \wedge B^{>}(\langle 1, 0\rangle, \langle\beta_2, \beta_1\rangle)) \tag{5.42}$$

$$= a_0 \vee (\neg 0 \wedge ((1 \wedge \neg\beta_2) \vee ((1 \leftrightarrow \beta_2) \wedge B^{>}(\langle 0\rangle, \langle\beta_1\rangle)))) \tag{5.43}$$

$$= a_0 \vee (\neg 0 \wedge ((1 \wedge \neg\beta_2) \vee ((1 \leftrightarrow \beta_2) \wedge (0 \wedge \neg\beta_1)))) \tag{5.44}$$

$$= a_0 \vee \neg\beta_2 \tag{5.45}$$

Then we can construct the encoding for $2 \oplus a$ itself:

$$||2 \oplus a||_{bin} = \langle (a_0 \vee \neg\beta_2) \,?\, 1 : \beta_2,\ (a_0 \vee \neg\beta_2) \,?\, 0 : \beta_1\ ;\ 0 \wedge a_0 \rangle \tag{5.46}$$

$$= \langle (a_0 \vee \neg\beta_2) \,?\, 1 : \beta_2,\ (a_0 \vee \neg\beta_2) \,?\, 0 : \beta_1\ ;\ 0 \rangle \tag{5.47}$$

For the arctic multiplication operation "$\otimes$" the authors of [KW08, KW09] state that

[54] To ease readability, we apply standard Boolean simplifications, e.g., we replace $1 \wedge \neg\beta_2$ by $\neg\beta_2$, etc.

the result of $x \otimes y$ is infinite if $x$ or $y$ is infinite. Moreover, they state that for finite $x$ and $y$ the result is obtained via the encoding of the operation "+" on $\mathbb{N}$ on the finite parts of $x$ and $y$ where one additionally again adds the requirement that the bits of the finite parts of the result are only set if the infinity bit of the result is unset. Also for the encoding of "+", [KW08, KW09] do not give an explicit definition, but it is likely that the intended encoding is that of [EWZ08] (corresponding to that of [FGM+07]) since there is a non-empty intersection of the set of authors of the papers. This gives rise to the following encoding:

$$||p \otimes q||_{bin} = \langle \xi_{k+1} \wedge \neg(\varphi_0 \vee \tau_0),\ \ldots,\ \xi_1 \wedge \neg(\varphi_0 \vee \tau_0)\ ;\ \varphi_0 \vee \tau_0 \rangle \tag{5.48}$$

$$\text{where} \quad \langle \xi_{k+1}, \ldots, \xi_1 \rangle = B^+(\langle \varphi_k, \ldots, \varphi_1 \rangle, \langle \tau_k, \ldots, \tau_1 \rangle) \tag{5.49}$$

Here $B^+$ is defined as follows (cf. [FGM+07, EWZ08]):

$$B^+(\langle \varphi_1 \rangle, \langle \tau_1 \rangle) = \langle \varphi_1 \wedge \tau_1, \varphi_1 \oplus \tau_1 \rangle \tag{5.50}$$

$$B^+(\langle \varphi_k, \ldots, \varphi_1 \rangle, \langle \tau_k, \ldots, \tau_1 \rangle) = \langle \psi,\ \kappa,\ \theta_{k-1},\ \ldots,\ \theta_1 \rangle \tag{5.51}$$

$$\text{where } k \geq 2, \quad \psi = (\varphi_k \wedge \tau_k) \vee (\varphi_k \wedge \theta_k) \vee (\tau_k \wedge \theta_k), \tag{5.52}$$

$$\kappa = \varphi_k \oplus \tau_k \oplus \theta_k, \tag{5.53}$$

$$\text{and} \quad B^+(\langle \varphi_{k-1}, \ldots, \varphi_1 \rangle, \langle \tau_{k-1}, \ldots, \tau_1 \rangle) = \langle \theta_k,\ \ldots,\ \theta_1 \rangle \tag{5.54}$$

To encode, e.g., the arctic expression $a \otimes 1$, we thus construct the following auxiliary formula tuples:

$$B^+(\langle \beta_1 \rangle, \langle 1 \rangle) = \langle \beta_1 \wedge 1, \beta_1 \oplus 1 \rangle \tag{5.55}$$

$$= \langle \beta_1, \neg\beta_1 \rangle \tag{5.56}$$

$$B^+(\langle \beta_2, \beta_1 \rangle, \langle 0, 1 \rangle) = \langle (\beta_2 \wedge 0) \vee (\beta_2 \wedge \beta_1) \vee (0 \wedge \beta_1), \beta_2 \oplus 0 \oplus \beta_1, \neg\beta_1 \rangle \tag{5.57}$$

$$= \langle \beta_2 \wedge \beta_1, \beta_2 \oplus \beta_1, \neg\beta_1 \rangle \tag{5.58}$$

Hence, we get:[55]

$$||a \otimes 1||_{bin} \tag{5.59}$$

$$= \langle \beta_2 \wedge \beta_1 \wedge \neg(a_0 \vee 0), (\beta_2 \oplus \beta_1) \wedge \neg(a_0 \vee 0), \neg\beta_1 \wedge \neg(a_0 \vee 0)\ ;\ a_0 \vee 0 \rangle \tag{5.60}$$

$$= \langle \beta_2 \wedge \beta_1 \wedge \neg a_0, (\beta_2 \oplus \beta_1) \wedge \neg a_0, \neg\beta_1 \wedge \neg a_0\ ;\ a_0 \rangle \tag{5.61}$$

$$= \langle \beta_2 \wedge \beta_1, (\beta_2 \oplus \beta_1) \wedge \neg a_0, \neg\beta_1 \wedge \neg a_0\ ;\ a_0 \rangle \tag{5.62}$$

[55] Note for the last equality that $\beta_1 \wedge \neg a_0 = a_1 \wedge \neg a_0 \wedge \neg a_0 = a_1 \wedge \neg a_0 = \beta_1$.

Thus, for instance a constraint

$$a \otimes 1 \gg 2 \oplus a \tag{5.63}$$

is then encoded in binary representation as follows:

$$||a \otimes 1 \gg 2 \oplus a||_{bin} \tag{5.64}$$

$$\begin{aligned} &= 0 \vee (\neg a_0 \wedge B^{>}(\langle \beta_2 \wedge \beta_1, (\beta_2 \oplus \beta_1) \wedge \neg a_0, \neg\beta_1 \wedge \neg a_0\rangle, \\ &\qquad \langle 0, (a_0 \vee \neg\beta_2)\ ?\ 1 : \beta_2,\ (a_0 \vee \neg\beta_2)\ ?\ 0 : \beta_1\rangle)) \end{aligned} \tag{5.65}$$

$$\begin{aligned} &= \neg a_0 \wedge B^{>}(\langle \beta_2 \wedge \beta_1, (\beta_2 \oplus \beta_1) \wedge \neg a_0, \neg\beta_1 \wedge \neg a_0\rangle, \\ &\qquad \langle 0, (a_0 \vee \neg\beta_2)\ ?\ 1 : \beta_2,\ (a_0 \vee \neg\beta_2)\ ?\ 0 : \beta_1\rangle) \end{aligned} \tag{5.66}$$

$$\begin{aligned} &= \neg a_0 \wedge ((\beta_2 \wedge \beta_1 \wedge \neg 0) \vee ((\beta_2 \wedge \beta_1 \leftrightarrow 0) \wedge \\ &\quad B^{>}(\langle (\beta_2 \oplus \beta_1) \wedge \neg a_0, \neg\beta_1 \wedge \neg a_0\rangle, \langle (a_0 \vee \neg\beta_2)\ ?\ 1 : \beta_2,\ (a_0 \vee \neg\beta_2)\ ?\ 0 : \beta_1\rangle))) \end{aligned} \tag{5.67}$$

$$\begin{aligned} &= \neg a_0 \wedge ((\beta_2 \wedge \beta_1) \vee (\neg(\beta_2 \wedge \beta_1) \wedge \\ &\quad B^{>}(\langle (\beta_2 \oplus \beta_1) \wedge \neg a_0, \neg\beta_1 \wedge \neg a_0\rangle, \langle (a_0 \vee \neg\beta_2)\ ?\ 1 : \beta_2,\ (a_0 \vee \neg\beta_2)\ ?\ 0 : \beta_1\rangle))) \end{aligned} \tag{5.68}$$

$$\begin{aligned} &= \neg a_0 \wedge ((\beta_2 \wedge \beta_1) \vee (\neg(\beta_2 \wedge \beta_1) \wedge \\ &\qquad (((\beta_2 \oplus \beta_1) \wedge \neg a_0 \wedge \neg((a_0 \vee \neg\beta_2)\ ?\ 1 : \beta_2)) \vee \\ &\qquad\quad ((\beta_2 \oplus \beta_1) \wedge \neg a_0 \leftrightarrow ((a_0 \vee \neg\beta_2)\ ?\ 1 : \beta_2)) \wedge \\ &\qquad\qquad B^{>}(\langle \neg\beta_1 \wedge \neg a_0\rangle, \langle (a_0 \vee \neg\beta_2)\ ?\ 0 : \beta_1\rangle))))) \end{aligned} \tag{5.69}$$

$$\begin{aligned} &= \neg a_0 \wedge ((\beta_2 \wedge \beta_1) \vee (\neg(\beta_2 \wedge \beta_1) \wedge \\ &\qquad (((\beta_2 \oplus \beta_1) \wedge \neg a_0 \wedge \neg((a_0 \vee \neg\beta_2)\ ?\ 1 : \beta_2)) \vee \\ &\qquad\quad ((\beta_2 \oplus \beta_1) \wedge \neg a_0 \leftrightarrow ((a_0 \vee \neg\beta_2)\ ?\ 1 : \beta_2)) \wedge \\ &\qquad\qquad \neg\beta_1 \wedge \neg a_0 \wedge \neg((a_0 \vee \neg\beta_2)\ ?\ 0 : \beta_1)))) \end{aligned} \tag{5.70}$$

This example shows that while the resulting bit tuples are short (here, the longest bit tuple $||a \otimes 1||_{bin}$ has length 4), the formulas needed even for simple constraints quickly become rather complex. Note also that since this encoding relies on binary arithmetic, the Boolean $\oplus$-connective is needed to encode the underlying adder circuits for the arctic multiplication "$\otimes$". It is frequently remarked that the presence of *parity constraints* as implemented by the $\oplus$-connective tends to degrade performance of CDCL SAT solvers (cf., e.g., [ES06]). This motivates to investigate other encodings of arithmetic on natural numbers.

In [KW08, KW09] the authors point out that one can encode both "$\oplus$" and "$\otimes$" with a binary representation by linear-size formulas. As we shall see in the next section, also a *unary* representation comes within reach.

## 5.3 A Unary SAT Encoding for Arctic Constraints

Binary SAT encodings of arithmetic are convenient because of their compactness. However, size is not the only metric to regard for SAT encodings. Instead, we want to produce a SAT encoding which minimizes the *runtime* of the SAT solver.[56]

While at first glance seemingly paradoxical, one can often observe that an (asymptotically) exponentially larger SAT encoding leads to notably better runtimes (cf., e.g., [GLP06]). For instance, this is the case for *unary* representations of numbers using *order encoding* [CB94, TTKB09]. The reason is that such encodings can be beneficial for the performance of current state-of-the-art algorithms for SAT solving that are improvements of the *DPLL algorithm* [DP60, DLL62]. Such *conflict-driven clause-learning (CDCL)* [MLM09] algorithms generally benefit from encodings which facilitate *unit propagation*.

Of course, this requires that the values that need to be represented by the SAT encoding are still "sufficiently small". This means that the improved propagation properties of the encoding are not outweighed by its sheer size, which in the end of course also does have an impact on the performance of the SAT solver. Fortunately, experience in termination analysis of term rewriting shows that in practice it often suffices to consider small maximum parameter values for parametric interpretations. For instance, in [FGM+07] we observe that for linear polynomial interpretations in the DP setting, with a timeout of 10 minutes no additional termination proofs are found on the TPDB version 3.2 if the maximum value for the parametric coefficients is increased from 6 to 63.

### Order Encoding for Arctic Constraints

As in the previous section, we wish to obtain a finite SAT encoding. Thus, to render the search space finite, we again fix a value $\ell \in \mathbb{N}$ and only consider interpretations which assign values from $\{-\infty, 0, \ldots, \ell\}$ to arctic variables.

First, let us consider the case of finite values, i.e., values from $\{0, \ldots, \ell\}$. To represent such a(n unknown) value $a$ in order encoding (cf., e.g., [CB94, TTKB09, MCLS11]), we use $\ell + 1$ propositional variables and obtain a bit tuple $\langle a_\ell, \ldots, a_1 \rangle$. We then get the value of $a$ from $a_\ell + \ldots + a_1$, so here each position has the same significance. To impose a unique representation for a value by a bit tuple of a given length, order encoding uses the invariant that $a_\ell \leq a_{\ell-1} \leq \ldots \leq a_2 \leq a_1$ must hold. Thus, a number $n$ is represented by a bit tuple of the shape $0^*1^n$, i.e., a bit tuple $\langle 0, \ldots, 0, \underbrace{1, \ldots, 1}_{n \text{ times}} \rangle$. To ensure the desired invariant for the representation of $a$, we additionally impose the *global constraints* $a_\ell \to a_{\ell-1}$, $\ldots$, $a_2 \to a_1$ (which must be satisfied along with the remaining SAT encoding). By this invariant, if the variable $a_i$ is set, we have that $a \leq i$ holds.

[56] More precisely, the goal is to obtain a SAT encoding where the runtime of the whole *toolchain* of SAT encoder and SAT solver is as small as possible (otherwise precomputing a solution as a conjunction of literals as part of the "encoding" process would be considered "optimal").

Lifting order encoding of atomic expressions to arctic values (including $-\infty$) now works similar to the binary setting. Using order encoding, we encode an arctic variable which may take values from the set $\{-\infty, 0, \ldots, \ell\}$ to a bit tuple of $\ell + 2$ elements. So for an arctic variable $a$, in a first encoding we get $\langle a_\ell, \ldots, a_1 \,;\, a_0\rangle$, imposing the global constraints $a_\ell \to a_{\ell-1}, \ldots, a_2 \to a_1$. As before, if $a_0$ is set, the encoded value is $-\infty$ (we call $a_0$ again the *infinity bit*). Otherwise, the (finite) value of $a$ is given by $a_\ell + \ldots + a_2 + a_1$. We follow the suggestion by Koprowski and Waldmann that a set infinity bit should imply that all other bits in the bit tuple are set to zero, which is helpful to break symmetries in the search problem.

This way, we obtain the following representation for arctic variables in our unary encoding $||\cdot||_{un}$:

$$||a||_{un} = \langle a_\ell \wedge \neg a_0,\ \ldots,\ a_1 \wedge \neg a_0 \,;\, a_0\rangle \tag{5.71}$$

At the same time, we need to enforce the invariant of order encoding that $(a_i \wedge \neg a_0) \leq (a_{i-1} \wedge \neg a_0)$ must hold for all $2 \leq i \leq \ell$. To achieve this, it suffices to require the following global constraints: $a_\ell \to a_{\ell-1}, \ldots, a_2 \to a_1$.

Consider again the arctic constraint (5.63), i.e., $a \otimes 1 \gg 2 \oplus a$ where $a$ may take values from the set $\{-\infty, 0, 1, 2, 3\}$. We now illustrate step by step how to represent this constraint using our unary SAT encoding. As in the previous section, we write $\beta_i$ to denote $a_i \wedge \neg a_0$ for $i \geq 1$. So we get $||a||_{un} = \langle \beta_3, \beta_2, \beta_1 \,;\, a_0\rangle$ where we additionally need to impose the global constraints $a_3 \to a_2$ and $a_2 \to a_1$ if the bit tuple $||a||_{un}$ is used in some formula.

Correspondingly, we obtain the unary encoding for arctic constants as follows:

$$||-\infty||_{un} = \langle 0 \,;\, 1\rangle \tag{5.72}$$

$$||0||_{un} = \langle 0 \,;\, 0\rangle \tag{5.73}$$

$$||n||_{un} = \langle \underbrace{1,\ \ldots,\ 1}_{n \text{ times}} \,;\, 0\rangle \qquad \text{for } n > 0 \tag{5.74}$$

For instance, the constant 2 is represented by $||2||_{un} = \langle 1, 1 \,;\, 0\rangle$.

In the following encodings of comparisons and operations for arctic expressions, let $||p||_{un} = \langle \varphi_k,\ \ldots,\ \varphi_1 \,;\, \varphi_0\rangle$ and $||q||_{un} = \langle \tau_k,\ \ldots,\ \tau_1 \,;\, \tau_0\rangle$. Again, we use 0-padding (on the left) together with Boolean simplifications where necessary so that we can assume equal length of argument tuples.

Perhaps a bit surprisingly, for comparisons it turns out to be beneficial to reuse the corresponding encodings for the binary case and to modify only the encodings of arctic

expressions (we reuse $B^{>}$ from (5.38) and (5.39)):

$$||p \gg q||_{un} = \tau_0 \vee (\neg\varphi_0 \wedge B^{>}(\langle\varphi_k, \ldots, \varphi_1\rangle, \langle\tau_k, \ldots, \tau_1\rangle)) \quad (5.75)$$

$$||p = q||_{un} = (\varphi_1 \leftrightarrow \tau_1) \wedge \ldots \wedge (\varphi_k \leftrightarrow \tau_k) \quad (5.76)$$

$$||p \geq q||_{un} = ||p \gg q||_{un} \vee ||p = q||_{un} \quad (5.77)$$

Alternatively, one could represent $p \gg q$ by $\tau_0 \vee (\neg\varphi_0 \wedge ((\varphi_1 \wedge \neg\tau_1) \vee \ldots \vee (\varphi_k \wedge \neg\tau_k)))$. However, experiments show that this representation leads to a (small) degradation in performance.

Encoding the arctic addition operation "$\oplus$" (i.e., the maximum function) is straightforward in the order encoding representation. The infinity bit of the result is set if the infinity bits of both inputs are set ($p \oplus q = -\infty$ holds iff $p = q = -\infty$). A bit in the finite part of the result is set if the corresponding bit in one of the inputs is set (and if the infinity bit is not set).

$$||p \oplus q||_{un} = \langle(\varphi_k \vee \tau_k) \wedge \neg\eta, \ldots, (\varphi_1 \vee \tau_1) \wedge \neg\eta \;;\; \eta\rangle \quad (5.78)$$

$$\text{where} \quad \eta = \varphi_0 \wedge \tau_0 \quad (5.79)$$

Note that this encoding preserves the property that for $||p \oplus q||_{un} = \langle\xi_k, \ldots, \xi_1\rangle$, we have $\xi_i \leq \xi_{i-1}$ for all $2 \leq i \leq k$.

Consider again the expression $2 \oplus a$. We get the following unary encoding:

$$||2 \oplus a||_{un} \quad (5.80)$$

$$= \langle(0 \vee \beta_3) \wedge \neg(0 \wedge a_0),\ (1 \vee \beta_2) \wedge \neg(0 \wedge a_0),\ (1 \vee \beta_1) \wedge \neg(0 \wedge a_0) \;;\; 0 \wedge a_0\rangle \quad (5.81)$$

$$= \langle\beta_3,\ 1,\ 1 \;;\; 0\rangle \quad (5.82)$$

The bit tuple obtained here with the unary encoding has length 4, whereas the bit tuple obtained in (5.47) with the binary encoding only has length 3. However, the structure of the formulas in the bit tuples is much simpler for the unary encoding since we do not need to construct a dedicated encoding for comparison.

For arctic multiplication "$\otimes$", we use an encoding which exhaustively enumerates the possible situations where an output bit gets set.

$$||p \otimes q||_{un} = \langle\xi_{2k} \wedge \neg\eta, \ldots, \xi_1 \wedge \neg\eta \;;\; \eta\rangle \quad (5.83)$$

$$\text{where} \quad \eta = \varphi_0 \vee \tau_0 \quad (5.84)$$

$$\xi_1 = \varphi_1 \vee \tau_1 \quad (5.85)$$

$$\xi_2 = \varphi_2 \vee (\varphi_1 \wedge \tau_1) \vee \tau_2 \quad (5.86)$$

$$\xi_3 \quad = \quad \varphi_3 \ \vee \ (\varphi_2 \wedge \tau_1) \ \vee \ (\varphi_1 \wedge \tau_2) \ \vee \ \tau_3 \tag{5.87}$$

$$\dots$$

$$\xi_i \quad = \quad \varphi_i \ \vee \bigvee_{\substack{c,d \geq 1 \\ c+d=i}} (\varphi_c \wedge \tau_d) \ \vee \ \tau_i \tag{5.88}$$

$$\dots$$

$$\xi_{2k} \quad = \quad \varphi_k \wedge \tau_k \tag{5.89}$$

Here we again use 0-padding and perform Boolean simplifications to ease readability (e.g., $\varphi_j = 0$ if $j > k$), and our implementation omits leading 0s. The intuition behind this encoding is the following: For the infinity bit of the result, the reasoning is as for the binary encoding—if $p$ or $q$ take the value $-\infty$, then also $p \otimes q$ becomes $-\infty$. Now let us consider the case where $p$ and $q$ take finite values from $\mathbb{N}$. Recall that $\varphi_c$ is true iff $p \geq c$ holds and that $\tau_d$ is true iff $q \geq d$ holds. Likewise, $\xi_i$ is supposed to become true iff $p \otimes q \geq i$ holds. To achieve this, we use that $p \otimes q \geq i$ holds (for finite $i$) iff there exist $c, d \in \mathbb{N}$ with $i = c + d$, $p \geq c$, and $q \geq d$. For this, we encode $\xi_i$ as the disjunction of all (at most $i + 1$ many) combinations of $\varphi_c$ and $\tau_d$ such that $i = c + d$.

For example, this encoding represents the expression $a \otimes 1$ as follows:

$$||a \otimes 1||_{un} \tag{5.90}$$

$$= \langle \beta_3 \wedge \neg(a_0 \vee 0), (\beta_3 \vee (\beta_2 \wedge 1)) \wedge \neg(a_0 \vee 0), (\beta_2 \vee (\beta_1 \wedge 1)) \wedge \neg(a_0 \vee 0), (\beta_1 \vee 1) \wedge \neg(a_0 \vee 0) \ ; \ (a_0 \vee 0) \rangle \tag{5.91}$$

$$= \langle \beta_3, (\beta_3 \vee \beta_2) \wedge \neg a_0, (\beta_2 \vee \beta_1) \wedge \neg a_0, \neg a_0 \ ; \ a_0 \rangle \tag{5.92}$$

Now we can revisit the arctic constraint (5.63) $a \otimes 1 \gg 2 \oplus a$. Here, our unary encoding yields:

$$||a \otimes 1 \gg 2 \oplus a||_{un} \tag{5.93}$$

$$= 0 \vee (\neg a_0 \wedge B^{>}(\langle \beta_3, (\beta_3 \vee \beta_2) \wedge \neg a_0, (\beta_2 \vee \beta_1) \wedge \neg a_0, \neg a_0 \rangle, \langle 0, \beta_3, \ 1, \ 1 \rangle)) \tag{5.94}$$

$$= \neg a_0 \wedge B^{>}(\langle \beta_3, (\beta_3 \vee \beta_2) \wedge \neg a_0, (\beta_2 \vee \beta_1) \wedge \neg a_0, \neg a_0 \rangle, \langle 0, \beta_3, \ 1, \ 1 \rangle) \tag{5.95}$$

$$= \neg a_0 \wedge ((\beta_3 \wedge \neg 0) \vee ((\beta_3 \leftrightarrow 0) \wedge B^{>}(\langle (\beta_3 \vee \beta_2) \wedge \neg a_0, (\beta_2 \vee \beta_1) \wedge \neg a_0, \neg a_0 \rangle, \langle \beta_3, \ 1, \ 1 \rangle))) \tag{5.96}$$

$$= \neg a_0 \wedge (\beta_3 \vee (\neg \beta_3 \wedge B^{>}(\langle (\beta_3 \vee \beta_2) \wedge \neg a_0, (\beta_2 \vee \beta_1) \wedge \neg a_0, \neg a_0 \rangle, \langle \beta_3, \ 1, \ 1 \rangle))) \tag{5.97}$$

$$= \neg a_0 \wedge (\beta_3 \vee (\neg \beta_3 \wedge (((\beta_3 \vee \beta_2) \wedge \neg a_0 \wedge \neg \beta_3) \vee (((\beta_3 \vee \beta_2) \wedge \neg a_0 \leftrightarrow \neg \beta_3) \wedge B^{>}(\langle (\beta_2 \vee \beta_1) \wedge \neg a_0, \neg a_0 \rangle, \langle 1, \ 1 \rangle)))))) \tag{5.98}$$

$$= \neg a_0 \wedge (\beta_3 \vee (\neg\beta_3 \wedge (((\beta_3 \vee \beta_2) \wedge \neg a_0 \wedge \neg\beta_3) \vee (((\beta_3 \vee \beta_2) \wedge \neg a_0 \leftrightarrow \neg\beta_3) \wedge (((\beta_2 \vee \beta_1) \wedge \neg a_0 \wedge \neg 1) \vee (((\beta_2 \vee \beta_1) \wedge \neg a_0 \leftrightarrow 1) \wedge B^{>}(\langle\neg a_0\rangle, \langle 1\rangle)))))))) \tag{5.99}$$

$$= \neg a_0 \wedge (\beta_3 \vee (\neg\beta_3 \wedge (((\beta_3 \vee \beta_2) \wedge \neg a_0 \wedge \neg\beta_3) \vee (((\beta_3 \vee \beta_2) \wedge \neg a_0 \leftrightarrow \neg\beta_3) \wedge (((\beta_2 \vee \beta_1) \wedge \neg a_0) \wedge B^{>}(\langle\neg a_0\rangle, \langle 1\rangle)))))) \tag{5.100}$$

$$= \neg a_0 \wedge (\beta_3 \vee (\neg\beta_3 \wedge (((\beta_3 \vee \beta_2) \wedge \neg a_0 \wedge \neg\beta_3) \vee (((\beta_3 \vee \beta_2) \wedge \neg a_0 \leftrightarrow \neg\beta_3) \wedge (((\beta_2 \vee \beta_1) \wedge \neg a_0) \wedge \neg a_0 \wedge \neg 1))))) \tag{5.101}$$

$$= \neg a_0 \wedge (\beta_3 \vee (\neg\beta_3 \wedge ((\beta_3 \vee \beta_2) \wedge \neg a_0 \wedge \neg\beta_3))) \tag{5.102}$$

Due to the Boolean simplifications, here we obtain a formula with no $\oplus$-connectives and a rather simple Boolean structure in the end (the simplifications in the example can be automated at very little cost). In this example, the unary encoding yields the bit tuple $||a \otimes 1||_{un}$ with length 5 as the longest bit tuple, whereas the binary encoding yields the only slightly shorter bit tuple $||a \otimes 1||_{bin}$ with length 4 as the longest bit tuple.

## 5.4 Related Work

### Arctic Interpretations

In [Ama05] Amadio proposes arctic interpretations (with non-negative rational numbers instead of natural numbers) for analysis of space complexity of a simply-typed functional language via quasi-interpretations. In 2007, Waldmann uses arctic interpretations for termination analysis of string rewrite systems [Wal07]. Koprowski and Waldmann lift the approach to TRSs in [KW08, KW09] and also allow values below zero as parts of the interpretation. These papers provide the setting on which this chapter is based. In 2010, Sternagel and Thiemann [ST10] generalize both standard arctic interpretations and arctic interpretations below zero to a uniform setting. Our contributions to solving arctic constraints are orthogonal to this recent improvement.

### SAT Encodings for Arctic Constraints

In [Wal07], Waldmann proposes a binary SAT encoding for *tropical* constraints. This encoding is similar to the encoding for arctic constraints of [KW08, KW09]. Moreover, [Wal07] discusses both a binary and a unary (order) SAT encoding for *fuzzy algebras*, where (only) the max- and the min-functions are used for interpretations in weakly monotonic algebras. For this setting a unary encoding is much more straightforward than for arctic algebras since the $\otimes$-operation does not need to be encoded in unary. That attaining suitable unary representations in the arctic setting is more difficult is also confirmed by

the later follow-up works [KW08, KW09], where Koprowski and Waldmann still suggest a *binary* SAT encoding for arctic constraints.

Independent of the present work, [HKW10] proposes using *sorting networks* to encode the $\otimes$-operation for tropical algebras. Tropical algebras are dual to arctic algebras since one considers "min" and $+\infty$ instead of "max" and $-\infty$, but the $\otimes$-operation is essentially the same.

### Order Encoding

Concerning related applications of order encoding, an early work is [CB94], where order encoding is applied to solve scheduling problems. Independently from our work, in [TTKB09] Tamura *et al.* apply order encoding to tackle linear finite constraint satisfaction problems via SAT solving. In a preprocessing step, they propose a conversion of linear arithmetic constraints over a finite domain to a conjunctive normal form of comparisons of the shape $x \leq c$ (here $x$ is a variable and $c \in \mathbb{Z}$). This conversion is closely related to our encoding of the $\otimes$-operation. In [MCLS11], Metodi *et al.* propose a *SAT compiler* translating from a high-level problem description to an optimized CNF representation via order encoding. They successfully apply their system on several benchmark suites from the area of constraint satisfaction problems.

## 5.5 Experiments

We have implemented both the binary encoding and the unary encoding described in the chapter in the termination prover AProVE [GST06]. Here we have also used the optimization described for standard matrix orders in [EWZ08] that for the root symbols of the dependency pairs, it suffices to use values different from the zero element of the semi-ring for the matrix and vector entries in the first row. Here, this means that all other entries are filled with $-\infty$. This optimization is complete if the root symbols of dependency pairs do not occur anywhere else, which is virtually always the case in practice (and we only apply the optimization if this premise holds).

It is worth noting that [KW08, KW09] use the *same* bitwidth both for arctic variables and also for arctic intermediate expressions. In contrast, we only impose a bound on the bitwidth for *arctic variables*, but we do not do so for the intermediate expressions.

Note that a setting where the bitwidth is bounded for intermediate expressions is most likely more favorable for unary encodings than for binary encodings, provided that in both cases one uses the same *search space* $\{-\infty, 0, \ldots, n\}$ for the arctic variables. This is opposed to using the same *bitwidth*, which would mean that in $k$ bits for the bit tuple, one could represent only values up to $k-1$ in unary, whereas exponentially greater values $2^{k-1}-1$ could still be represented in binary.

The main disadvantage of the unary encoding is that it could lead to exponentially longer bit tuples than its binary counter-part, e.g., in case of deeply nested terms. With a fixed bitwidth, this is effectively prevented.

For greater flexibility, in all settings we used the previously described approach of shifting negative numbers when searching for below-zero interpretations instead of introducing an additional sign bit for two's complement.

To investigate the impact of our unary encoding in practice, we conducted several experiments using our implementation of the encodings described in this chapter in the automated termination prover AProVE [GST06]. As SAT solver we use version 2 of MiniSAT [ES04], and for conversion of propositional formulas to CNF we use SAT4J [LP10].

## Direct Comparison

Similar to the experiments of Chapter 3, we configured AProVE to use only a basic version of the DP method, i.e., we applied the DP transformation to the initial DP problem, followed by repeated applications of the dependency graph processor and the reduction pair processor with arctic interpretations. For automation of the reduction pair processors, here we experimented with arctic interpretations above and below zero both in the classic binary and in our novel unary setting. As parameters of the explored search space, we used different values for the *dimension* of the matrices and also for the *minimum* and the *maximum* finite arctic numbers in the search space (using Theorem 5.2 if this minimum was $\geq 0$, and Theorem 5.4, otherwise). We always include $-\infty$ in the search space for the arctic variables. We report on our results for the unary and for the binary setting in different combinations of settings.

As benchmark set, here we used the 1316 SRSs considered for the category *SRS Standard* of the Termination Competition 2011 (TPDB version 8.0.1). As discussed in Section 5.1, arctic interpretations are especially well suited for rewrite systems where function symbols have only one argument, i.e., SRSs, which motivates this choice.

The results of these runs on an Intel Xeon 5140 at 2.33 GHz are reported in Figure 5.6. Here the configuration AProVE binary uses the binary SAT encoding described in Section 5.2, and the configuration AProVE unary uses the novel unary SAT encoding described in Section 5.3. The columns "*d*", "*min*", and "*max*" describe the used dimension, the minimum value and the maximum value for finite arctic numbers, respectively. The column "Yes" states the number of termination proofs found within a timeout of 60 seconds (as in the Termination Competition), and the column "TO" indicates the number of examples on which runtime exceeds 60 seconds and which thus time out. Since the TPDB contains many very challenging SRSs, we observe several hundreds of timeouts. Thus, here we do not focus on total runtime on all examples, but in the column "Avg. Runtime(Yes)" we give the measured average runtime in seconds for the cases where a

| | | | AProVE unary | | | AProVE binary | | |
|---|---|---|---|---|---|---|---|---|
| $d$ | $min$ | $max$ | Yes | TO | Avg. Runtime(Yes) | Yes | TO | Avg. Runtime(Yes) |
| 1 | 0 | 1 | **61** | 278 | 2.95 | **61** | **277** | **2.94** |
| 1 | 0 | 2 | **70** | 319 | 3.30 | **70** | **302** | **3.10** |
| 1 | 0 | 3 | 81 | 385 | 4.24 | **82** | **306** | **3.58** |
| 1 | 0 | 4 | 83 | 426 | 5.57 | **85** | **337** | **4.34** |
| 1 | 0 | 5 | 83 | 461 | 5.90 | **87** | **364** | **5.34** |
| 1 | -1 | 1 | 91 | 325 | 5.62 | **92** | **308** | **5.20** |
| 1 | -1 | 2 | 108 | 399 | 5.74 | **115** | **327** | **5.14** |
| 1 | -1 | 3 | 118 | 446 | 6.27 | **128** | **371** | **6.05** |
| 1 | -3 | 4 | 106 | 537 | **6.86** | **129** | **420** | 7.47 |
| 2 | 0 | 1 | **237** | **544** | 6.23 | 215 | 567 | **4.74** |
| 2 | 0 | 5 | 226 | **638** | 4.53 | **229** | **638** | **4.29** |
| 2 | -1 | 3 | **269** | **624** | **4.10** | 266 | 641 | 4.23 |
| 3 | 0 | 1 | **290** | **620** | **4.02** | 276 | 672 | 4.68 |
| 3 | 0 | 2 | **319** | **739** | **5.74** | 294 | 844 | 7.71 |
| 3 | 0 | 3 | **311** | **882** | **7.45** | 299 | 904 | 8.53 |
| 3 | 0 | 4 | **288** | **973** | **7.89** | 272 | 987 | 9.80 |
| 3 | -1 | 2 | **316** | **934** | **7.26** | 292 | 962 | 9.15 |
| 3 | -1 | 3 | **305** | **992** | **8.08** | 262 | 1040 | 9.85 |
| 3 | -2 | 2 | **301** | **980** | **8.20** | 248 | 1039 | 9.53 |
| 3 | -2 | 3 | **292** | **1020** | **8.84** | 250 | 1061 | 10.29 |
| 4 | 0 | 1 | **294** | **874** | **5.24** | 267 | 971 | 9.39 |
| 4 | 0 | 2 | **285** | **1022** | **8.10** | 239 | 1077 | 14.14 |
| 4 | -1 | 1 | **287** | **968** | **8.73** | 217 | 1074 | 14.57 |
| 5 | 0 | 1 | **267** | **1009** | **8.04** | 216 | 1084 | 13.83 |
| 5 | -1 | 3 | **207** | **1109** | **15.61** | 75 | 1241 | 23.41 |
| 7 | 0 | 1 | **204** | **1102** | **14.28** | 86 | 1227 | 20.82 |
| 7 | -1 | 2 | **116** | **1200** | 24.24 | 28 | 1288 | **12.92** |

Figure 5.6: Unary and Binary Arctic SAT Encodings for Different Dimensions

termination proof was found by the respective prover (i.e., we only consider the "Yes"-examples and give average times).[57] In each row, we highlight the respective best result in **bold face**.

These measurements allow for the following observations and conclusions:

- The precision of termination analysis via arctic matrix interpretations improves significantly if one considers matrices of dimension $> 1$. The overall most successful configuration in these experiments (dimension 3, minimum value 0, maximum value 2, unary encoding) yields 319 termination proofs, whereas with matrices of dimension 1 we only obtain 129 termination proofs in the most successful measured setting

[57] Note that in general this measurement is favorable to tools which time out on hard examples instead of finding a proof after a relatively high runtime. Thus, one should always take also the number of successful termination proofs of a given configuration into account when drawing conclusions from the average runtime.

(min. value $-3$, max. value 4, binary encoding).

This general observation is not surprising since also conventional matrix interpretations [EWZ08] benefit from dimensions greater than 1 (cf., e.g., [ZM10]).

- For dimension 1, the results of AProVE binary are slightly better than those of AProVE unary, both w.r.t. number of successful proofs and w.r.t. timeouts and average runtime. For instance, with dimension 1 the most successful configuration of AProVE binary (min. value $-3$, max. value 4) discovers 129 termination proofs, whereas the most successful configuration of AProVE unary (min. value $-1$, max. value 3) in our experiments only finds 118 termination proofs, i.e., approx. 8.5 % less proofs.

  The effect becomes more noticeable as the size of the search space per variable (given by $|\{-\infty, min, \ldots, max-1, max\}| = max - min + 2$) increases. For instance, for the search space with min. value 0 and max. value 1, we have almost identical results of the two configurations, whereas for the search space with min. value $-3$ and max. value 4 AProVE binary succeeds on 129 SRSs, whereas AProVE unary succeeds only on 106 SRSs, i.e., on approx. 17.8 % less examples.

  A likely reason is that for dimension 1, the asymptotic size increase of the unary encoding over the binary encoding still has more impact on the runtimes than potentially beneficial aspects of the unary encoding for propagation inside the SAT solver. Since in this case (arctic) matrix multiplication corresponds to (arctic) scalar multiplication, the structure of the arithmetic operations that need to be encoded is relatively simple.

- The more the dimension $d$ of the used matrices increases, the better are the results of AProVE unary compared to those of AProVE binary. The effect is noticeable already for dimension 2, where with a min. value of 0 and a max. value of 1, AProVE binary only finds 215 termination proofs within 60 seconds, whereas AProVE unary finds 237 termination proofs, i.e., 10.2 % more termination proofs. For the notably greater search space per variable with min. value of 0 and a max. value of 5, however, AProVE binary discovers 229 termination proofs, whereas AProVE unary discovers only 226 termination proofs, i.e., 1.3 % less.

  For the (high) dimension of 7, with a min. value of 0 and a max. value of 1, AProVE binary only finds 86 termination proofs within 60 seconds, whereas AProVE unary still finds 204 termination proofs, i.e., over 137 % more.

- Among the considered configurations, the most successful variant of AProVE binary (dimension 3, min. value 0, max. value 3) discovers only 299 termination proofs, whereas the most successful variant of AProVE unary (dimension 3, min. value 0, max. value 2) discovers 319 termination proofs, i.e., 20 proofs or almost 6.7 % more.

A likely explanation for this point and also for the previous one is that for higher matrix dimensions, the unary encoding benefits both from the simplicity of the representation of the $\oplus$-operation (i.e., arctic addition) and from the absence of the $\oplus$-connective (i.e., Boolean XOR) in the representation of the $\otimes$-operation (i.e., arctic multiplication). These operations need to be encoded especially often if the matrix dimension is high, which consequently also leads to higher complexity of (arctic) matrix multiplication. In contrast, the impact of the more verbose representation of intermediate values by unary bit tuples instead of binary bit tuples seems to be less pronounced.

## Effects on a Full Termination Proving Strategy

While arctic interpretations do provide a powerful termination technique already in a stand-alone setting, it is desirable to embed them into the overall strategy of a termination tool which orchestrates many termination techniques (e.g., in the DP framework [GTS05a]). In this setting, it is especially important to provide a termination proof *step* within a short period of time to ensure progress of the overall termination proof, which can then possibly be completed by entirely different techniques. In case such a termination proof step is not possible with the configuration of the termination technique at hand, it is desirable to detect this early and leave more resources to other techniques for proving (non-)termination. So even if termination of an example cannot be shown solely based on arctic interpretations, it is nevertheless advantageous to automate them efficiently.

Thus, to assess the impact of the unary encoding for arctic interpretations on a fully-fledged termination tool such as AProVE, which combines many termination proving techniques, we ran the competition versions of AProVE of the years 2009, 2010, and 2011 in two setups. In the first setup AProVE TC original unary[58] we used our original strategies from the respective competitions, where the SAT-based search for arctic matrices was conducted using our unary encoding. For the second setup AProVE TC binary, we modified these strategies so that we would encode the search for arctic interpretations not in unary, but in binary.

As benchmark set, here we again used the 1316 SRSs considered for the category *SRS Standard* of the Termination Competition 2011 (TPDB version 8.0.1). This is a superset of the SRSs actually used at the competitions up to 2011 (to keep runtimes manageable and also to increase excitement, since 2009 only a randomly chosen subset of the TPDB examples is used in the competitions). The results of these runs with a timeout of 60 seconds on an Intel Xeon 5140 at 2.33 GHz are reported in Figure 5.7.

As the figure shows, for all the competition versions of AProVE from 2009 – 2011, the precision of termination analysis constantly increases by at least 17 examples on the SRSs

[58] Here "TC" stands for "Termination Competition".

| | AProVE TC original unary | | | AProVE TC binary | | |
|---|---|---|---|---|---|---|
| Year | Yes | No | Yes+No | Yes | No | Yes+No |
| 2009 | 609 | 99 | 708 | 596 | 98 | 694 |
| 2010 | 609 | 98 | 707 | 596 | 98 | 694 |
| 2011 | 687 | 91 | 778 | 671 | 90 | 761 |

Figure 5.7: Competition Versions AProVE 2009 – 2011 with Unary and Binary SAT Encodings for Arctic Interpretations

| | AProVE | | | TTT2 | | | |
|---|---|---|---|---|---|---|---|
| Year | Yes | No | Yes+No | Yes | No | Yes+No | Total |
| 2009 | **149** | **36** | **185** | 145 | 20 | 165 | 214 |
| 2010 | 162 | **32** | **194** | **181** | 13 | **194** | 289 |
| 2011 | **132** | **27** | **159** | **132** | 14 | 146 | 215 |

Figure 5.8: Some Results from the Category *SRS Standard* of the Termination Competitions 2009 – 2011

of the current TPDB if one modifies only the search for arctic interpretations to use a unary SAT encoding (which we also used in the competitions) instead of a binary encoding for arctic constraints.

To show that also a seemingly small difference in successful termination proofs can make a difference for a competition setting, consider the results of the termination tools AProVE [GST06] and TTT2 [KSZM09] in the category *SRS Standard* of the Termination Competitions 2009, 2010, and 2011, given in Figure 5.8.[59] In each of these years, the tools AProVE and TTT2 reached the highest scores among all participants in this category.[60] In Figure 5.8, we again highlight the best results in **bold face**.

Note that in 2009 AProVE only found 4 more proofs of termination than TTT2, in 2010 AProVE and TTT2 reached an equal score in discovered proofs of (non-)termination, and in 2011 AProVE and TTT2 reached an equal score in found termination proofs. Together with Figure 5.7, this makes it seem quite likely that without the contributions of this chapter, AProVE would not have reached the top positions in this category. More concretely, AProVE would most likely not have found the highest number of termination proofs in 2009 and 2011, and AProVE would not have reached the (shared) first place of the competition in 2010.

[59] These results have been taken from the Termination Competition Platform at: `http://termcomp.uibk.ac.at/`

[60] Note that the numbers in Figure 5.8 are not directly comparable between *different* years since each year a different random subset of the TPDB is used to obtain the benchmarks for the competition.

### SAT Competition 2009

The impact of our novel unary encoding is also observable in the results of the SAT Competition[61] of 2009.[62] In the benchmark submission to this competition by the author of this thesis, we included also two encodings of the same proof step with arctic below zero interpretations (using $4 \times 4$-matrices with coefficients from $\{-1, 0, 1\}$) for the TRS `SRS/Endrullis/04.srs` from the TPDB version 5.0.2.[63] Before this proof step, we applied the dependency pair transformation and two steps using linear polynomial interpretations to delete dependency pairs. The corresponding (satisfiable) SAT instances for the arctic proof step are `SAT09/APPLICATIONS/aprove09/AProVE09-20.cnf` for the binary encoding and `SAT09/APPLICATIONS/aprove09/AProVE09-21.cnf` for the unary (order) encoding.[64] All 16 SAT solvers which entered the second stage of the competition in the main track proved satisfiability for `SAT09/APPLICATIONS/aprove09/AProVE09-21.cnf` in less time than for `SAT09/APPLICATIONS/aprove09/AProVE09-20.cnf` (and indeed, in all tracks, every solver performed at least as good on the instance `AProVE09-21.cnf` as on the instance `AProVE09-20.cnf`). This indicates that the beneficial properties of our encoding are *robust* among the current generation of SAT solvers, i.e., our results apply independent of the SAT solver which is used as back-end. This is especially noteworthy since the instances in question are satisfiable, and thus a SAT solver only needs to explore the search space until the first model of the SAT instance is found. Depending on the heuristics for variable selection that the solver employs, this can happen also early during the search on an otherwise structurally hard instance (i.e., the solver "can be lucky" in the search for a model).

Of course, most solvers that are successful on real-world applications are currently based on the prevailing CDCL paradigm, which is a descendant of the DPLL algorithm. Thus, it is possible that SAT solvers using entirely different approaches may show different behavior.

## 5.6 Summary and Outlook

In this chapter we have provided a novel SAT encoding for automating interpretations based on max/plus algebras, which are also known as arctic algebras. This SAT encoding

[61] See `http://www.satcompetition.org/` for more details on this international bi-annual competition of SAT solvers.

[62] The author of this thesis conducted a significant part of the research underlying this chapter of the thesis already in November and December 2008.

[63] Following a reorganization of the TPDB, since version 7.0 this TRS can be found in the TPDB as `TRS/Mixed_SRS/04.xml`.

[64] These instances were run as part of the *application category* at the SAT Competition 2009. The corresponding full benchmark suite is available at `http://www.cril.univ-artois.fr/SAT09/bench/appli.7z`. We also provide further information on our benchmarks in the booklet of the competition [Fuh09].

is based on a unary representation of numbers via *order encoding* [CB94, TTKB09]. This representation is particularly useful if the arising bit tuples are relatively small, which is indeed the case for the constraints arising from termination analysis via arctic interpretations. Additionally, we have shown that it is not necessary to apply a special SAT encoding for interpretations which contain below zero values. To achieve this, we propose to shift negative constants via arctic multiplication.

Our empirical results indicate notable performance improvements by our novel unary encoding if matrices of dimension $> 1$ are applied. These results have been confirmed both in a stand-alone setting and also embedded in the versions of AProVE used for the Termination Competitions of the years 2009 – 2011. For the competition versions, it is likely that without the contributions of this chapter, AProVE would not have reached the highest scores in the category *SRS Standard* of the respective Termination Competitions (other powerful participants like, e.g., TTT2 [KSZM09] render the *SRS Standard* category particularly competitive).

We have also submitted SAT instances based on the binary and unary encodings presented in this chapter as benchmarks to the SAT Competition 2009. The results of the SAT solvers competing on these instances indicate that a wide variety of modern SAT solvers benefits from this alternative encoding.

As far as future work is concerned, one obvious next step would be to apply our SAT encoding also in the generalized setting for arctic interpretations proposed by Sternagel and Thiemann in [ST10]. In a recent talk [HKW10], Waldmann suggests using *sorting networks* (cf. [ES06]) for encoding *tropical multiplication* (corresponding to classic addition, analogous to the setting of this chapter) in unary when performing termination analysis via tropical interpretations, i.e., using min/plus algebras (cf. [Wal07]). This approach can easily be adapted to our setting of arctic multiplication, and it would be interesting to integrate this approach with our contributions.

Finally, in [HKW10], Waldmann also mentions *mixed linear programming* (cf., e.g., [JLN+10]) as an alternative to SAT encodings for tropical interpretations, which carries over also to arctic interpretations and is thus also worth considering.

# 6 Lazy Abstraction for Size-Change Termination

Size-change termination is a widely used means of proving termination where source programs are first abstracted to size-change graphs which are then analyzed to determine if they satisfy the size-change termination property. Here, the choice of the abstraction is crucial to the success of the method, and it is an open problem how to choose an abstraction such that no critical loss of precision occurs. In this chapter we show how to couple the search for a suitable abstraction and the test for size-change termination via an encoding to a single SAT instance. In this way, the problem of choosing the right abstraction is solved *en passant* by a SAT solver. We show that for the setting of term rewriting, the integration of this approach into the dependency pair framework works quite smoothly and gives rise to a new class of *size-change* reduction pairs. We implemented size-change reduction pairs in the termination prover AProVE and evaluated their usefulness in extensive experiments.

One of the challenges of termination analysis is to design a program abstraction that captures the properties needed to prove termination as often as possible, while providing a decidable sufficient criterion for termination. The *size-change termination (SCT)* method [LJB01] is one such technique where programs are abstracted to *size-change graphs* which describe how the sizes of program data are affected by the transitions made in a computation. Size is measured by a well-founded base order. A set of size-change graphs has the *SCT property* iff for every path through any infinite concatenation of these graphs, a value would descend infinitely often w.r.t. the base order. This contradicts the well-foundedness of the base order, which implies termination of the original program. Lee *et al.* prove in [LJB01] that the problem to determine if a set of size-change graphs has the SCT property is PSPACE-complete. The size-change termination method has been successfully applied in a variety of different application areas [Ave06, CT99, JB04, PR04b, SJ05, TG05].

Another approach emerges from the term rewriting community where termination proofs are performed by identifying suitable well-founded orders on terms in the form of reduction pairs and showing that every transition in a computation leads to a reduction w.r.t. the order. This approach provides a decidable sufficient termination criterion for a given class of reduction pairs and can be considered as a program abstraction because

terms are viewed modulo the reduction pair.

A major bottleneck when applying SCT is due to the fact that it is a two-phase process: First, a suitable program abstraction must be found, and then, the resulting size-change graphs are checked for termination. It is an open problem how to choose an abstraction such that no critical loss of precision occurs. Thus, our aim is to couple the search for a suitable abstraction with the test for size-change termination. To this end we model the search for an abstraction as the search for a reduction pair. Then we can encode both the abstraction and the test for the SCT property into a single SAT instance.

Using a SAT-based search for orders to prove termination is well established by now, cf. the discussion of related work in Chapter 1. However, there is one major obstacle when using SAT for SCT. SCT is PSPACE-complete and hence (unless NP = PSPACE), there is no polynomial-size encoding of SCT to SAT. Thus, we focus on a subset of SCT which is in NP and can therefore be effectively encoded to SAT. This subset, called SCNP, is introduced by Ben-Amram and Codish in [BC08] where experimental evidence indicates that the restriction to this subset of SCT hardly makes any difference in practice. We illustrate our approach in the context of term rewrite systems (TRSs). The basic idea is to give a SAT encoding for the following question:

> For a given TRS (and a class of base orders such as polynomial orders), is there a base order such that the resulting size-change graphs have the SCT property?

In [TG05], Thiemann and Giesl also apply the SCT method to TRSs and show how to couple it with the dependency pair method [AG00]. However, they take the two-phase approach, first (manually) choosing a base order, and then checking if the induced size-change graphs satisfy the SCT property. Otherwise, one might try a different order. The implementation of [TG05] in the tool AProVE [GST06] only uses the (weak) embedding order in combination with argument filters [AG00] as base order. It performs a naive search which enumerates all argument filters. The new approach presented in this chapter leads to a significantly more powerful implementation.

Using SCNP instead of SCT has an additional benefit. SCNP can be directly simulated by a new class of orders which can be used for *reduction pairs* in the DP framework that are amenable for automation. We require only a small restriction on the input DP problems (they must have the "tuple property" [FGP+11], i.e., for a DP problem with DPs $s \to t$, $root(s)$ and $root(t)$ may only occur at the roots of DPs) which is satisfied by virtually all instances that occur in practice. Thus, the processors of the DP framework usually do not have to be modified at all for the combination with SCNP. This makes the integration of the size-change method with DPs much smoother than in [TG05] and it also allows to use this integration directly in (future) extensions of the DP framework. The orders simulating SCNP are novel in the rewriting area.

This chapter is structured as follows: Section 6.1 briefly presents the SCT method for DPs. In Section 6.2 we present an extension of the DP framework that allows us to integrate SCNP seamlessly while at the same time not requiring any changes to existing processors. Section 6.3 adapts the SCNP approach to term rewriting in the DP framework. In Section 6.4 we give a challenge example that highlights the scenarios where SCNP can be particularly useful. Section 6.5 then shows how to encode the search for a base order which satisfies the SCNP property into a single SAT problem. Section 6.6 presents our experimental evaluation using our termination tool AProVE [GST06]. We conclude in Section 6.7.

## 6.1 Size-Change Termination and Dependency Pairs

Size-change termination [LJB01] is a program abstraction where termination is decidable. As mentioned in the introduction, an abstract program is a finite set of size-change graphs which describe, in terms of size-change, the possible transitions between consecutive function calls in the original program.

Size-change termination and the DP framework have some similarities: (i) size-change graphs provide a representation of the paths from one function call to the next, and (ii) in a second stage, we show that these graphs do not allow infinite descent.

Also in the DP framework, the basic idea is (i) to describe all (finitely many) paths in the input program (i.e., input TRS) from one function call to the next by dependency pairs. Then (ii) one has to prove that these paths cannot follow each other infinitely often in a computation. So these steps correspond to steps (i) and (ii) in size-change termination.

The main difference between SCT and the DP method is the stage when the undecidable termination problem is abstracted to a decidable one. For SCT, we use a base order to obtain the finite representation of the paths by size-change graphs. For DPs, no such abstraction is performed and, indeed, the termination (i.e., finiteness) of a DP problem is undecidable. Here, the abstraction step is only the second stage where typically a decidable class of base orders is used.

The SCT method can be used with any base order. It only requires the information which arguments of a function call become (strictly or weakly) smaller w.r.t. the base order. To prove termination, the base order has to be well founded. For the adaptation to term rewriting, we will use a reduction pair $(\succsim, \succ)$ for this purpose and introduce the notion of a size-change graph directly for DP problems.

If the DP problem has a DP $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m)$, then the corresponding size-change graph has nodes $\{F_1, \ldots, F_n, G_1, \ldots, G_m\}$ representing the argument positions of $F$ and $G$. The labeled edges in the size-change graph indicate whether there is a strict or weak decrease between these arguments.

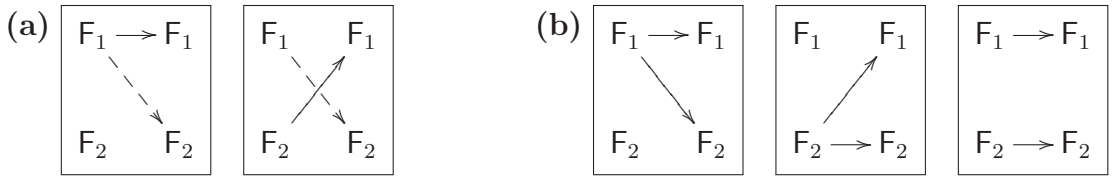


Figure 6.2: Size-change graphs from Example 6.3

**Definition 6.1** (size-change graphs)**.** *Let* $(\succsim, \succ)$ *be a (possibly non-monotonic) reduction pair on base terms, and let* $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m) \in \mathcal{P}$ *for a DP problem* $(\mathcal{P}, \mathcal{R})$*. The* size-change graph *resulting from* $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m)$ *and from* $(\succsim, \succ)$ *is the graph* $(V_s, V_t, E)$ *with source vertices* $V_s = \{F_1, \ldots, F_n\}$*, target vertices* $V_t = \{G_1, \ldots, G_m\}$*, and labeled edges* $E = \{(F_i, G_j, \succ) \mid s_i \succ t_j\} \cup \{(F_i, G_j, \succsim) \mid s_i \succsim t_j\}$*.*

Size-change graphs are depicted as in Figure 6.2. Each graph consists of source vertices, on the left, target vertices, on the right, and edges drawn as full and dashed arrows to indicate strict and weak decrease (i.e., corresponding to "$\succ$" and "$\succsim$", respectively). We introduce the main ideas underlying SCT by example.

**Example 6.3** (Showing SCT Property by Composition)**.** Consider the following TRS $\mathcal{R} = \{(6.1), (6.2)\}$.

$$\mathsf{f}(\mathsf{s}(x), y) \to \mathsf{f}(x, \mathsf{s}(x)) \tag{6.1}$$
$$\mathsf{f}(x, \mathsf{s}(y)) \to \mathsf{f}(y, x) \tag{6.2}$$

It has the DPs (6.3) and (6.4).

$$\mathsf{F}(\mathsf{s}(x), y) \to \mathsf{F}(x, \mathsf{s}(x)) \tag{6.3}$$
$$\mathsf{F}(x, \mathsf{s}(y)) \to \mathsf{F}(y, x) \tag{6.4}$$

For analysis of full termination, we thus obtain the initial DP problem $(\mathcal{P}, \mathcal{R})$ with $\mathcal{P} = \{(6.3), (6.4)\}$.

We use a reduction pair $(\succsim, \succ)$ based on the embedding order, i.e., $\mathsf{s}(x) \succ x$, $\mathsf{s}(y) \succ y$, $\mathsf{s}(x) \succsim \mathsf{s}(x)$, $\mathsf{s}(y) \succsim \mathsf{s}(y)$ hold. Then we get the size-change graphs in Figure 6.2**(a)**. Between consecutive function calls, the first argument decreases in size or becomes smaller than the original second argument. In both cases, the second argument weakly decreases compared to the original first argument. By repeated composition of the size-change graphs, we obtain the three "idempotent" graphs in Figure 6.2**(b)**. All of them exhibit *in situ decrease* (i.e., there is an edge $(F_i, F_i, \succ)$ for some $i$) at $\mathsf{F}_1$ in the first graph, at $\mathsf{F}_2$ in the second graph, and at both $\mathsf{F}_1$ and $\mathsf{F}_2$ in the third graph. This means that the original size-change graphs from Figure 6.2**(a)** *satisfy the SCT property.*

Earlier work [TG05] shows how to apply SCT on TRSs. Let $\mathcal{R}$ be a TRS and $(\succsim, \succ)$ a reduction pair such that if $s \succ t$ (or $s \succsim t$), then $t$ contains no defined symbols of $\mathcal{R}$, i.e., no root symbols of left-hand sides of rules from $\mathcal{R}$.[65] Let $\mathcal{G}$ be the set of size-change graphs resulting from all DPs with $(\succsim, \succ)$. In [TG05] the authors prove that if $\mathcal{G}$ satisfies SCT then $\mathcal{R}$ is terminating w.r.t. an *innermost* rewriting strategy.[66]

**Example 6.4.** If one restricts the reduction pair in Example 6.3 to just terms without defined symbols, then one still obtains the same size-change graphs. Since these graphs satisfy the SCT property, one can conclude that the TRS is indeed terminating w.r.t. an innermost evaluation strategy. Note that to show termination without SCT, an order like RPO would fail (since the first rule requires a lexicographic comparison and the second requires a multiset comparison). While termination could be proved by polynomial orders, as in [TG05] one could unite these rules with the rules for the Ackermann function. Then SCT with the embedding order would still work, whereas a direct application of RPO or polynomial orders fails.

So our example illustrates a major strength of SCT. A proof of termination is obtained by using just a *simple* base order and by considering each idempotent graph in the closure under composition afterwards. In contrast, without SCT, one would need more complex termination arguments.

In [TG05] the authors show that when constructing the size-change graphs from the DPs, one can even use *arbitrary* reduction pairs as base orders, provided that all rules of the TRS are weakly decreasing. In other words, this previous work essentially addresses the following question for any DP problem $(\mathcal{P}, \mathcal{R})$:

> For a given reduction pair where $\mathcal{R}$ is weakly decreasing, do all idempotent size-change graphs, under composition closure, exhibit in situ decrease?

Note that in [TG05], the base order is always given and the only way to search for a base order automatically would be a hopeless generate-and-test approach.

## 6.2 Tuple-Typed DP Problems

Before we can proceed with an integration of size-change termination into the DP framework in the form of novel reduction pairs, in this section we still need to resolve a technical issue, which is however not problematic in practical instances.

As we have seen, size-change termination clearly distinguishes between *program points* in the control flow (an infinite sequence of which indicates non-termination) and *arguments* of such program points (where only the size changes matter, not possible non-termination).

[65] Strictly speaking, this is not a reduction pair, since it is only stable under substitutions which do not introduce defined symbols.

[66] When rewriting w.r.t. an innermost rewriting strategy, one may only replace redexes that do not have another redex as a strict subterm.

The very same idea is also underlying the *dependency pair approach* [AG01], where fresh *tuple symbols* are introduced to denote where a (potentially non-terminating) sequence of rewriting steps takes place. Tuple symbols are intended to occur only at the root of the terms of such an evaluation sequence, and they are exactly the symbols that occur as $root(s)$ and $root(t)$ in some dependency pair $s \to t$. Below the root, in contrast, we only have symbols from the original TRS, which is used for the evaluations of the arguments, and we call these original symbols *base symbols*.

However, the dependency pair *framework* [GTS05a, GTSF06, Thi07] is more general. Following [GTSF06], the TRSs $\mathcal{P}$ and $\mathcal{R}$ of a DP problem $(\mathcal{P}, \mathcal{R})$ can be completely arbitrary. On the one hand, this can be convenient to relax requirements on the *outputs* of DP processors. But on the other hand, this also means that DP processors in general cannot make many assumptions on the shape of their *inputs* if they are supposed to be applicable on as wide a variety of DP problems as possible. Moreover, also the substitutions that are used for a $(\mathcal{P}, \mathcal{R})$-chain may instantiate variables with tuple symbols.

One instance of such very generally applicable techniques is the *reduction pair processor* (cf. Theorem 2.13). Here the underlying term orders in the form of a reduction pair are required to be sufficiently general (cf. Definition 2.10) that they can handle DP problems with arbitrary TRSs. However, for instance certain classes of term orders actually need the property that tuple symbols never occur within a non-empty context. As we shall show, a significant subset of *size-change termination*, namely *size-change termination in NP (SCNP)* [BC08] can be represented by such orders. Here program points correspond to tuple symbols, and a nesting of program points only makes little sense.

In order to be able to model SCNP as a reduction pair, it is therefore necessary to restrict the set of terms that need to be considered. More concretely, we shall consider (monomorphically) *typed* terms where the typing enforces that tuple symbols can only occur at the root of a term. This way, we capture both the intuition for program points in size-change termination and the intuition for tuple symbols in dependency pairs in a unified setting.

To achieve this, we use the two types $\mathsf{tuple}$ and $\mathsf{base}$. Here we use $\mathsf{tuple}$ only as the result type for tuple symbols, and we use $\mathsf{base}$ in all other cases. The goal is then to be able to switch back and forth between the usual untyped DP problems and DP problems which are typed in this way. This way, termination proving techniques that require this typed setting also become available for the untyped setting.

We say that a DP problem has the *tuple property* [FGP$^+$11] if the root symbols of terms in $\mathcal{P}$ occur *only* at the root of terms in $\mathcal{P}$ and nowhere else in $\mathcal{P}$ or $\mathcal{R}$. In particular, for $s \to t \in \mathcal{P}$, the terms $s$ and $t$ then *have* root symbols and are not variables, so no rule of $\mathcal{P}$ is collapsing. Sternagel and Thiemann [ST11a] refer to such a DP problem as a *standard DP problem*.

The initial DP problem of the DP framework has the tuple property, and most DP

processors from the literature preserve it. Also the DP problems that stem from automatic translations from programming languages such as Haskell [GRS+11], Prolog [SGST09], or Java Bytecode [OBEG10, BOG11] satisfy the tuple property. Thus, restricting a DP processor to inputs that have the tuple property is not a hard restriction in practice. So in the DP framework, for a DP problem $(\mathcal{P}, \mathcal{R})$ with the tuple property we call the root symbols of terms in $\mathcal{P}$ *tuple symbols*.

Thus, the goal of this section is to lift DP problems with the tuple property to a *typed setting* where tuple symbols have a new output type tuple, and otherwise we only use the type base. That means that in a DP chain, we need not consider any term containing tuple symbols in a substitution, and nested tuple symbols cannot occur in a well-typed term either. We show that for DP problems that fulfill the tuple property, we can switch between the completely untyped setting and the typed setting.

Recall that up to now we have regarded ordinary TRSs and DP problems over untyped signatures $\mathcal{F}$. The following definition shows how to extend such signatures by (monomorphic) types, cf., e.g., [Zan94].

**Definition 6.5** (Typing)**.** *Let $\mathcal{F}$ be an (untyped) signature. A many-sorted signature $\mathcal{F}'$ is a* typed variant *of $\mathcal{F}$ if it contains the same function symbols as $\mathcal{F}$, with the same arities. So $f$ is a symbol of $\mathcal{F}$ with arity $n$ iff $f$ is a symbol of $\mathcal{F}'$ with a type of the form $\tau_1 \times \ldots \times \tau_n \to \tau$. Similarly, a typed variant $\mathcal{V}'$ of the set of variables $\mathcal{V}$ contains the same variables as $\mathcal{V}$, but now every variable has a type $\tau$. We always assume that for every type $\tau$, $\mathcal{V}'$ contains infinitely many variables of type $\tau$. A term over $\mathcal{F}$ and $\mathcal{V}$ is* well typed *w.r.t. $\mathcal{F}'$ and $\mathcal{V}'$ iff*

- *$t$ is a variable (of some type $\tau$ in $\mathcal{V}'$) or*
- *$t = f(t_1, \ldots, t_n)$ with $n \geq 0$, where all $t_i$ are well typed and have some type $\tau_i$, and where $f$ has type $\tau_1 \times \ldots \times \tau_n \to \tau$ in $\mathcal{F}'$. Then $t$ has type $\tau$.*

*We only permit typed variants $\mathcal{F}'$ where there exist well-typed ground terms of types $\tau_1, \ldots, \tau_n$ over $\mathcal{F}'$, whenever some $f \in \mathcal{F}'$ has type $\tau_1 \times \ldots \times \tau_n \to \tau$.*[67]

*A TRS $\mathcal{R}$ over*[68] *$\mathcal{F}$ and $\mathcal{V}$ is* well typed *w.r.t. $\mathcal{F}'$ and $\mathcal{V}'$ iff for all $\ell \to r \in \mathcal{R}$, we have that $\ell$ and $r$ are well typed and that they have the same type.*[69] *Similarly, a DP problem $(\mathcal{P}, \mathcal{R})$ over $\mathcal{F}$ and $\mathcal{V}$ is* well typed *w.r.t. $\mathcal{F}'$ and $\mathcal{V}'$ iff $\mathcal{P} \cup \mathcal{R}$ is well typed.*

In the setting of this chapter, we only consider the typed variants where tuple symbols have the output type tuple and where otherwise only the type base is used. When using such a typed variant as the underlying (many-sorted) signature, we call a DP problem $(\mathcal{P}, \mathcal{R})$ also a *tuple-typed DP problem* and write $(\mathcal{P}, \mathcal{R})^{tt}$.

[67] This is not a restriction, as one can simply add new constants to $\mathcal{F}$ and $\mathcal{F}'$.

[68] Note that $\mathcal{F}$ may well contain function symbols that do not occur in $\mathcal{R}$.

[69] W.l.o.g., here one may rename the variables in every rule. Then it is not a problem if the variable $x$ is used with type $\tau_1$ in one rule and with type $\tau_2$ in another rule.

**Definition 6.6** (Tuple Typing, Base Terms, Tuple Terms)**.** *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem with the tuple property where $\mathcal{F}_{tup}$ are the underlying tuple symbols. Then we call the typed variant $\mathcal{F}'$ the* tuple typing *(for $(\mathcal{P}, \mathcal{R})$) with $F : \mathsf{base}^n \to \mathsf{tuple} \in \mathcal{F}'$ where $ar(F) = n$ and $F \in \mathcal{F}_{tup}$ and with $g : \mathsf{base}^n \to \mathsf{base} \in \mathcal{F}'$ where $ar(g) = n$, otherwise. Correspondingly, we call $(\mathcal{P}, \mathcal{R})$* tuple typed *(by $\mathcal{F}'$), and we also write $(\mathcal{P}, \mathcal{R})^{tt}$ to indicate that we only need to consider chains with terms which are well typed with respect to the tuple typing $\mathcal{F}'$.*[70]

*If we work on a tuple typing, we call a term $t$ a* base term *if $t$ has the type* base*, and we call $t$ a* tuple term *if $t$ has the type* tuple*.*

**Corollary 6.7** (Tuple-Typed DP Problems are Well Typed)**.** *Let $(\mathcal{P}, \mathcal{R})^{tt}$ be a tuple-typed DP problem with the tuple property. Then $(\mathcal{P}, \mathcal{R})^{tt}$ is well typed with respect to its tuple typing.*

It is worth noting that in related work [LM08], Lucas and Meseguer propose a dependency pair approach for *order-sorted* rewriting where they also use a dedicated fresh type for all arising tuple symbols. Here, in contrast to our setting, also the original TRS is already typed. Despite the typed setting, the reduction pairs used in [LM08, Theorem 5] are defined for the classic untyped setting, i.e., they need to deal with the untyped variants of the TRS rules and the DPs. It would be interesting to investigate for the order-sorted setting how one could further benefit from the explicit types to improve the termination techniques based on reduction pairs.

Now let us consider an example for a tuple typing.

**Example 6.8** (Tuple Typing for Example 6.3)**.** Consider again Example 6.3. The DP problem $(\mathcal{P}, \mathcal{R})$ has the tuple property. The corresponding tuple-typed DP problem is $(\mathcal{P}, \mathcal{R})^{tt}$ where we have the tuple typing $\mathcal{F}'$, which includes:

$$\mathsf{F} : \mathsf{base} \times \mathsf{base} \to \mathsf{tuple} \tag{6.5}$$

$$\mathsf{f} : \mathsf{base} \times \mathsf{base} \to \mathsf{base} \tag{6.6}$$

$$\mathsf{s} : \mathsf{base} \to \mathsf{base} \tag{6.7}$$

The following theorem now states that in the DP framework we can indeed switch back and forth between untyped DP problems and corresponding tuple-typed DP problems.[71]

**Theorem 6.9** (Tuple Typing Does Not Alter Finiteness)**.** *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem which has both the tuple property and the tuple typing $\mathcal{F}'$ (which gives rise to the tuple-typed DP problem $(\mathcal{P}, \mathcal{R})^{tt}$). There is an infinite minimal $(\mathcal{P}, \mathcal{R})$-chain in the untyped*

[70] Existing DP processors *Proc* can easily be lifted to this extended DP framework which may also contain tuple-typed DP problems $(\mathcal{P}, \mathcal{R})^{tt}$ simply by defining $Proc(\,(\mathcal{P}, \mathcal{R})^{tt}\,) = \{\,(\mathcal{P}, \mathcal{R})^{tt}\,\}$.

[71] By analogous reasoning, the theorem also holds if one does not consider minimal, but arbitrary chains. Likewise, the theorem also holds if one considers an innermost rewriting strategy instead of a full rewriting strategy.

*setting iff there is an infinite minimal $(\mathcal{P},\mathcal{R})$-chain over terms which are well typed with respect to its tuple typing $\mathcal{F}'$.*

*Proof.* Consider the right-to-left direction of the statement. Let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite minimal $(\mathcal{P},\mathcal{R})$-chain over terms which are well typed with respect to $\mathcal{F}'$, with the substitution $\sigma$. Using $\sigma$, the same infinite minimal $(\mathcal{P},\mathcal{R})$-chain carries over also to the untyped setting.

Now consider the left-to-right direction of the statement. Let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite minimal $(\mathcal{P},\mathcal{R})$-chain with the substitution $\sigma$ where $\sigma$ may map to terms which contain tuple symbols. For each $F : \mathsf{base}^n \to \mathsf{tuple} \in \mathcal{F}'$ introduce a corresponding fresh symbol $F_{base} : \mathsf{base}^n \to \mathsf{base}$. (The idea is to replace tuple symbols $F$ occurring in a chain below the root position, which are the only possible way well-typedness could be affected here, by corresponding fresh symbols $F_{base}$ in any infinite minimal chain.) Let *toBase* map terms to terms by $\mathit{toBase}(x) = x$ for $x \in \mathcal{V}$, $\mathit{toBase}(f(t_1,\ldots,t_n)) = f(\mathit{toBase}(t_1),\ldots,\mathit{toBase}(t_n))$ if $f$ is not a tuple symbol, and $\mathit{toBase}(F(t_1,\ldots,t_n)) = F_{base}(\mathit{toBase}(t_1),\ldots,\mathit{toBase}(t_n))$ if $F$ is a tuple symbol. Now let the substitution $\delta$ be defined via $\delta(x) = \mathit{toBase}(t)$ if $\sigma(x) = t$. As witnessed by $\delta$, then also $s_1 \to t_1, s_2 \to t_2, \ldots$ is an infinite minimal $(\mathcal{P},\mathcal{R})$-chain using only terms from $\mathcal{T}(\mathcal{F}',\mathcal{V})$. The reason that the minimality behavior is not changed is that by the tuple property of the DP problem $(\mathcal{P},\mathcal{R})$ and by construction of *toBase*, the same reduction steps are possible on a term $\mathit{toBase}(s)$ and on a term $s$. Moreover, for two terms $s,t$ we have $\mathit{toBase}(s) \to_{\mathcal{R}} \mathit{toBase}(t)$ iff $s \to_{\mathcal{R}} t$. Thus, a term $\mathit{toBase}(t)$ is terminating with respect to $\mathcal{R}$ iff $t$ is terminating with respect to $\mathcal{R}$. □

This theorem now allows us to switch between the untyped setting and the setting where a tuple typing is used for a DP problem.

**Corollary 6.10** (Tuple Typing Introduction)**.** *Let Proc be a DP processor defined as follows:*

$$\mathit{Proc}(\,(\mathcal{P},\mathcal{R})\,) = \begin{cases} \{\,(\mathcal{P},\mathcal{R})^{tt}\,\}, & \text{if } (\mathcal{P},\mathcal{R}) \text{ has the tuple property} \\ \{\,(\mathcal{P},\mathcal{R})\,\}, & \text{otherwise} \end{cases}$$

$$\mathit{Proc}(\,(\mathcal{P},\mathcal{R})^{tt}\,) = \{\,(\mathcal{P},\mathcal{R})^{tt}\,\}$$

*Then Proc is sound.*

**Corollary 6.11** (Tuple Typing Elimination)**.** *Let Proc be a DP processor defined as*

*follows:*

$$Proc(\,(\mathcal{P},\mathcal{R})^{tt}\,) = \begin{cases} \{\,(\mathcal{P},\mathcal{R})\,\}, & \text{if } (\mathcal{P},\mathcal{R}) \text{ has the tuple property} \\ \{\,(\mathcal{P},\mathcal{R})^{tt}\,\}, & \text{otherwise} \end{cases}$$

$$Proc(\,(\mathcal{P},\mathcal{R})\,) = \{\,(\mathcal{P},\mathcal{R})\,\}$$

*Then Proc is sound.*

So for DP problems with the tuple property, we can choose if we want to use the tuple-typed or the usual untyped setting. Thus, we can benefit from the fact that termination techniques do not need to consider terms which are not well typed with respect to the tuple typing. One instance of such techniques are *reduction pairs* (cf. Definition 2.10).

In the tuple-typed setting, it suffices to ensure the required properties of a reduction pair $(\gtrsim, \succ)$ on *well-typed terms* instead of all terms. In particular, this means that one does not require monotonicity for tuple terms. The reason is that in the tuple-typed setting, there are no non-empty contexts for the type tuple. We call a reduction pair on tuple-typed terms with the tuple typing $\mathcal{F}'$ also a *tuple-typed reduction pair* (for $\mathcal{F}'$). In the remainder of this chapter, we develop such a class of tuple-typed reduction pairs in the form of *SCNP reduction pairs*.

A review of the reduction pair processor (cf. Theorem 2.13) with its main extensions (usable rules for innermost [AG00] and full termination [GTSF06], usable rules with respect to an argument filtering [GTSF06]) shows that this central processor of the DP framework is also sound for tuple-typed DP problems with tuple-typed reduction pairs. The corresponding soundness proofs from the untyped setting directly carry over to the tuple-typed setting.

Thus, we can now also use orders for reduction pairs in the DP framework that require that tuple symbols only occur at the root of a term. This contribution is a stepping stone for the SCNP reduction pairs, but may well have implications beyond the present thesis. Without loss of generality, we can now use the untyped or the tuple-typed setting when developing a new DP processor for DP problems which satisfy the tuple property.

## 6.3 Approximating SCT in NP

In [BC08] the authors identify a subset of SCT, called SCNP, that is powerful enough for practical use and is in NP. For SCNP just as for SCT, programs are abstracted to sets of size-change graphs. But instead of checking SCT by the closure under composition, one identifies a suitable ranking function to certify the termination of programs described by the set of graphs. Ranking functions map "program states" to elements of a well-founded domain and one has to show that they (strictly) decrease on all program transitions

described by the size-change graphs.

In the rewriting context, program states are terms. Here, instead of a ranking function one can use an arbitrary stable well-founded order $\sqsupset$. Let $(V_s, V_t, E)$ be a size-change graph which has source vertices $V_s = \{F_1, \ldots, F_n\}$ and target vertices $V_t = \{G_1, \ldots, G_m\}$, and let $(\succsim, \succ)$ be the reduction pair on base terms which was used for the construction of the graph. Now the goal is to extend the order $\succ$ to a well-founded order $\sqsupset$ which can also compare tuple terms and which *satisfies* the size-change graph (i.e., $F(s_1, \ldots, s_n) \sqsupset G(t_1, \ldots, t_m)$). Similarly, we say that $\sqsupset$ satisfies a *set* of size-change graphs iff it satisfies all the graphs in the set.

If the size-change graphs describe the transitions of a program, then the existence of a corresponding ranking function obviously implies termination of the program. As in [TG05], to ensure that the size-change graphs really describe the transitions of the TRS correctly, one has to impose suitable restrictions on the reduction pair (e.g., by demanding that all rules of the TRS are weakly decreasing w.r.t. $\succsim$). Then one can indeed conclude termination of the TRS.

In [Lee09], a class of ranking functions is identified which can simulate SCT. So if a set of size-change graphs has the SCT property, then there is a ranking function of that class satisfying these size-change graphs. However, expressions for these ranking functions can be exponential in size [Lee09]. To obtain a subset of SCT in NP, [BC08] considers a restricted class of ranking functions. A set of size-change graphs has the *SCNP property* iff it is satisfied by a ranking function from this restricted class.

Our goal is to adapt this class of ranking functions to term rewriting. The main motivation is to facilitate the *simultaneous* search for a ranking function on the size-change graphs and for the base order which is used to derive the size-change graphs from a TRS. It means that we are searching both for a program abstraction to size-change graphs, and also for the ranking function which proves that these graphs have the SCNP (and hence also the SCT) property.

This is different from [BC08], where the concrete structure of the program has already been abstracted away to size-change graphs that must be given as inputs. It is also different from the earlier adaption of SCT to term rewriting in [TG05], where the base order was fixed. As shown by the experiments with [TG05] in Section 6.6, fixing the base order for the size-change graphs leads to severe limitations in power.

The following example illustrates the SCNP property and presents a ranking function (inducing a well-founded order $\sqsupset$) satisfying a set of size-change graphs.

**Example 6.12.** Consider the DP problem from Example 6.3 and its size-change graphs in Figure 6.2**(a)**. Here, the base order is the reduction pair $(\succsim, \succ)$ resulting from the embedding order. We now extend $\succ$ to an order $\sqsupset$ which can also compare tuple terms and which satisfies the size-change graphs in this example. To compare tuple terms $\mathsf{F}(s_1, s_2)$ and $\mathsf{F}(t_1, t_2)$, we first map them to the multisets $\{ \langle s_1, 1\rangle, \langle s_2, 0\rangle \}$ and $\{ \langle t_1, 1\rangle, \langle t_2, 0\rangle \}$ of

*tagged* terms (where a tagged term is a pair of a term and a number). Now a multiset $S$ of tagged terms is greater than a multiset $T$ of tagged terms iff for every $\langle t, m\rangle \in T$ there is an $\langle s, n\rangle \in S$ where $s \succ t$ or both $s \succsim t$ and $n > m$.

For the first graph, we have $s_1 \succ t_1$ and $s_1 \succsim t_2$ and hence the multiset $\{\, \langle s_1, 1\rangle, \langle s_2, 0\rangle \,\}$ is greater than $\{\, \langle t_1, 1\rangle, \langle t_2, 0\rangle \,\}$. For the second graph, $s_1 \succsim t_2$ and $s_2 \succ t_1$ also implies that the multiset $\{\, \langle s_1, 1\rangle, \langle s_2, 0\rangle \,\}$ is greater than $\{\, \langle t_1, 1\rangle, \langle t_2, 0\rangle \,\}$. Thus, if we define our well-founded order $\sqsupset$ in this way, then it indeed satisfies both size-change graphs of the example. Since this order $\sqsupset$ belongs to the class of ranking functions defined in [BC08], this shows that the size-change graphs in Figure 6.2**(a)** have the SCNP property.

## Integrating SCNP into the DP Framework

In term rewriting, size-change graphs correspond to DPs and the arcs of the size-change graphs are built by only comparing the arguments of the DPs (which are *base terms*). The ranking function then corresponds to a well-founded order on tuple terms. We now reformulate the class of ranking functions of [BC08] in the term rewriting context by defining *SCNP reduction pairs*. The advantage of this reformulation is that it allows us to integrate the SCNP approach directly into the DP framework and that it allows a SAT encoding of both the search for suitable base orders and of the test for the SCNP property.

This way, we can use the reduction pair processor to ask the following question:

> For a given DP problem $(\mathcal{P}, \mathcal{R})$, is there a(n SCNP) reduction pair that orients all rules of $\mathcal{R}$ and $\mathcal{P}$ weakly and at least one of the rules of $\mathcal{P}$ strictly?

In [BC08], the class of ranking functions for SCNP is defined incrementally. We follow this, but adapt the definitions of [BC08] to the (tuple-typed) term rewriting setting and prove that the resulting orders always constitute reduction pairs. More precisely, we proceed as follows:

**step one:** $(\succsim, \succ)$ is an arbitrary reduction pair on base terms that we start with (e.g., based on RPO and argument filters or on polynomial orders). The main observation that can be drawn from the SCNP approach is that it is helpful to compare base terms and tuple terms in a different way. Thus, our goal is to extend $(\succsim, \succ)$ appropriately to a tuple-typed reduction pair $(\sqsupseteq, \sqsupset)$. By defining $(\sqsupseteq, \sqsupset)$ in the same way as the ranking functions of [BC08], it can simulate the SCNP approach.

**step two:** $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is a reduction pair on *tagged* base terms, i.e., on pairs $\langle t, n\rangle$, where $t$ is a base term and $n \in \mathbb{N}$. Essentially, $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is a lexicographic combination of the reduction pair $(\succsim, \succ)$ with the usual order on $\mathbb{N}$.

**step three:** $(\succsim^{\mathbb{N},\mu}, \succ^{\mathbb{N},\mu})$ extends $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ to compare *multisets* of tagged base terms. The *status* $\mu$ determines how $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is extended to multisets.

**step four:** $(\succsim^{\mu,L}, \succ^{\mu,L})$ is a full tuple-typed reduction pair (i.e., it is the reduction pair $(\sqsupseteq, \sqsupset)$ we were looking for). The *level mapping* $L$ determines which arguments of a tuple term are selected and tagged, resulting in a multiset of tagged base terms. On tuple terms, $(\succsim^{\mu,L}, \succ^{\mu,L})$ behaves according to $(\succsim^{\mathbb{N},\mu}, \succ^{\mathbb{N},\mu})$ on the multisets as determined by $L$, and on base terms, it behaves like $(\succsim, \succ)$.

Thus, we start with extending a reduction pair $(\succsim, \succ)$ on base terms to a reduction pair $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ on tagged base terms. We compare tagged terms lexicographically by $(\succsim, \succ)$ and by the standard orders $\geq$ and $>$ on numbers.

**Definition 6.13** (Comparing Tagged Terms)**.** *Let $(\succsim, \succ)$ be a reduction pair on terms. We define the corresponding reduction pair $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ on tagged terms:*

- $\langle t_1, n_1\rangle \succsim^{\mathbb{N}} \langle t_2, n_2\rangle \quad \Leftrightarrow \quad t_1 \succ t_2 \ \vee \ (t_1 \succsim t_2 \wedge n_1 \geq n_2)$.
- $\langle t_1, n_1\rangle \succ^{\mathbb{N}} \langle t_2, n_2\rangle \quad \Leftrightarrow \quad t_1 \succ t_2 \ \vee \ (t_1 \succsim t_2 \wedge n_1 > n_2)$.

The motivation for tagged terms is that we will use different tags (i.e., numbers) for the different argument positions of a function symbol. For instance, when comparing the terms $s = \mathsf{F}(\mathsf{s}(x), y)$ and $t = \mathsf{F}(x, \mathsf{s}(x))$ as in Example 6.12, one can assign the tags 1 and 0 to the first and second argument position of $\mathsf{F}$, respectively. Then, if $(\succsim, \succ)$ is the reduction pair based on the embedding order, we have $\langle \mathsf{s}(x), 1\rangle \succ^{\mathbb{N}} \langle x, 1\rangle$ and $\langle \mathsf{s}(x), 1\rangle \succ^{\mathbb{N}} \langle \mathsf{s}(x), 0\rangle$. In other words, the first argument of $s$ is greater than both the first and the second argument of $t$.

The following lemma states that if $(\succsim, \succ)$ is a reduction pair on terms, then $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is a reduction pair on tagged terms (where we do not require monotonicity, since monotonicity is not defined for tagged terms). This lemma will be needed for our main theorem (Theorem 6.20), which proves that the reduction pair defined to simulate SCNP is really a reduction pair.

**Lemma 6.14** (Reduction Pairs on Tagged Terms)**.** *Let $(\succsim, \succ)$ be a reduction pair. Then $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is a non-monotonic reduction pair on tagged terms.*

*Proof.* Reflexivity of $\succsim^{\mathbb{N}}$ follows directly from reflexivity of $\succsim$ and of $\geq$. Transitivity of $\succsim^{\mathbb{N}}$ and $\succ^{\mathbb{N}}$ as well as compatibility follow from a simple case analysis on whether we have strict or weak decrease in the term components. Well-foundedness of $\succ^{\mathbb{N}}$ follows from the well-foundedness of lexicographic combinations of well-founded orders.

We now prove stability of $\succsim^{\mathbb{N}}$ by contradiction. Assume that $\succsim^{\mathbb{N}}$ were not stable, i.e., there exist two tagged terms $\langle s, c\rangle$, $\langle t, d\rangle$ and a substitution $\sigma$ with $\langle s, c\rangle \succsim^{\mathbb{N}} \langle t, d\rangle$ and $\langle s\sigma, c\rangle \not\succsim^{\mathbb{N}} \langle t\sigma, d\rangle$. By definition, this implies $\neg(s\sigma \succ t\sigma \vee (s\sigma \succsim t\sigma \wedge c \geq d))$. But by

$\langle s, c\rangle \succsim^{\mathbb{N}} \langle t, d\rangle$, we have $s \succ t \vee (s \succsim t \wedge c \geq d)$ which is a contradiction to the stability of $\succ$ and $\succsim$. Stability of $\succ^{\mathbb{N}}$ is completely analogous. □

The next step is to introduce a "reduction pair" $(\succsim^{\mathbb{N},\mu}, \succ^{\mathbb{N},\mu})$ on multisets of tagged base terms, where $\mu$ is a status which determines how $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ is extended to multisets. Of course, there are many possibilities for such an extension. In Definition 6.15, we present the four extensions which correspond to the ranking functions defining SCNP in [BC08]. The main difference to the definitions in [BC08] is that we do not restrict ourselves to *total* base orders. Hence, the notions of maximum and minimum of a multiset of terms are not defined in the same way as in [BC08].

**Definition 6.15** (Multiset Extensions of Reduction Pairs)**.** *Let $(\succsim, \succ)$ be a reduction pair on (tagged) terms. We define an extended reduction pair $(\succsim^{\mu}, \succ^{\mu})$ on multisets of (tagged) terms, for $\mu \in \{max, min, ms, dms\}$. Let $S$ and $T$ be multisets of (tagged) terms.*

*(i)* (max order) $S \succsim^{max} T$ *holds iff* $\forall t{\in}T.\ \exists s{\in}S.\ s \succsim t$.
$S \succ^{max} T$ *holds iff* $S \neq \varnothing$ *and* $\forall t{\in}T.\ \exists s{\in}S.\ s \succ t$.

*(ii)* (min order) $S \succsim^{min} T$ *holds iff* $\forall s{\in}S.\ \exists t{\in}T.\ s \succsim t$.
$S \succ^{min} T$ *holds iff* $T \neq \varnothing$ *and* $\forall s{\in}S.\ \exists t{\in}T.\ s \succ t$.

*(iii)* (multiset order [DM79]) $S \succ^{ms} T$ *holds iff* $S = S_{\text{strict}} \uplus \{ s_1, \ldots, s_k \}$, $T = T_{\text{strict}} \uplus \{ t_1, \ldots, t_k \}$, $S_{\text{strict}} \succ^{max} T_{\text{strict}}$, *and* $s_i \succsim t_i$ *for* $1 \leq i \leq k$.
$S \succsim^{ms} T$ *holds iff* $S = S_{\text{strict}} \uplus \{ s_1, \ldots, s_k \}$, $T = T_{\text{strict}} \uplus \{ t_1, \ldots, t_k \}$, *either* $S_{\text{strict}} \succ^{max} T_{\text{strict}}$ *or* $S_{\text{strict}} = T_{\text{strict}} = \varnothing$, *and* $s_i \succsim t_i$ *for* $1 \leq i \leq k$.

*(iv)* (dual multiset order [BL07]) $S \succ^{dms} T$ *holds iff* $S = S_{\text{strict}} \uplus \{ s_1, \ldots, s_k \}$, $T = T_{\text{strict}} \uplus \{ t_1, \ldots, t_k \}$, $S_{\text{strict}} \succ^{min} T_{\text{strict}}$, *and* $s_i \succsim t_i$ *for* $1 \leq i \leq k$.
$S \succsim^{dms} T$ *holds iff* $S = S_{\text{strict}} \uplus \{ s_1, \ldots, s_k \}$, $T = T_{\text{strict}} \uplus \{ t_1, \ldots, t_k \}$, *either* $S_{\text{strict}} \succ^{min} T_{\text{strict}}$ *or* $S_{\text{strict}} = T_{\text{strict}} = \varnothing$, *and* $s_i \succsim t_i$ *for* $1 \leq i \leq k$.

Here $\succ^{ms}$ is the standard multiset extension of an order $\succ$ as used, e.g., for the classical definition of RPO. However, our use of tagged terms as elements of the multiset introduces a lexicographic aspect that is missing in RPO.

**Example 6.16.** Consider again the TRS from Example 6.3 with the reduction pair based on the embedding order. We have $\{\mathsf{s}(x), y\} \succsim^{max} \{x, \mathsf{s}(x)\}$, since for both terms in $\{x, \mathsf{s}(x)\}$ there is an element in $\{\mathsf{s}(x), y\}$ which is weakly greater (w.r.t. $\succsim$). Similarly, $\{x, \mathsf{s}(y)\} \succsim^{max} \{y, x\}$. However, $\{x, \mathsf{s}(y)\} \not\succ^{max} \{y, x\}$, since not every element from $\{y, x\}$ has a strictly greater one in $\{x, \mathsf{s}(y)\}$. We also have $\{x, \mathsf{s}(y)\} \succsim^{min} \{y, x\}$, but $\{\mathsf{s}(x), y\} \not\succsim^{min} \{x, \mathsf{s}(x)\}$, since for $y$ in $\{\mathsf{s}(x), y\}$, there is no term in $\{x, \mathsf{s}(x)\}$ which is weakly smaller.

We have $\{\mathsf{s}(x), y\} \not\succ^{ms} \{x, \mathsf{s}(x)\}$, since even if we take $\{\mathsf{s}(x)\} \succ^{max} \{x\}$, we still do not have $y \succsim \mathsf{s}(x)$. Moreover, also $\{\mathsf{s}(x), y\} \not\precsim^{ms} \{x, \mathsf{s}(x)\}$. Otherwise, for every element of $\{x, \mathsf{s}(x)\}$ there would have to be a *different* weakly greater element in $\{\mathsf{s}(x), y\}$. In contrast, we have $\{x, \mathsf{s}(y)\} \succ^{ms} \{y, x\}$. The element $\mathsf{s}(y)$ is replaced by the strictly smaller element $y$ and for the remaining element $x$ on the right-hand side there is a weakly greater one on the left-hand side. Similarly, we also have $\{\mathsf{s}(x), y\} \not\precsim^{dms} \{x, \mathsf{s}(x)\}$ and $\{x, \mathsf{s}(y)\} \succ^{dms} \{y, x\}$.

So there is no $\mu$ such that the multiset of arguments strictly decreases in some DP and weakly decreases in the other DP. We can only achieve a weak decrease for all DPs. To obtain a strict decrease in such cases, one can add tags.

We want to define a reduction pair $(\succsim^{\mu,L}, \succ^{\mu,L})$ which is like $(\succsim^{\mathbb{N},\mu}, \succ^{\mathbb{N},\mu})$ on tuple terms and like $(\succsim, \succ)$ on base terms. Here, we use a *level mapping* $L$ to map tuple terms $F(s_1, \ldots, s_n)$ to multisets of tagged base terms.

**Definition 6.17** (Level Mapping)**.** *For each tuple symbol $F$ of arity $n$, let $\pi(F) \subseteq \{1, \ldots, n\} \times \mathbb{N}$ such that for each $1 \leq j \leq n$ there is at most one $m \in \mathbb{N}$ with $\langle j, m\rangle \in \pi(F)$. Then $L(F(s_1, \ldots, s_n)) = \{\langle s_i, n_i\rangle \mid \langle i, n_i\rangle \in \pi(F)\}$.*[72]

**Example 6.18.** Consider again the TRS from Example 6.3 with the reduction pair based on the embedding order. Let $\pi$ be a status function with $\pi(\mathsf{F}) = \{\langle 1, 1\rangle, \langle 2, 0\rangle\}$. So $\pi$ selects both arguments of terms rooted with $\mathsf{F}$ for comparison and associates the tag 1 with the first argument and the tag 0 with the second argument. This means that it puts "more weight" on the first than on the second argument. The level mapping $L$ defined by $\pi$ transforms the tuple terms from the DPs of our TRS into the following multisets of tagged terms:

$$\begin{array}{rclcrcl} L(\mathsf{F}(\mathsf{s}(x), y)) & = & \{\langle \mathsf{s}(x), 1\rangle, \langle y, 0\rangle\} & \qquad & L(\mathsf{F}(x, \mathsf{s}(x))) & = & \{\langle x, 1\rangle, \langle \mathsf{s}(x), 0\rangle\} \\ L(\mathsf{F}(x, \mathsf{s}(y))) & = & \{\langle x, 1\rangle, \langle \mathsf{s}(y), 0\rangle\} & & L(\mathsf{F}(y, x)) & = & \{\langle y, 1\rangle, \langle x, 0\rangle\} \end{array}$$

Now we observe that for the multisets of the tagged terms above, we have

$$L(\mathsf{F}(\mathsf{s}(x), y)) \succ^{\mathbb{N},max} L(\mathsf{F}(x, \mathsf{s}(x))) \qquad L(\mathsf{F}(x, \mathsf{s}(y))) \succ^{\mathbb{N},max} L(\mathsf{F}(y, x))$$

So due to the tagging, now we can find an order such that both DPs are strictly decreasing. This order corresponds to the ranking function given in Example 6.12.

Finally we define the class of reduction pairs which corresponds to the class of ranking functions considered for SCNP in [BC08].

[72] One could also allow multiple tags $n_{i_1}, n_{i_2}, \ldots$ with $n_{i_1}, n_{i_2}, \ldots \in \pi(F)$ for the same $F$ as in [BC08, Definition 9]. For simplicity, here we follow the SAT encoding of [BC08, Section 5], where only one tag $n_i$ is used for fixed $F$ and $i$ such that $\langle i, n_i\rangle \in \pi(F)$.

**Definition 6.19** (SCNP Reduction Pair)**.** *Let $(\succsim, \succ)$ be a reduction pair on base terms and let $L$ be a level mapping. For $\mu \in \{max, min, ms, dms\}$, we define the* SCNP reduction pair $(\succsim^{\mu,L}, \succ^{\mu,L})$ *as follows. For base terms $l$ and $r$ we define $l \mathrel{(\succsim)^{\mu,L}} r \Leftrightarrow l \mathrel{(\succsim)} r$, and for tuple terms $s$ and $t$ we define $s \succ^{\mu,L} t \Leftrightarrow L(s) \succ^{\mathbb{N},\mu} L(t)$ and $s \succsim^{\mu,L} t \Leftrightarrow s = t \vee L(s) \succsim^{\mathbb{N},\mu} L(t)$.*[73]

So we have $s \succ^{max,L} t$ for the DPs $s \to t$ in Example 6.3 and the level mapping $L$ in Example 6.18. Theorem 6.20 states that SCNP reduction pairs actually *are* (tuple-typed) reduction pairs.

**Theorem 6.20.** *For $\mu \in \{max, min, ms, dms\}$, $(\succsim^{\mu,L}, \succ^{\mu,L})$ is a tuple-typed reduction pair.*

In the proof of Theorem 6.20, we make use of *multiset covers* [STA+07, Sch08, CGST12] as an equivalent representation for the multiset extension $\succsim^{ms}$ and $\succ^{ms}$. This is not only convenient for the proof, but multiset covers can also be used to express the multiset extension in a way which is natural to encode as a propositional formula.

**Definition 6.21** (Multiset Cover)**.** *Let $S$ and $T$ be multisets with $|S| = n$ and $|T| = m$. A multiset cover $\langle \gamma, \varepsilon \rangle$ (for $S$ and $T$) is a pair of mappings $\gamma : \{1, \ldots, m\} \to \{1, \ldots, n\}$ and $\varepsilon : \{1, \ldots, n\} \to \{false, true\}$ such that for each $1 \leq i \leq n$, if $\varepsilon(i) = true$, then $\{j \mid \gamma(j) = i\}$ is a singleton set.*

For any $t_j \in T$, $\gamma(j) = i$ intuitively means that $t_j$ is *covered* by $s_i$ (i.e., $s_i \succ t_j$ or $s_i \succsim t_j$). Moreover, $\varepsilon(i)$ indicates whether $s_i$ covers elements from $T$ strictly or weakly. Hence, we can now express the $ms$-extension via multiset covers:

- $S \succsim^{ms} T$ holds iff $S = \{s_1, \ldots, s_n\}$, $T = \{t_1, \ldots, t_m\}$, and there exists a multiset cover $\langle \gamma, \varepsilon \rangle$ such that for all $i, j$:

$$\gamma(j) = i \Rightarrow (\texttt{if } \varepsilon(i) \texttt{ then } s_i \succsim t_j \texttt{ else } s_i \succ t_j)$$

- $S \succ^{ms} T$ holds iff $S \succsim^{ms} T$ and $\varepsilon(i) = false$ for some $i$ (i.e., some $s_i$ is replaced by zero or more strictly smaller elements).

*Proof of Theorem 6.20.* We first illustrate the proof for $\mu = max$ (the proof for $\mu = min$ is analogous).

For base terms, $(\succsim^{max,L}, \succ^{max,L})$ is like $(\succsim, \succ)$ and thus, here it is clearly a reduction pair. Thus, it suffices to consider comparisons between tuple terms.

[73] The explicit requirement for tuple terms $s, t$ that $s = t$ implies $s \succsim^{\mu,L} t$ is needed since we need $\succsim^{\mu,L}$ to be reflexive also on variables to obtain a proper tuple-typed reduction pair, and $L(x)$ is not defined for variables $x$. Note that for a variable $x$ of type tuple, $x \succsim^{\mathbb{N},\mu} t$ or $t \succsim^{\mathbb{N},\mu} x$ implies $x = t$, and $x \succ^{\mathbb{N},\mu} t$ or $t \succ^{\mathbb{N},\mu} x$ does not hold for any term $t$.

(i) $\underline{\succsim^{max,L} \text{ and } \succ^{max,L} \text{ are compatible}}$

We only show $\succ^{max,L} \circ \succsim^{max,L} \subseteq \succ^{max,L}$. The proof for $\succsim^{max,L} \circ \succ^{max,L} \subseteq \succ^{max,L}$ is analogous. Concretely we show that for tuple terms $s, t, u$, we have that $s \succ^{max,L} t \wedge t \succsim^{max,L} u$ implies $s \succ^{max,L} u$. W.l.o.g., we can assume that $s, t$, and $u$ are not variables ($s$ or $t$ being a variable would imply that the premise does not hold, and $u$ being a variable would imply $t = u$). So we get $s = F(\overline{s})$, $t = G(\overline{t})$, and $u = H(\overline{u})$ where $s \succ^{max,L} t \wedge t \succsim^{max,L} u$. Clearly, $s \succ^{max,L} t$ means

$$L(F(\overline{s})) \neq \varnothing \wedge \forall \langle t_j, d_j \rangle \in L(G(\overline{t})).\ \exists \langle s_i, c_i \rangle \in L(F(\overline{s})).\ \langle s_i, c_i \rangle \succ^{\mathbb{N}} \langle t_j, d_j \rangle$$

Furthermore, $t \succsim^{max,L} u$ means

$$\forall \langle u_j, e_j \rangle \in L(H(\overline{u})).\ \exists \langle t_i, d_i \rangle \in L(G(\overline{t})).\ \langle t_i, d_i \rangle \succsim^{\mathbb{N}} \langle u_j, e_j \rangle$$

Since $\succsim^{\mathbb{N}}$ and $\succ^{\mathbb{N}}$ are compatible by Lemma 6.14, we obtain

$$L(F(\overline{s})) \neq \varnothing \wedge \forall \langle u_j, e_j \rangle \in L(H(\overline{u})).\ \exists \langle s_i, c_i \rangle \in L(F(\overline{s})).\ \langle s_i, c_i \rangle \succ^{\mathbb{N}} \langle u_j, e_j \rangle$$

and thus $s \succ^{max,L} t$.

(ii) $\underline{\succsim^{max,L} \text{ is reflexive}}$

Reflexivity of $\succsim^{max,L}$ follows directly from the definition.

(iii) $\underline{\succsim^{max,L} \text{ and } \succ^{max,L} \text{ are stable and transitive}}$

For $x \succsim^{max,L} x$, $x\sigma \succsim^{max,L} x\sigma$ follows by stability of syntactic equality. So let $F(\overline{s}) \succsim^{max,L} G(\overline{t})$, which is equivalent to $L(F(\overline{s})) \succsim^{\mathbb{N},max} L(G(\overline{s}))$ and thus

$$\forall \langle t_j, d_j \rangle \in L(G(\overline{t})).\ \exists \langle s_i, c_i \rangle \in L(F(\overline{s})).\ \langle s_i, c_i \rangle \succsim^{\mathbb{N}} \langle t_j, d_j \rangle \tag{6.8}$$

The claim $F(\overline{s})\sigma \succsim^{max,L} G(\overline{t})\sigma$ now is equivalent to $L(F(\overline{s})\sigma) \succsim^{\mathbb{N},max} L(G(\overline{s})\sigma)$, which means

$$\forall \langle t_j\sigma, d_j \rangle \in L(G(\overline{t})\sigma).\ \exists \langle s_i\sigma, c_i \rangle \in L(F(\overline{s})\sigma).\ \langle s_i\sigma, c_i \rangle \succsim^{\mathbb{N}} \langle t_j\sigma, d_j \rangle$$

This holds by stability of $\succsim^{\mathbb{N}}$ (Lemma 6.14) and by (6.8).

Stability of $\succ^{max,L}$ and transitivity of $\succsim^{max,L}$ and $\succ^{max,L}$ follow in an analogous way from the corresponding properties of $\succsim^{\mathbb{N}}$ and $\succ^{\mathbb{N}}$.

(iv) $\underline{\succsim^{max,L} \text{ is monotonic}}$

Since $\succsim^{max,L}$ is obviously monotonic on base terms, we only consider the case where $s, t$ are base terms with $s \succsim^{max,L} t$ (i.e., $s \succsim t$) and $F$ is a tuple symbol. One has to show $s' \succsim^{max,L} t'$ for the tuple terms $s' = F(u_1, \ldots, u_{i-1}, s, u_{i+1}, \ldots, u_n)$ and $t' = F(u_1, \ldots, u_{i-1}, t, u_{i+1}, \ldots, u_n)$.

Clearly, $s' \succsim^{max,L} t'$ holds iff

$$\forall \langle v_j, d_j \rangle \in L(t'). \, \exists \langle v_l, d_l \rangle \in L(s'). \, \langle v_l, d_l \rangle \succsim^{\mathbb{N}} \langle v_j, d_j \rangle$$

For $\langle v_j, d_j \rangle = \langle u_p, e_p \rangle$ with $p \in \{1, \ldots, i-1, i+1, n\}$, we choose $\langle v_l, d_l \rangle = \langle v_j, d_j \rangle$ (since $\succsim^{\mathbb{N}}$ is reflexive by Lemma 6.14). For $\langle v_j, d_j \rangle = \langle t, c \rangle$, we choose $\langle v_l, d_l \rangle = \langle s, c \rangle$. By $s \succsim t$ and $c \geq c$, we also have $\langle s, c \rangle \succsim^{\mathbb{N}} \langle t, c \rangle$.

(v) $\succ^{max,L}$ is well founded

Analogous to the proof that the standard multiset extension of an order $\succ$ is well founded iff $\succ$ itself is well founded.

We now illustrate the proof for $\mu = ms$ (the proof for $\mu = dms$ is analogous).

(i) $\succsim^{ms,L}$ and $\succ^{ms,L}$ are compatible, and both $\succsim^{ms,L}$ and $\succ^{ms,L}$ are transitive

We only show $\succ^{ms,L} \circ \succsim^{ms,L} \subseteq \succ^{ms,L}$. The proofs for $\succsim^{ms,L} \circ \succ^{ms,L} \subseteq \succ^{ms,L}$, $\succ^{ms,L} \circ \succ^{ms,L} \subseteq \succ^{ms,L}$, and $\succsim^{ms,L} \circ \succsim^{ms,L} \subseteq \succsim^{ms,L}$ are analogous. For variables, the same reasoning as for $\mu = max$ applies. So let $s = F(\overline{s})$, $t = G(\overline{t})$, and $u = H(\overline{u})$ where $s \succ^{ms,L} t \wedge t \succsim^{ms,L} u$.

Clearly, $s \succ^{ms,L} t$ means that there exists a multiset cover $\langle \gamma_1, \varepsilon_1 \rangle$ such that

$$\begin{array}{l} \forall \langle i, c \rangle \in \pi(F). \, \forall \langle j, d \rangle \in \pi(G). \\ \quad \gamma_1(j) = i \Rightarrow (\texttt{if } \varepsilon_1(i) \texttt{ then } \langle s_i, c \rangle \succsim^{\mathbb{N}} \langle t_j, d \rangle \texttt{ else } \langle s_i, c \rangle \succ^{\mathbb{N}} \langle t_j, d \rangle) \\ \text{and } \exists \langle i, c \rangle \in \pi(F). \neg \varepsilon(i) \end{array}$$

Moreover, $t \succsim^{ms,L} u$ means that there exists a multiset cover $\langle \gamma_2, \varepsilon_2 \rangle$ such that:

$$\begin{array}{l} \forall \langle j, d \rangle \in \pi(G). \, \forall \langle k, e \rangle \in \pi(H). \\ \quad \gamma_2(k) = j \Rightarrow (\texttt{if } \varepsilon_2(j) \texttt{ then } \langle t_j, d \rangle \succsim^{\mathbb{N}} \langle u_k, e \rangle \texttt{ else } \langle t_j, d \rangle \succ^{\mathbb{N}} \langle u_k, e \rangle) \end{array}$$

We now construct a multiset cover $\langle \gamma_3, \varepsilon_3 \rangle$ to show that $s \succ^{ms,L} u$ holds as well. Let $\gamma_3(k) = \gamma_1(\gamma_2(k))$ and let

$$\varepsilon_3(i) = \begin{cases} \mathit{true}, & \text{if } \varepsilon_1(i) \wedge i = \gamma_1(j) \text{ for some } j \text{ with } \varepsilon_2(j) \\ \mathit{false}, & \text{otherwise} \end{cases}$$

To see that $\langle\gamma_3, \varepsilon_3\rangle$ actually is a multiset cover showing $s \succ^{ms,L} u$, we need to prove:

(a) For each $1 \leq i \leq n$, if $\varepsilon_3(i)$ then $\{\, k \mid \gamma_3(k) = i \,\}$ is a singleton set.

This holds by construction.

(b) $\forall\langle i, c\rangle \in \pi(F).\, \forall\langle k, e\rangle \in \pi(H).$
$\gamma_3(k) = i \Rightarrow (\texttt{if}\ \ \varepsilon_3(i)\ \ \texttt{then}\ \ \langle s_i, c\rangle \succsim^{\mathbb{N}} \langle u_k, e\rangle\ \ \texttt{else}\ \ \langle s_i, c\rangle \succ^{\mathbb{N}} \langle u_k, e\rangle)$

Let $\gamma_3(k) = i$. Then $\exists j.\ j = \gamma_2(k) \wedge i = \gamma_1(j)$. Hence, we have:

$$\langle s_i, c\rangle \mathrel{{}_{(}\!\succsim_{)}^{\mathbb{N}}} \langle t_j, d\rangle \mathrel{{}_{(}\!\succsim_{)}^{\mathbb{N}}} \langle u_k, e\rangle$$

where $x \mathrel{{}_{(}\!\succsim_{)}^{\mathbb{N}}} y$ iff $x \succ^{\mathbb{N}} y$ or $x \succsim^{\mathbb{N}} y$.
By transitivity of $\succsim^{\mathbb{N}}$, then $\varepsilon_1(i) \wedge \varepsilon_2(j)$ implies $\langle s_i, c\rangle \succsim^{\mathbb{N}} \langle u_k, e\rangle$. Likewise, by transitivity of $\succ^{\mathbb{N}}$ and compatibility of $\succsim^{\mathbb{N}}$ and $\succ^{\mathbb{N}}$, $\neg(\varepsilon_1(i) \wedge \varepsilon_2(j))$ implies $\langle s_i, c\rangle \succ^{\mathbb{N}} \langle u_k, e\rangle$.
Since $\langle\gamma_1, \varepsilon_1\rangle$ and $\langle\gamma_2, \varepsilon_2\rangle$ are multiset covers, $\varepsilon_1(i) \wedge \varepsilon_2(j)$ holds iff $\varepsilon_3(i)$ holds, which shows the proposition.

(c) $\exists i.\neg\varepsilon_3(i)$.

This holds due to $\exists i.\neg\varepsilon_1(i)$ and $\neg\varepsilon_1(i) \Rightarrow \neg\varepsilon_3(i)$.

(ii) $\underline{\succsim^{ms,L} \text{ and } \succ^{ms,L} \text{ are stable, and } \succsim^{ms,L} \text{ is reflexive.}}$

This follows from the corresponding properties of $\succsim^{\mathbb{N}}$, of $\succ^{\mathbb{N}}$, and of syntactic equality on terms.

(iii) $\underline{\succsim^{ms,L} \text{ is monotonic}}$

Since $\succsim^{ms,L}$ is obviously monotonic on base terms, we only consider the case where $s, t$ are base terms with $s \succsim^{ms,L} t$ (i.e., $s \succsim t$), where $F$ is a tuple symbol, and one has to show $s' \succsim^{ms,L} t'$ for the tuple terms $s' = F(u_1, \ldots, u_{i-1}, s, u_{i+1}, \ldots, u_n)$ and $t' = F(u_1, \ldots, u_{i-1}, t, u_{i+1}, \ldots, u_n)$. Here the multiset cover $\langle\gamma, \varepsilon\rangle$ with $\forall i'.\varepsilon(i') \wedge \gamma(i') = i'$ implies $s' \succsim^{ms,L} t'$.

(iv) $\underline{\succ^{ms,L} \text{ is well founded}}$

This can be shown via an adaption of the proof that the standard multiset extension of an order $\succ$ is well founded iff $\succ$ itself is well founded. We additionally need to use transitivity and compatibility of $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$.

□

We now automate the SCNP criterion of [BC08]. For a DP problem $(\mathcal{P}, \mathcal{R})^{tt}$ satisfying the tuple property, we have to find a suitable base order $(\succsim, \succ)$ to construct the size-change graphs $\mathcal{G}$ corresponding to the DPs in $\mathcal{P}$. So every graph $(V_s, V_t, E)$ from $\mathcal{G}$ with source vertices $V_s = \{F_1, \ldots, F_n\}$ and target vertices $V_t = \{G_1, \ldots, G_m\}$ corresponds to a DP $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m)$. Moreover, we have an edge $(F_i, G_j, \succ) \in E$ iff $s_i \succ t_j$ and $(F_i, G_j, \succsim) \in E$ iff $s_i \succsim t_j$.

In our example, if we use the reduction pair $(\succsim, \succ)$ based on the embedding order, then $\mathcal{G}$ are the size-change graphs from Figure 6.2**(a)**. For instance, the first size-change graph results from the DP (6.3).

For SCNP, we have to extend $\succ$ to a well-founded order $\sqsupset$ which can also compare tuple terms and which satisfies all size-change graphs in $\mathcal{G}$. For $\sqsupset$, we could take any order $\succ^{\mu,L}$ from an SCNP reduction pair $(\succsim^{\mu,L}, \succ^{\mu,L})$. To show that $\sqsupset$ satisfies the size-change graphs from $\mathcal{G}$, one then has to prove $F(s_1, \ldots, s_n) \succ^{\mu,L} G(t_1, \ldots, t_m)$ for every DP $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m)$. Moreover, to ensure that the size-change graphs correctly describe the transitions of the TRS-program $\mathcal{R}$, one also has to require that all rules of the TRS $\mathcal{R}$ are weakly decreasing w.r.t. $\succsim$ (cf. the remarks at the beginning of Section 6.3). Of course, as in [TG05], this requirement can be weakened (e.g., by only regarding *usable* rules) when proving *innermost* termination.

As in [BC08], we define $\sqsupset$ via a lexicographic combination of several orders having the form $\succ^{\mu,L}$. We define the lexicographic combination of two reduction pairs as follows:

$$(\succsim_1, \succ_1) \times (\succsim_2, \succ_2) = (\succsim_{1\times2}, \succ_{1\times2}) \tag{6.9}$$

Here, $s \succsim_{1\times2} t$ holds iff both $s \succsim_1 t$ and $s \succsim_2 t$. Moreover, $s \succ_{1\times2} t$ holds iff $s \succ_1 t$ or both $s \succsim_1 t$ and $s \succ_2 t$. It is clear that $(\succsim_{1\times2}, \succ_{1\times2})$ is again a reduction pair.

A suitable well-founded order $\sqsupset$ is now constructed automatically as follows. The pair of orders $(\sqsupseteq, \sqsupset)$ is initialized by defining $\sqsupseteq$ to be the relation where only $t \sqsupseteq t$ holds for two tuple or base terms $t$ and where $\sqsupset$ is the empty relation. As long as the set of size-change graphs $\mathcal{G}$ is not empty, a status $\mu$ and a level mapping $L$ are synthesized such that $(\succsim^{\mu,L}, \succ^{\mu,L})$ orients all DPs weakly and at least one DP strictly. In other words, the corresponding ranking function satisfies one size-change graph and "weakly satisfies" the others. Then the strictly oriented DPs (corresponding to the strictly satisfied size-change graphs) are removed, and $(\sqsupseteq, \sqsupset) := (\sqsupseteq, \sqsupset) \times (\succsim^{\mu,L}, \succ^{\mu,L})$ is updated. In this way, the SCNP approach can be simulated by a repeated application of the *reduction pair processor* in the DP framework, using the special class of SCNP reduction pairs.

So in our example, we could first look for a $\mu_1$ and $L_1$ where the first DP (6.3) decreases strictly (w.r.t. $\succ^{\mu_1,L_1}$) and the second decreases weakly (w.r.t. $\succsim^{\mu_1,L_1}$). Then we would remove the first DP and could now search for a $\mu_2$ and $L_2$ such that the remaining second DP (6.4) decreases strictly (w.r.t. $\succ^{\mu_2,L_2}$). The resulting reduction pair would then be

$(\sqsupseteq, \sqsupset) = (\gtrsim^{\mu_1,L_1}, \succ^{\mu_1,L_1}) \times (\gtrsim^{\mu_2,L_2}, \succ^{\mu_2,L_2})$.

While in [BC08], the set of size-change graphs remains fixed throughout the whole termination proof, the DP framework allows to use a lexicographic combination of SCNP reduction pairs which are constructed from *different* reduction pairs $(\gtrsim, \succ)$ on base terms. In other words, after a base order and a ranking function satisfying one size-change graph and weakly satisfying all others have been found, the satisfied size-change graph (i.e., the corresponding DP in the DP problem at hand) is removed, and one can synthesize a possibly different ranking function and also a *possibly different base order* for the remaining DPs (i.e., different abstractions to different size-change graphs can be used in one and the same termination proof).

**Example 6.22.** We add a third rule to the TRS from Example 6.3: $\mathsf{f}(\mathsf{c}(x), y) \to \mathsf{f}(x, \mathsf{s}(x))$. Now no SCNP reduction pair based only on the embedding order can orient all DPs strictly at the same time anymore, even if one permits combinations with arbitrary argument filters. However, we can first apply an SCNP reduction pair that sets all tags to 0 and uses the embedding order together with an argument filter to collapse the symbol $\mathsf{s}$ to its argument. Then the DP for the newly added rule is oriented strictly and all other DPs are oriented weakly. After removing the new DP, the SCNP reduction pair that we already used for the DPs of Example 6.3 again orients all DPs strictly. Note that the base order for this second SCNP reduction pair is the embedding order without argument filters, i.e., it differs from the base order used in the first SCNP reduction pair.

By representing the SCNP method via SCNP reduction pairs, we can now benefit from the flexibility of the DP framework. Thus, we can use other termination methods in addition to SCNP. More precisely, as usual in the DP framework, we can apply arbitrary processors one after another in a modular way. This allows us to interleave arbitrary other termination techniques with termination proof steps based on size-change termination, whereas in [TG05], size-change proofs can only be used as a final step in a termination proof.

## The Need for Tuple Typing

So far, we have substantiated the claim that reduction pairs in an *untyped* setting are not a suitable modeling for SCNP by intuition, but not yet by a proper counterexample. For this purpose, consider the following example which shows that an SCNP reduction pair is not monotonic in general:

**Example 6.23** (SCNP Reduction Pairs Are Not Monotonic in an Untyped Setting)**.** Consider a TRS $\mathcal{R} = \{\, \mathsf{f}(x) \to \mathsf{g}(x), \mathsf{g}(x) \to \mathsf{a} \,\}$. We get $DP(\mathcal{R}) = \{\, \mathsf{F}(x) \to \mathsf{G}(x) \,\}$. Moreover consider a base reduction pair $(\gtrsim_{\mathcal{A}}, \succ_{\mathcal{A}})$ induced by a polynomial interpretation $[\cdot]_{\mathcal{A}}$ with $[\mathsf{F}](x_1) = x_1$ and $[\mathsf{G}](x_1) = x_1 + 1$. We use a level mapping $L$ induced by

$\pi(\mathsf{F}) = \{\, \langle 1, 0\rangle \,\}$ and $\pi(\mathsf{G}) = \varnothing$ to lift this base reduction pair $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$ to the SCNP "reduction pair" $(\succsim^{max,L}, \succ^{max,L})$.

We then indeed get $\mathsf{F}(x) \succsim^{max,L} \mathsf{G}(x)$ (because $L(\mathsf{G}(x)) = \varnothing$). However, $\mathsf{F}(\mathsf{F}(x)) \succsim^{max,L} \mathsf{F}(\mathsf{G}(x))$ does *not* hold (because $x \not\geq x + 1$).

One may also wonder if the literature does not already provide more suitable means than reduction *pairs* (which require monotonicity for the weak order used for the comparison of DPs) to capture the orders induced by SCNP. Indeed, in related work [HM07], Hirokawa and Middeldorp introduce the notion of *reduction triples* $(\succsim, \geqslant, >)$. For a DP problem $(\mathcal{P}, \mathcal{R})$, here one may use potentially *different* weak orders $\succsim$ and $\geqslant$ for comparison of rules from $\mathcal{R}$ and from $\mathcal{P}$, respectively. Here $\geqslant$ does not need to be monotonic. This makes reduction triples look like a more suitable candidate for formalization of SCNP in the DP framework.

However, [HM07] still requires a form of compatibility also for $\succsim$ and $>$, i.e., that one of $\succsim \circ > \subseteq >$ or $> \circ \succsim \subseteq >$ must hold. As René Thiemann [Thi11] points out, this property does not hold in general.

**Example 6.24** (SCNP Does Not Induce Untyped Reduction Triples [Thi11])**.** Consider terms $s = \mathsf{F}(x, y)$ and $t = \mathsf{F}(x, \mathsf{s}(y))$, a polynomial interpretation $[\cdot]_{\mathcal{A}}$ with $[\mathsf{F}](x_1, x_2) = x_1$ and $[\mathsf{s}](x_1) = x_1 + 1$ giving rise to a base reduction pair $(\succsim_{\mathcal{A}}, \succ_{\mathcal{A}})$, and a level mapping $L$ induced by $\pi(\mathsf{F}) = \{\, \langle 2, 1\rangle \,\}$. Then one might assume that $(\succsim_{\mathcal{A}}, \succsim^{max,L}, \succ^{max,L})$ was a reduction triple.

However, then we have $s \succsim_{\mathcal{A}} t$ (because $[s]_{\mathcal{A}} = x \geq x = [t]_{\mathcal{A}}$) and $t \succ^{max,L} s$ (because $\langle \mathsf{s}(y), 1\rangle \succ^{\mathbb{N}} \langle y, 1\rangle$). But neither $s \succ^{max,L} s$ nor $t \succ^{max,L} t$ hold, which refutes the required compatibility property.

As these examples indicate, a novel processor is required to integrate SCNP into the DP framework. Via the switches from untyped to tuple-typed DP problems and back, we can restrict the set of well-typed terms in such a way that for these terms, SCNP indeed induces a reduction pair.

So in practice, the user of a termination technique is not even required to become aware of the switches between untyped and tuple-typed setting. To use SCNP reduction pairs as a component of the usual reduction pair processor of [GTSF06] along with its extensions, it suffices to require that the input DP problem has the tuple property, which in practice holds in most cases. The switches between untyped and tuple-typed settings can then be performed implicitly. This way, the integration of SCNP is quite seamless, and we do not need to modify any existing processors.

## 6.4 A Challenge Example

In this section, as an extended example we present a TRS where all existing termination tools fail, but where termination can easily be proved automatically by an SCNP reduction pair with a *matrix order* [EWZ08] as a base order.

**Example 6.25.** Consider the following rewrite system $\mathcal{R}_1$, which is taken from the termination problem data base (`TRS/Zantema_06/03.xml`).

$$\mathsf{a}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x)))) \to \mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(\mathsf{a}(\mathsf{a}(x))))))) \tag{6.10}$$

$$\mathsf{a}(\mathsf{c}(x)) \to \mathsf{c}(\mathsf{a}(x)) \tag{6.11}$$

$$\mathsf{c}(\mathsf{b}(x)) \to \mathsf{b}(\mathsf{c}(x)) \tag{6.12}$$

When attempting to prove termination by the standard techniques of the DP framework (such as the dependency graph processor or the reduction pair processor with matrix orders over the naturals), the initial DP problem is eventually transformed into the DP problem $(\mathcal{P}_1, \mathcal{R}_1)$ with $\mathcal{P}_1 = \{\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x)))) \to \mathsf{A}(\mathsf{a}(x))\}$.

We can conclude the termination proof of $\mathcal{R}_1$ by deleting the DP of $\mathcal{P}_1$ using the reduction pair processor. To this end, we can use the following (standard) matrix order [EWZ08] based on an interpretation $[\cdot]_{\mathcal{M}}$ to the carrier $\mathbb{N}^4$ and matrix entries from $\{0, 1\}$.

$$[\mathsf{A}]_{\mathcal{M}}(x_1) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot x_1$$

$$[\mathsf{a}]_{\mathcal{M}}(x_1) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot x_1$$

$$[\mathsf{b}]_{\mathcal{M}}(x_1) = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot x_1 + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$[\mathsf{c}]_{\mathcal{M}}(x_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot x_1$$

This example can easily be modified into an example where an SCNP reduction pair is

required. The idea is to exploit that SCNP allows us to perform a "max comparison" on the level of the tuple symbols of the dependency pairs. This is a feature that many base orders like matrix orders over the naturals cannot provide.

To this end, we introduce a unary function symbol $\mathsf{dup}$ and a binary function symbol $\mathsf{collapse}$. Then we add a rule where $\mathsf{dup}$ applied to the left-hand side of the DP in $\mathcal{P}_1$ rewrites to *two copies* of the corresponding right-hand side of the DP, wrapped inside the symbol $\mathsf{collapse}$.

$$\mathsf{dup}(\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x))))) \to \mathsf{collapse}(\mathsf{A}(\mathsf{a}(x)), \mathsf{A}(\mathsf{a}(x))) \tag{6.13}$$

We also have to make sure that we can still continue the rewrite sequence as before. So we add rules to eliminate any of the copies again.

$$\mathsf{collapse}(x, y) \to \mathsf{dup}(x) \tag{6.14}$$
$$\mathsf{collapse}(x, y) \to \mathsf{dup}(y) \tag{6.15}$$

We now unite the rules from $\mathcal{R}_1$ with these three new rules and obtain a TRS $\mathcal{R}_2$ that turns out to be very challenging for current automated termination tools.

$$\mathsf{a}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x)))) \to \mathsf{b}(\mathsf{b}(\mathsf{b}(\mathsf{a}(\mathsf{a}(\mathsf{a}(x))))))) \tag{6.10}$$
$$\mathsf{a}(\mathsf{c}(x)) \to \mathsf{c}(\mathsf{a}(x)) \tag{6.11}$$
$$\mathsf{c}(\mathsf{b}(x)) \to \mathsf{b}(\mathsf{c}(x)) \tag{6.12}$$
$$\mathsf{dup}(\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x))))) \to \mathsf{collapse}(\mathsf{A}(\mathsf{a}(x)), \mathsf{A}(\mathsf{a}(x))) \tag{6.13}$$
$$\mathsf{collapse}(x, y) \to \mathsf{dup}(x) \tag{6.14}$$
$$\mathsf{collapse}(x, y) \to \mathsf{dup}(y) \tag{6.15}$$

When attempting the termination proof in the DP framework, after a few proof steps, we get the DP problem $(\mathcal{P}_2, \mathcal{R}_2)$ where $\mathcal{P}_2$ consists of the following dependency pairs.

$$\mathsf{DUP}(\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x))))) \to \mathsf{COLLAPSE}(\mathsf{A}(\mathsf{a}(x)), \mathsf{A}(\mathsf{a}(x))) \tag{6.16}$$
$$\mathsf{COLLAPSE}(x, y) \to \mathsf{DUP}(x) \tag{6.17}$$
$$\mathsf{COLLAPSE}(x, y) \to \mathsf{DUP}(y) \tag{6.18}$$

Now we can delete the first dependency pair using an SCNP reduction pair with the status $\mu = max$. This reduction pair uses the previous matrix order $\mathcal{M}$ when comparing base terms. For comparing tuple terms, it uses a level mapping $L$ induced by with $\pi(\mathsf{DUP}) = \{\langle 1, 0\rangle\}$ and $\pi(\mathsf{COLLAPSE}) = \{\langle 1, 0\rangle, \langle 2, 0\rangle\}$ (i.e., here we do not really need the tags in the level mapping). Thus, the first DP is strictly decreasing because for every term in the multiset $\{\mathsf{A}(\mathsf{a}(x)), \mathsf{A}(\mathsf{a}(x))\}$ of arguments on the right-hand side, there

is a strictly larger term in the multiset $\{\mathsf{A}(\mathsf{a}(\mathsf{b}(\mathsf{b}(x))))\}$ of arguments on the left-hand side. After this proof step, applying the dependency graph processor on the resulting DP problem concludes the termination proof for $\mathcal{R}_2$.

Note that without SCNP reduction pairs, termination of this example is not easy to prove. When disabling SCNP reduction pairs, AProVE can no longer accomplish the termination proof and we aborted the proof attempt after 15 minutes. Here, we used the reduction pair processor with reduction pairs from the regarded class of base orders (i.e., matrix orders over the naturals with matrices of dimension 4 and entries from $\{0, 1\}$). Indeed, none of the tools that participated in the Termination Competition 2009 for standard term rewriting succeeded in finding a termination proof for $\mathcal{R}_2$. This shows that even for very powerful classes of orders like matrix orders, it is not difficult to come up with examples where SCNP reduction pairs enable an automated termination proof whereas none could be found otherwise. While this example is arguably rather a hand-crafted challenge example than a real-life application problem, it does provide an intuition for settings where the contributions of this chapter may be particularly helpful. SCNP reduction pairs allow to search for sophisticated abstractions for base terms as arguments for function calls and at the same time for suitable multiset comparisons of these—a priori arbitrary—abstractions on top.

This SCNP-based termination proof has been formally verified using the Isabelle-based certification tool CeTA [TS09] as a termination proof via the earlier adaption of *size-change termination* by Thiemann and Giesl [TG05]. Here we use that also the combination of an SCNP proof step with a (tagged) level mapping and a final step using simple dependency graph approximations (denoted *numeric level mappings* in [BC08]) are still subsumed by size-change termination. This is necessary since a classic size-change termination proof can only be used as a final step in the certification setting, i.e., all DPs must be deleted from the DP problem.

In very recent work, Thiemann *et al.* provide a formalization of SCNP reduction pairs as part of the IsaFoR project [TS09], from which the certification tool CeTA is generated by code extraction. To avoid the introduction (and formalization) of types for rewriting in IsaFoR, this formalization does not make use of tuple-typed DP problems, but instead, a generalization of the reduction pair processor is used.[74] Also using this alternative formalization of SCNP reduction pairs, CeTA certifies correctness of our termination proof.

## 6.5 SAT-Based Automation

Recently, the search problem for many common base orders has been reduced successfully to SAT problems (cf. Chapter 1). In this section, we build on this earlier work and use

[74] For details see also the IsaFoR developer repository at:
`http://cl2-informatik.uibk.ac.at/rewriting/mercurial.cgi/IsaFoR`

these encodings as components for a SAT encoding of SCNP reduction pairs. This way, we benefit from *compositionality* of SAT encodings. The corresponding decision problem is stated as follows:

> For a DP problem $(\mathcal{P},\mathcal{R})^{tt}$ and a given class of base orders, is there a status $\mu$, a level mapping $L$, and a concrete base reduction pair $(\succsim,\succ)$ such that the SCNP reduction pair $(\succsim^{\mu,L},\succ^{\mu,L})$ orients all rules of $\mathcal{R}$ and $\mathcal{P}$ weakly and at least one of $\mathcal{P}$ strictly?

We assume a given *base SAT encoding* $||.||_{base}$ which maps base term constraints of the form $s\ {}_{(}\!\succsim_{)}\ t$ to propositional formulas. Every satisfying assignment for the formula $||s\ {}_{(}\!\succsim_{)}\ t||_{base}$ corresponds to a particular order where $s\ {}_{(}\!\succsim_{)}\ t$ holds.

We also assume a given encoding for partial orders (on tags), cf. [CLS08]. The function $||.||_{po}$ maps partial order constraints of the form $n_1 \geq n_2$ or $n_1 > n_2$ where $n_1$ and $n_2$ represent natural numbers (in some fixed number of bits) to corresponding propositional formulas on the bit representations for the numbers.

For brevity, we only show the encoding for SCNP reduction pairs $(\succsim^{\mu,L},\succ^{\mu,L})$ where $\mu = max$. The encodings for the other cases are similar: The encoding for the min comparison is completely analogous. To encode (dual) multiset comparison one can adapt previous approaches to encode multiset orders [STA+07, Sch08, CGST12].

First, for each tuple symbol $F$ of arity $n$, we introduce natural number variables denoted $tag_i^F$ for $1 \leq i \leq n$. These encode the tags associated with the argument positions of $F$ by representing them in a fixed number of bits. In our case, it suffices to consider tag values which are less than the sum of the arities of the tuple symbols. In this way, every argument position of every tuple symbol could get a different tag, i.e., this suffices to represent all possible level mappings.

Now consider a size-change graph corresponding to a DP $\delta = s \to t$ with $s = F(s_1,\ldots,s_n)$ and $t = G(t_1,\ldots,t_m)$. The edges of the size-change graph are determined by the base order, which is not fixed. For any $1 \leq i \leq n$ and $1 \leq j \leq m$, we define a propositional formula $weak_{i,j}^{\delta}$ which is *true* iff $\langle s, tag_i^F\rangle \succsim^{\mathbb{N}} \langle t, tag_j^G\rangle$. Similarly, $strict_{i,j}^{\delta}$ is *true* iff $\langle s, tag_i^F\rangle \succ^{\mathbb{N}} \langle t, tag_j^G\rangle$. The definition of $weak_{i,j}^{\delta}$ and $strict_{i,j}^{\delta}$ corresponds directly to Definition 6.13. It is based on the encodings $||.||_{base}$ and $||.||_{po}$ for the base order and for the tags, respectively.

$$\begin{aligned} weak_{i,j}^{\delta} &= ||s_i \succ t_j||_{base} \ \vee\ (||s_i \succsim t_j||_{base} \wedge ||tag_i^F \geq tag_j^G||_{po}) \\ strict_{i,j}^{\delta} &= ||s_i \succ t_j||_{base} \ \vee\ (||s_i \succsim t_j||_{base} \wedge ||tag_i^F > tag_j^G||_{po}) \end{aligned}$$

To facilitate the search for level mappings, for each tuple symbol $F$ of arity $n$ we introduce propositional variables $reg_i^F$ for $1 \leq i \leq n$. Here, $reg_i^F$ is *true* iff the $i$-th argument position of $F$ is regarded for comparison. The formulas $||s \succsim^{max,L} t||$ and $||s \succ^{max,L} t||$ then encode

that the DP $s \to t$ can be oriented weakly or strictly, respectively. By this encoding, one can simultaneously search for a base order that gives rise to the edges in the size-change graph and for a level mapping that satisfies this size-change graph.

$$\begin{aligned} ||s \succsim^{max,L} t|| &= \bigwedge_{1\le j\le m} (reg_j^G \to \bigvee_{1\le i\le n} (reg_i^F \wedge weak_{i,j}^{\delta})) \\ ||s \succ^{max,L} t|| &= \bigwedge_{1\le j\le m} (reg_j^G \to \bigvee_{1\le i\le n} (reg_i^F \wedge strict_{i,j}^{\delta})) \wedge \bigvee_{1\le i\le n} reg_i^F \end{aligned}$$

For any DP problem $(\mathcal{P}, \mathcal{R})^{tt}$ we can now generate a propositional formula which ensures that the corresponding SCNP reduction pair orients all rules from $\mathcal{R}$ and $\mathcal{P}$ weakly and at least one rule from $\mathcal{P}$ strictly:

$$\bigwedge_{l\to r\in\mathcal{R}} ||l \succsim r||_{base} \quad \wedge \quad \bigwedge_{s\to t\in\mathcal{P}} ||s \succsim^{max,L} t|| \quad \wedge \quad \bigvee_{s\to t\in\mathcal{P}} ||s \succ^{max,L} t||$$

Similar to [CGST12, ZHM09], our approach is easily extended to refinements of the DP method where one only regards the *usable* rules of $\mathcal{R}$ and where these usable rules can also depend on the (explicit or implicit) argument filter of the order.

## 6.6 Experiments

We implemented our contributions in the automated termination prover AProVE [GST06]. To assess their impact, we compared three configurations of AProVE. In the first configuration, we use SCNP reduction pairs in the reduction pair processor of the DP framework. This configuration is parameterized by the choice whether we allow just max comparisons of multisets or all four multiset extensions from Definition 6.15. Moreover, the configuration is also parameterized by the choice whether we use classical size-change graphs or *extended* size-change graphs as in [TG05]. In an extended size-change graph, to compare $s = F(s_1, \ldots, s_n)$ with $t = G(t_1, \ldots, t_m)$, the source and target vertices $\{s_1, \ldots, s_n\}$ and $\{t_1, \ldots, t_m\}$ are extended by additional vertices $s$ and $t$, respectively. Now an edge from $s$ to $t_j$ indicates that the whole term $s$ is greater (or equal) to $t_j$, etc. So these additional vertices also allow us to compare the whole terms $s$ and $t$. By adding these vertices, size-change termination incorporates the standard comparison of terms as well.

In the second configuration, we use the base orders directly in the reduction pair processor (i.e., here we disregard SCNP reduction pairs). In the third configuration, we use the implementation of the SCT method as described in [TG05]. For a fair comparison, we updated this old implementation from the DP *approach* to the modular DP *framework* and used SAT encodings for the base orders. (While this approach only uses the embedding order and argument filters as the base order for the construction of size-change graphs, it uses more complex orders (containing the base order) to weakly orient the rules

| order | | SCNP fast | SCNP max | SCNP all | reduction pairs | SCT [TG05] |
|---|---|---|---|---|---|---|
| EMB | proved | 346 | 346 | **347** | 325 | 341 |
| | runtime | **2882.6** | 3306.4 | 3628.5 | 2891.3 | 10065.4 |
| LPO | proved | 500 | **530** | 527 | 505 | 385 |
| | runtime | **3093.7** | 5985.5 | 7739.2 | 3698.4 | 10015.5 |
| RPO | proved | 501 | **531** | **531** | 527 | 385 |
| | runtime | **3222.2** | 6384.1 | 8118.0 | 4027.5 | 10053.4 |
| POLO | proved | 477 | **514** | **514** | 511 | 378 |
| | runtime | 3153.6 | 5273.6 | 7124.4 | **2941.7** | 9974.0 |

Figure 6.26: Comparison of SCNP reduction pairs to SCT and direct reduction pairs.

from the TRS.)

We considered all 1381 examples from the standard TRS category of the TPDB, version 7.0.2, as used in the International Termination Competition 2009. The experiments were run on a 2.66 GHz Intel Core 2 Quad and we used a time limit of 60 seconds per example. We applied SAT4J [LP10] to transform propositional formulas to conjunctive normal form and the SAT solver MiniSAT2 [ES04] to check the satisfiability of the resulting formulas.

Figure 6.26 compares the power and runtimes of the three configurations depending on the base order. The column "*order*" indicates the base order: embedding order with argument filters (EMB), lexicographic path order with arbitrary permutations and argument filters (LPO), recursive path order with argument filters (RPO), and linear polynomial interpretations with coefficients from $\{0, 1\}$ (POLO). For the first configuration, we used three different settings: full SCNP reduction pairs with extended size-change graphs ("SCNP all"), SCNP reduction pairs restricted to max-comparisons with extended size-change graphs ("SCNP max"), and SCNP reduction pairs restricted to max comparisons and non-extended size-change graphs ("SCNP fast"). The second and third configuration are called "reduction pairs" and "SCT [TG05]", respectively. For each experiment, we give the number of TRSs which could be proved terminating ("proved") and the analysis time in seconds for running AProVE on all 1381 TRSs ("runtime"). The "best" numbers are always printed in **bold**. For further details on the experiments, we refer to `http://aprove.informatik.rwth-aachen.de/eval/SCNP`. The table allows the following observations:

**(1)** Our SCNP reduction pairs are *much more powerful and significantly faster* than the implementation of [TG05]. By integrating the search for the base order with SCNP, our new implementation can use a much larger class of base orders and thus, SCNP reduction pairs can prove significantly more examples. The reason for the relatively low speed of [TG05] is that this approach iterates through argument filters and then generates and analyzes size-change graphs for each of these argument filters. (So the low speed is not due to the repeated composition of size-change graphs in the SCT criterion.)

**(2)** Our new implementation of SCNP reduction pairs is *more powerful* than using the re-

duction pairs directly. Note that when using extended size-change graphs, *every* reduction pair can be simulated by an SCNP reduction pair.
**(3)** SCNP reduction pairs *add significant power when used for simple orders* like EMB and LPO. The difference is less dramatic for RPO and POLO. Intuitively, the reason is that SCNP allows for multiset comparisons which are lacking in EMB and LPO, while RPO contains multiset comparisons and POLO can often simulate them. Nevertheless, SCNP also adds some power to RPO and POLO, e.g., by extending them by a concept like "maximum". This even holds for more powerful base orders like matrix orders [EWZ08] (cf. Example 6.25).

## 6.7 Summary and Outlook

We show that the practically relevant part of size-change termination (SCNP) can be formulated as a reduction pair. Thus, SCNP can be applied in the DP framework, which is used in virtually all termination tools for term rewriting. The requirement that the DP problem satisfies the tuple property is only a very small restriction in practice.

Moreover, by combining the search for the base order and for the SCNP level mapping into one search problem, we can automatically find the right base order for constructing size-change graphs. Thus, we now generate program abstractions automatically such that termination of the abstracted programs can be shown.

The implementation in AProVE confirms the usefulness of our contribution. Our experiments indicate that the automation of our technique is more powerful than both the direct use of reduction pairs and the SCT adaptation from [TG05].

As part of the integration of SCNP into the DP framework, we develop the notion of a tuple-typed DP problem and show that for most DP problems arising in practice (i.e., those which satisfy the tuple property), a switch between untyped and tuple-typed DP problems is transparently possible. The advantage is that termination techniques can assume that certain undesired terms need not be considered for termination analysis. This facilitates the adaption of termination techniques—such as SCNP—designed for other formalisms into the DP framework.

One obvious direction of future research is to lift Theorem 6.20 from SCNP reduction pairs to reduction pairs for *full* size-change termination. Here Lee's result [Lee09] that also full size-change termination can be expressed equivalently via a class of ranking functions would be a good starting point. This line of research is probably interesting primarily from a theoretical perspective, to get a deeper understanding of the relation between size-change termination and the dependency pair framework. The reason is that following [BC08], SCNP almost always suffices in practice instead of full size-change termination. Moreover, the expressions for ranking functions corresponding to full size-change termination can be exponentially large, which could render this approach prohibitive in practice.

Another possible line of research could have more direct applications. Codish, Lagoon, and Stuckey [CLS05] present an extension of size-change termination from the classic well-founded setting to the non-well-founded setting of the *integers*, called *monotonicity constraints*. As with size-change termination, also here the decision problem if the abstraction of a program terminates according to the method is PSPACE-complete. In very recent work [CGB+11], we identify a subset of this termination method called *Monotonicity Constraints in NP (MCNP)* which is NP-complete. We present an automation by a SAT encoding and show how to use MCNP for termination analysis of Java Bytecode programs. However, for the MCNP setting the abstraction process is more involved, and we perform termination analysis using the two-stage approach (i.e., in [CGB+11] we lift [BC08] to the integer setting). We first compute a fixed abstraction and only afterwards analyze if we can show termination with MCNP for this given abstraction. Here, it is not as clear as for SCNP how to encode the search for an abstraction and the search for a ranking function which implies termination into a single SAT instance.

# 7 SAT Encodings for Optimal XOR Circuits

In the previous chapters, we have seen that SAT encodings can render even NP-complete problems algorithmically feasible for many practical instances arising in automated termination analysis. One may wonder, though, if this effect is confined to termination analysis. Moreover, the question is to what extent one can transfer methodology surrounding SAT-based problem solving from termination analysis to other application domains. Building on this line of thought, we have reached the hypothesis that there are indeed significant synergies to be gained also for applications outside the area of termination analysis. In this chapter, we put this hypothesis to the test and transfer methodology from automated termination analysis to *synthesis* of optimal programs for a given specification. To get a workable approach even with respect to the size of the resulting formula, such an approach obviously requires severe restrictions on the class of programs in question to be feasible.

Here, we have opted for the class of linear straight-line programs over the Galois field of two elements. Such programs are an equivalent representation of Boolean circuits that consist only of XOR gates. Finding the shortest linear straight-line program for a given set of linear forms is known to be MaxSNP-complete, i.e., there is no $\varepsilon$-approximation scheme for the problem unless P = NP.

This chapter presents a non-approximative approach for finding the shortest linear straight-line program. In other words, we show how to search for a circuit of XOR gates with the minimal number of such gates. The approach is based on a reduction of the associated decision problem ("Is there a program of length $k$?") to satisfiability of propositional logic. Using modern SAT solvers, optimal solutions to interesting problem instances can be obtained.

Straight-line programs over the Galois field of two elements, often denoted GF(2), have many practically relevant applications. The most prominent ones are probably in high performance computing (inversion of sparse binary matrices), networking and storage (error detection by checksumming), and encryption (hashing, symmetric ciphers).

In this chapter, we focus on linear straight-line programs over GF(2) with applications in cryptography. The motivation behind this choice is that modern symmetric ciphers like AES can be implemented by lookup tables and addition in GF(2). Multiplication and addition in GF(2) correspond to the Boolean AND and XOR operations, respectively.

In other words, we are looking at straight-line programs composed of array lookups and sequences of XOR operations.

The goal of this chapter is, given a specification of a linear function from a number of inputs to a number of outputs, to find the shortest linear straight-line program over GF(2) that satisfies the specification. In other words, we show how to find a XOR circuit with the minimal number of gates that connects inputs to outputs. Finding such shortest programs is obviously interesting both for software and for hardware implementations of, for example, the symmetric cipher *Advanced Encryption Standard (AES)* [Fed01].

While there are heuristic methods for finding short straight-line linear programs [BP10] (see also [BP09] for the corresponding patent application), to the best of our knowledge, there has not been any feasible method for finding an optimal solution prior to this work. In this chapter, we present an approach based on reducing the associated decision problem ("Is there a program of length $k$?") to satisfiability of propositional logic. The reduction is performed in a way that every model found by the SAT solver represents a solution. Recent work [KKY09] has shown that reductions to satisfiability problems are a promising approach for circuit synthesis. By restricting our attention to linear functions, we now obtain a polynomial-size encoding.

The structure of this chapter is as follows. In Section 7.1, we formally introduce our optimization problem and show how linear straight-line programs can be used to compute a given set of linear forms. Section 7.2 presents a novel encoding for the associated decision problem to SAT. Then, we discuss in Section 7.3 how to tackle our optimization problem by reducing it to the associated decision problem using a customized search for $k$.

In Section 7.4 we present an empirical case study where we optimize an important component of AES. To prove optimality of the solution found, the case study prompts us to improve the performance of our encoding for the decision problem in the unsatisfiable case, which ultimately allows us to achieve this optimality proof. We discuss different approaches to achieve the needed improvements in Section 7.5. We conclude with a summary of our contributions and an outlook to possible future work in Section 7.6.

## 7.1 Linear Straight-Line Programs

In this chapter, we assume that we have $n$ inputs $x_1, \ldots, x_n$ and $m$ outputs $y_1, \ldots, y_m$. Then the linear function to be computed can be specified by $m$ equations of the following

form:

$$\begin{aligned} y_1 &= a_{1,1} \cdot x_1 \oplus a_{1,2} \cdot x_2 \oplus \ldots \oplus a_{1,n} \cdot x_n \\ y_2 &= a_{2,1} \cdot x_1 \oplus a_{2,2} \cdot x_2 \oplus \ldots \oplus a_{2,n} \cdot x_n \\ &\ldots \\ y_m &= a_{m,1} \cdot x_1 \oplus a_{m,2} \cdot x_2 \oplus \ldots \oplus a_{m,n} \cdot x_n \end{aligned}$$

We call each equation a *linear form*. Note that each $a_{\ell,j}$ is a constant from $\mathrm{GF}(2) = \{0, 1\}$, each $x_j$ is a variable over $\mathrm{GF}(2)$, and for the setting of this chapter "$\oplus$" and "$\cdot$" denote standard addition and multiplication on $\mathrm{GF}(2)$, respectively. In this chapter, we always assume that the linear forms are pairwise different.

Our goal is to come up with an algorithm that computes these linear forms given $x_1, \ldots, x_n$ as inputs. More specifically, we would like to express this algorithm via a *linear straight-line program* (or, for brevity, just *program*). Here, every line of the program has the shape $u := e \cdot v \oplus f \cdot w$ with $e, f \in \mathrm{GF}(2)$ and $v, w$ variables. Some lines of the program will contain the output, i.e., assign the value of one of the desired linear forms to a variable. The *length* of a program is the number of lines the program contains. Without loss of generality, we perform write operations only to fresh variables, so no input is overwritten and no intermediate variable is written to twice. A program is *optimal* if there is no shorter program that computes the same linear forms.

**Example 7.1.** Consider the following linear forms:

$$\begin{array}{lllllllllll} y_1 & = & x_1 & \oplus & x_2 & \oplus & x_3 & \oplus & x_4 & \oplus & x_5 \\ y_2 & = & x_1 & \oplus & x_2 & \oplus & x_3 & \oplus & x_4 & & \\ y_3 & = & x_1 & \oplus & x_2 & \oplus & x_3 & & & \oplus & x_5 \\ y_4 & = & & & & & x_3 & \oplus & x_4 & \oplus & x_5 \\ y_5 & = & x_1 & & & & & & & \oplus & x_5 \end{array}$$

A shortest linear program for computing these linear forms has length 6. The following linear program is an optimal solution for this example.

$$\begin{array}{llllll} v_1 & = & x_1 & \oplus & x_5 & [y_5] \\ v_2 & = & x_2 & \oplus & v_1 & \\ v_3 & = & x_3 & \oplus & v_2 & [y_3] \\ v_4 & = & x_4 & \oplus & v_3 & [y_1] \\ v_5 & = & x_5 & \oplus & v_4 & [y_2] \\ v_6 & = & v_2 & \oplus & v_5 & [y_4] \end{array}$$

It is easy to check that for each output $y_\ell$ there is a variable $v_i$ that contains the linear form for $y_\ell$. In the above program, this mapping from intermediate variables to outputs

is given by annotating the program lines with the associated output in square brackets.

Note that finding the shortest program over GF(2) is *not* an instance of the common subexpression elimination problem known from program optimization. The above shortest program makes extensive use of *cancellation*, i.e., of the fact that for all $x$ in GF(2), we have $x \oplus x = 0$. For example, the output $y_4$ is computed by adding $v_2$ and $v_5$. These two variables are described by the linear forms $x_1 \oplus x_2 \oplus x_3 \oplus x_4$ and $x_1 \oplus x_2 \oplus x_5$, respectively. By adding these two linear forms, we obtain the desired $x_3 \oplus x_4 \oplus x_5$ since $x_1 \oplus x_1 \oplus x_2 \oplus x_2 = 0$ for all values of $x_1$ and $x_2$. Without cancellations, a shortest linear straight-line program has length 8, i.e., it uses 25% more XOR gates.

The goal that we are now pursuing in this chapter is to synthesize an optimal linear straight-line program for a given set of linear forms both automatically and efficiently. Formally, this problem can be described as follows:

> Given $n$ variables $x_1, \ldots, x_n$ over GF(2) and $m$ linear forms $y_\ell = a_{\ell,1} \cdot x_1 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$, find the shortest linear program that computes all $y_\ell$.

Note that here we are aiming at a (provably) *optimal* solution. This is opposed to allowing approximations with more lines than actually necessary, which is the previous state of the art [BMP08].

As a step towards solving this optimization problem, first let us consider the corresponding decision problem:

> Given $n$ variables $x_1, \ldots, x_n$ over GF(2), $m$ linear forms $y_\ell = a_{\ell,1} \cdot x_1 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ and a natural number $k$, decide if there exists a linear program of length $k$ that computes all $y_\ell$.

In [BMP08], Boyar, Matthews, and Peralta show that this problem is NP-complete. Of course, if the answer to this question is "Yes", we do not only wish to get this answer, but we would also like to obtain a corresponding program of length (at most) $k$. In line $i$, the variable $v_i$ is defined as the sum of two other variables. Here, one may read from the variables $x_1, \ldots, x_n$ and also from the intermediate variables $v_1, \ldots, v_j$ with $j < i$, i.e., from those intermediate variables that have been defined before.

The general idea for our approach to obtaining an answer to this question makes use of a concept that surfaces several times in the present thesis (cf., e.g., Definition 2.20): We give a *parametric program* (i.e., in the program we do not have the concrete coefficients 0 or 1, but we have *parametric* coefficients over GF(2) in the form of Boolean variables) and formulate constraints over these parametric coefficients to ensure that only valid solutions are found, i.e., that the resulting instantiation of the parameters yields a concrete program that actually implements the given specification. So the $i^{th}$ line of a parametric program has the shape $v_i = b_{i,1} \cdot x_1 \oplus \ldots \oplus b_{i,n} \cdot x_n \oplus c_{i,1} \cdot v_1 \oplus \ldots \oplus c_{i,i-1} \cdot v_{i-1}$ where $b_{i,1}, \ldots, b_{i,n}, c_{i,1}, \ldots, c_{i,i-1}$ are Boolean variables.

Thus, although at first glance automated termination analysis and program synthesis might not appear to be closely related topics, it turns out that these areas quite conveniently allow for a transfer of methodology.

To facilitate the description of our encoding in the following section, we reformulate the problem via matrices over GF(2). Here, given a natural number $k$, we represent the given coefficients of the $m$ linear forms over $n$ inputs with $y_\ell = a_{\ell,1} \cdot x_1 \oplus a_{\ell,2} \cdot x_2 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ $(1 \leq \ell \leq m)$ as rows of an $m \times n$-matrix $A$. The $\ell$-th row thus consists of the entries $a_{\ell,1} a_{\ell,2} \ldots a_{\ell,n}$ from GF(2).

Likewise, we can also express the resulting program via two matrices:

- A matrix $B = (b_{i,j})_{k \times n}$ over GF(2), where $b_{i,j} = 1$ iff in line $i$ of the program the input variable $x_j$ is read.

- A matrix $C = (c_{k,k})_{k \times k}$ over GF(2) where $c_{i,j} = 1$ iff in line $i$ of the program the intermediate variable $v_j$ is read.

To represent for example the program line $v_3 = x_3 \oplus v_2$ from Example 7.1, all $b_{3,j}$ except for $b_{3,3}$ and all $c_{3,j}$ except for $c_{3,2}$ have to be 0. Thus, the third row in $B$ is $\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \end{pmatrix}$ while in $C$ it is $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$.

Now, for the matrices $B$ and $C$ to actually represent a legal linear straight-line program, for any row $i$ there must be exactly two non-zero entries in the combined $i$-th row of $B$ and $C$. That is, the vector $\begin{pmatrix} b_{i,1} & \ldots & b_{i,n} & c_{i,1} & \ldots & c_{i,k} \end{pmatrix}$ must contain exactly two 1s.

Furthermore, for the represented program to actually compute our linear forms, we have to demand that for each desired output $y_\ell$, there is a line $i$ in the program (and the matrices) such that $v_i = y_\ell$ where $y_\ell = a_{\ell,1} \cdot x_1 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ and $v_i = b_{i,1} \cdot x_1 \oplus \ldots \oplus b_{i,n} \cdot x_n \oplus c_{i,1} \cdot v_1 \oplus \ldots \oplus c_{i,i-1} \cdot v_{i-1}$. Note that for $C$ we only use the lower triangular matrix, as a program may only read intermediate values that have already been written. To represent the mapping of intermediate variables to outputs, we use a function $f : \{1, \ldots, m\} \mapsto \{1, \ldots, k\}$.

**Example 7.2.** Consider again the linear forms from Example 7.1. They are represented by the following matrix $A$.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Likewise, the program is represented by the matrices $B$ and $C$ and the function $f$.

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \quad f = \begin{cases} 1 & \mapsto & 4 \\ 2 & \mapsto & 5 \\ 3 & \mapsto & 3 \\ 4 & \mapsto & 6 \\ 5 & \mapsto & 1 \end{cases}$$

Obviously, all combined rows of $B$ and $C$ contain exactly two non-zero elements. Furthermore, by computing the $v_i$ and the $y_\ell$, we can see that each of the linear forms described by $A$ is computed by the program represented by $B$ and $C$.

As a side note, we remark that the specification need not be given by explicit linear forms, but it can also be given directly via an existing program together with the information which of its program variables contain the desired linear forms.

In [Sch11], Peter Schneider-Kamp reports on a discussion following a guest talk he gave at the University of Volgograd, Russia. One result of this discussion was the observation that the following decision problem for *program optimality* is in Co-NP:

> Given a program $P$ of length $k$ with the intermediate variables $v_1, \ldots, v_k$ and with the $n$ variables $x_1, \ldots, x_n$ over GF(2) as inputs and given also a set $F \subseteq \{v_1, \ldots, v_k\}$, decide if there do *not* exist a program $P'$ of length $k' < k$ with the variables $v'_1, \ldots, v'_{k'}$ and a set $F'$ with $|F'| = |F|$ such that the variables $v'_j \in F'$ contain the linear forms given by the variables $v_i \in F$ in $P$.

So in this optimality problem, we are asking the question: Is a given program $P$ computing certain linear forms (given by the intermediate program variables in $F$) an *optimal* program for these linear forms (w.r.t. program length)?

The proof idea is straightforward: The program $P'$ and the set $F'$ themselves are the certificate for a "NO"-answer of this decision problem on an instance given by $P$ and $F$. The reason is that one can indeed check in polynomial time that $k' < k$ and that the linear forms computed by $P$ in $F$ are equivalent to those computed by $P'$ in $F'$ (i.e., program *equivalence* is in the complexity class P for the class of programs considered in this chapter). Thus, the problem is in Co-NP.

So one could also rephrase our decision problem

> Given $n$ variables $x_1, \ldots, x_n$ over GF(2), $m$ linear forms $y_\ell = a_{\ell,1} \cdot x_1 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ and a natural number $k$, decide if there exists a linear program of length $k$ that computes all $y_\ell$.

as follows:

> Given a linear program $P$ of length $k+1$ with the intermediate variables $v_1, \ldots, v_{k+1}$ over the $n$ variables $x_1, \ldots, x_n$ over GF(2) as inputs and a set $F \subseteq \{v_1, \ldots, v_k\}$, decide if there exists a linear program $P'$ of length $k$ with the variables $v'_1, \ldots, v'_k$ and a set $F'$ with $|F'| = |F|$ such that the variables $v'_j \in F'$ contain the linear forms given by the variables $v_i \in F$ in $P$.

For the remainder of this chapter, however, we consider the specification to be given via a set of linear forms $y_\ell = a_{\ell,1} \cdot x_1 \oplus \ldots \oplus a_{\ell,n} \cdot x_n$ and a natural number $k$. Nonetheless, it is interesting to note that one could also take a given program as an input and synthesize an equivalent optimal program based on the input program.

## 7.2 Encoding to Propositional Logic

Now that the scenario has been set up and the matrix formulation has been introduced, we start by giving a high-level encoding of the decision problem as a logical formula in second order logic. Then we perform a stepwise refinement of that encoding where in each step we eliminate some elements that cannot directly be expressed by satisfiability of propositional logic.

For our first encoding, the carrier is the set of natural numbers, and we use predicates over indices to represent the matrices $A$, $B$, and $C$ as well as the vectors $x$, $y$, and $v$. We also use a function $f$ to map indices of outputs from $y$ to indices of intermediate variables from $v$. Finally, we make use of cardinality constraints by predicates $\text{exactly}_k$ that take a list of variables and check that the number of variables that are assigned 1 is exactly $k$.

First, we need to ensure that $B$ and $C$ represent a legal linear straight-line program. This is encoded by the following formula $\alpha_1$:

$$\alpha_1 = \bigwedge_{1 \le i \le k} \text{exactly}_2(B(i,1), \ldots, B(i,n), C(i,1), \ldots, C(i,i-1))$$

Second, we demand that the values for the intermediate variables from $v$ are computed by using the values from $B$ and $C$:

$$\alpha_2 = \bigwedge_{1 \le i \le k} \left( v(i) \;\leftrightarrow\; \bigoplus_{1 \le j \le n} B(i,j) \wedge x(j) \;\oplus\; \bigoplus_{1 \le p < i} C(i,p) \wedge v(p) \right)$$

Third, we ensure that the value of the intermediate variable determined by $f$ for the $\ell$-th output actually takes the same value as the $\ell$-th linear form:

$$\alpha_3(\ell) = v(f(\ell)) \leftrightarrow \bigoplus_{1 \le j \le n} A(\ell, j) \wedge x(j)$$

Here, $v(f(\ell))$ denotes the intermediate variable which stores the result of the linear form

$y(\ell)$. In other words, the (existentially quantified) function $f$ maps the index $\ell$ of the linear form $y_\ell$ to the index $i = f(\ell)$ of the variable $v_i$ in the vector $v$ that contains the result of $y_\ell$.

Now we can give our first encoding by the following formula $\alpha$:

$$\alpha = \exists B.\exists C.\exists f.\forall x.\exists v.\ \alpha_1 \wedge \alpha_2 \wedge \bigwedge_{1\le\ell\le m} \alpha_3(\ell)$$

Note that we indeed use the expressivity of second order logic as all our quantifications are over predicates and functions. Fortunately, all these only need to be defined on finite domains. In order not to have to deal with quantification over predicates representing matrices and vectors, we can just introduce a finite number of Boolean variables to represent the elements of the matrices and vectors and work on these directly. For example, for the $k \times n$ matrix $B$ we introduce the $k \cdot n$ Boolean variables $b_{1,1} \ldots, b_{k,n}$.

Similarly, for the function $f$ we introduce $m \cdot k$ Boolean variables $f_{\ell,i}$ that denote that the $\ell$-th linear form is computed by the $i$-th intermediate variable. This encoding is a variation of the encoding for permutations introduced by [STA+07, CGST12] for encoding the search for a *recursive path order* in termination analysis.[75] To make sure that these variables actually represent a function, we need to encode well-definedness: for each $\ell$ there must be exactly one $i$ with $f_{\ell,i}$.

We obtain the refined overall constraint $\beta$, which is a formula from QBF:

$$\beta_1 = \bigwedge_{1\le i\le k} \mathrm{exactly}_2(b_{i,1}, \ldots, b_{i,n}, c_{i,1}, \ldots, c_{i,i-1})$$

$$\beta_2 = \bigwedge_{1\le i\le k} \left( v_i \leftrightarrow \bigoplus_{1\le j\le n} b_{i,j} \wedge x_j \oplus \bigoplus_{1\le p<i} c_{i,p} \wedge v_p \right)$$

$$\beta_3(\ell) = \bigwedge_{1\le i\le k} \left( f_{\ell,i} \rightarrow \left( v_i \leftrightarrow \bigoplus_{1\le j\le n} a_{\ell,j} \wedge x_j \right) \right) \ \wedge\ \mathrm{exactly}_1(f_{\ell,1}, \ldots, f_{\ell,k})$$

$$\begin{aligned} \beta \ = \ & \exists b_{1,1}. \ldots \exists b_{k,n}.\exists c_{1,1}. \ldots \exists c_{k,k}.\exists f_{1,1}. \ldots \exists f_{m,k}.\forall x_1. \ldots \forall x_n.\exists v_1. \ldots \exists v_k. \\ & \beta_1 \ \wedge\ \beta_2 \ \wedge \textstyle\bigwedge_{1\le\ell\le m} \beta_3(\ell) \end{aligned}$$

The above formula $\beta$ is in prenex normal form and has a quantifier prefix of the shape "$\exists^+\forall^+\exists^+$". This precludes us from using a SAT solver on $\beta$ directly. For this, we would need to have a quantifier prefix of the shape "$\exists^+$" alone. Thus, unless we want to use a QBF solver, we need to get rid of the "$\forall^+\exists^+$" suffix of the quantifier prefix of $\beta$. In other words, we need to get rid of the quantifications over $x_1, \ldots, x_n$ and $v_1, \ldots, v_k$.

We observe that $\beta$ explicitly contains the computed values $v_i$ of the intermediate vari-

[75] This indicates that SAT encoding *patterns* transcend the concrete application domain of the problem that is encoded. Here expertise obtained in the setting of termination analysis for term rewriting is used for program synthesis.

ables. We can eliminate them by unrolling the defining equations of an intermediate variable $v_i$ to be expressed directly via $x_1, \ldots, x_n$. In other words, we do not regard the intermediate variables for "computing" the result of the linear forms $y_\ell$, but we directly use a closed expression that depends on the $b_{i,j}$ and the $c_{i,p}$. Here, we introduce the auxiliary formulas $\varphi(i)$ for $1 \leq i \leq k$ whose truth value should correspond to the value taken by the corresponding $v_i$:

$$\varphi(i) = (\bigoplus_{1 \leq j \leq n} b_{i,j} \wedge x_j) \oplus (\bigoplus_{1 \leq p < i} c_{i,p} \wedge \varphi(p))$$

We now reformulate $\beta$ to obtain a refined encoding $\gamma$. Note that we do not need to redefine $\beta_1$ and we do not need an equivalent of $\beta_2$ as we unroll the definition of the $v_i$ into $\gamma_3$ using $\varphi(i)$.

$$\gamma_3(\ell) = \bigwedge_{1 \leq i \leq k} \left( f_{\ell,i} \rightarrow \left( \varphi(i) \leftrightarrow \bigoplus_{1 \leq j \leq n} a_{\ell,j} \wedge x_j \right) \right) \wedge \text{ exactly}_1(f_{\ell,1}, \ldots, f_{\ell,k})$$

$$\gamma \;=\; \exists b_{1,1} \ldots \exists b_{k,n} . \exists c_{1,1} \ldots \exists c_{k,k} . \exists f_{1,1} \ldots \exists f_{m,k} . \forall x_1 \ldots \forall x_n . \beta_1 \wedge \textstyle\bigwedge_{1 \leq \ell \leq m} \gamma_3(\ell)$$

Note that it looks as though for each $i$ we had obtained many redundant copies of the subformulas $\varphi(i)$, which would entail a blow-up in formula size. However, in practical implementations it is beneficial to represent propositional formulas not as trees, but as directed acyclic graphs with sharing of common subformulas. This technique is also known as *structural hashing* [ES06]. As discussed in Chapter 5, we perform standard Boolean simplifications (e.g., $\varphi \wedge 1 = \varphi$), we share applications of Boolean connectives modulo commutativity and idempotence (where applicable), and we use varyadic "$\wedge$" and "$\vee$". In contrast, the connectives "$\leftrightarrow$" and "$\oplus$" are binary and associate to the left.

Nevertheless, we still have universal quantification over the inputs as part of our encoding. This states that regardless of the input values for $x_1, \ldots, x_n$, our program should yield the correct result. Fortunately, we can now benefit from linearity of the operation "$\oplus$" on GF(2), which means that the absolute positiveness criterion for polynomials [HJ98] (cf. Theorem 2.18), which is commonly used in automated termination provers using polynomial interpretations ([Lan79, CMTU05], cf. Section 2.3), is not only sound, but also complete (cf. Corollary 2.19). Essentially, the idea is that two linear forms compute the same function iff their coefficients are identical. In this way, we can now drop the inputs $x_1, \ldots, x_n$.

For $1 \leq j \leq n$ and $1 \leq i \leq k$, we introduce the auxiliary formulas $\psi(j, i)$, which should denote the *dependence* of the value for $v_i$ with respect to $x_j$ (i.e., whether the value of $v_i$

toggles if $x_j$ changes or not):

$$\psi(j,i) = b_{i,j} \oplus \bigoplus_{1 \le p < i} c_{i,p} \wedge \psi(j,p)$$

We finally get an encoding $\delta$ in prenex normal form that can be used as input for a SAT solver (by dropping explicit existential quantification, encoding cardinality constraints using [CLS08, ANOR09], and performing Tseitin's transformation [Tse68]).

$$\delta_3(\ell) = \bigwedge_{1 \le i \le k} \left( f_{\ell,i} \rightarrow \bigwedge_{1 \le j \le n} (\psi(j,i) \leftrightarrow a_{\ell,j}) \right) \wedge \text{ exactly}_1(f_{\ell,1}, \ldots, f_{\ell,k})$$

$$\delta = \exists b_{1,1} \ldots \exists b_{k,n} . \exists c_{1,1} \ldots \ldots \exists c_{k,k} . \exists f_{1,1} \ldots \exists f_{m,k} .\ \beta_1 \wedge \bigwedge_{1 \le \ell \le m} \delta_3(\ell)$$

For the implementation of $\delta$ we used the same setting as in the previous chapters of this thesis, i.e., we employed the SAT framework in the verification environment AProVE [GST06] and the Tseitin implementation from SAT4J [LP10].

## Size of the Encoding

Given a decision problem with an $m \times n$ matrix and a natural number $k$ (where w.l.o.g. $m \le k$ holds since for $m > k$, we could just set $\delta = 0$), our encoding $\delta$ has size $\mathcal{O}(n \cdot k^2)$ if the cardinality constraints are encoded in space linear in the number of arguments [CLS08]. To see this, consider the following size estimation for $\delta$ where due to the use of structural hashing we can look at $\delta_3$ and $\psi$ separately.

$$|\delta| = \mathcal{O}(k \cdot n + k \cdot k + m \cdot k + |\beta_1| + m \cdot |\delta_3| + n \cdot k \cdot |\psi|)$$

For $\beta_1$ and $\delta_3$ we obtain the following estimations where $g$ is a function describing the size of the cardinality constraint:

$$|\beta_1| = \mathcal{O}(k \cdot g(n+k)) \qquad |\delta_3| = \mathcal{O}(k \cdot n + g(k))$$

For $\psi$ we immediately obtain the size estimation $|\psi| = \mathcal{O}(k)$. Now, we can simplify the estimation for $\delta$ by using $m \le k$:

$$\begin{aligned} |\delta| &= \mathcal{O}(k \cdot n + k \cdot k + m \cdot k + k \cdot g(n+k) + m \cdot (k \cdot n + g(k)) + n \cdot k \cdot k) \\ &= \mathcal{O}(n \cdot k^2 + k \cdot g(n+k) + m \cdot g(k)) \end{aligned}$$

## Tuning the Encoding

The models of the encoding $\delta$ from this section are all linear straight-line programs of length $k$ that compute the $m$ linear forms $y_1, \ldots, y_m$. The programs can be decoded from a satisfying assignment of the propositional formula by simply reconstructing the matrices $B$ and $C$.

In this chapter, we are interested in finding short programs. Thus, we can exclude many programs that perform redundant computations. We do so by adding further conjuncts that exclude those undesired programs. While we change the set of models, note that we do not change the satisfiability of the decision problem. That is, if there is a program that computes the given linear forms in $k$ steps, we will find one which does not perform these kinds of redundant computation.

The first kind of redundant programs are programs that compute the same linear form twice, i.e., there are two different intermediate variables that contain the same linear form. We exclude such programs by demanding that for all distinct pairs of intermediate variables $v_i$ and $v_p$, there is also some $x_j$ that influences exactly one of the two variables:

$$\bigwedge_{1 \leq i \leq k} \bigwedge_{1 \leq p < i} \bigvee_{1 \leq j \leq n} (\psi(j,p) \oplus \psi(j,i))$$

The second kind of redundant programs are programs that compute the constant 0 or a linear form just depending on the value of one input variable. To exclude such programs, we add cardinality constraints stating that each compute linear form must depend on at least two input variables.

$$\bigwedge_{1 \leq i \leq k} \text{atLeast}_2(\psi(1,i), \ldots, \psi(n,i))$$

In fact, statements that compute linear forms that only depend on two input variables can be restricted not to use any other intermediate variables (as they could be computed in one step from the inputs).

$$\bigvee_{1 \leq j < i} c_{i,j} \rightarrow \bigwedge_{1 \leq i \leq k} \text{atLeast}_3(\psi(1,i), \ldots, \psi(n,i))$$

Apart from disallowing redundant programs, we additionally include implied conjuncts to further constrain the search space. In this way, the SAT solver becomes more efficient as unit propagation can be employed in more situations.

As stated in Section 7.1, we require that the input does not contain duplicate linear forms. Consequently, we may require $f$ to be injective, i.e., any intermediate variable covers at most one linear form.

$$\bigwedge_{1 \leq i \leq k} \text{atMost}_1(f_{1,i}, \ldots, f_{m,i})$$

Often, CDCL-based SAT solvers are not very good at solving the pigeonhole problem. Additional constraints facilitate better unit propagation in these cases. Since $f$ maps from $\{1, \ldots, m\}$ to $\{1, \ldots, k\}$, only at most $k$ of the $f_{\ell,i}$ may become true.

$$\text{atMost}_k(f_{1,1}, \ldots, f_{m,k})$$

Similarly, we can even state that at least $m$ of the $f_{\ell,i}$ need to become true as we have to compute all given (distinct) $m$ linear forms.

$$\text{atLeast}_m(f_{1,1}, \ldots, f_{m,k})$$

## 7.3 From Decision Problem to Optimization

A simple approach for solving an optimization problem given a decision procedure for the associated decision problem is to search for the parameter to be optimized by repeatedly calling the decision procedure.

In our case, for minimizing the length $k$ of the synthesized linear straight-line program, we start by observing that this minimal length must be at least the number of linear forms. At the same time, if we compute each linear form separately, we obtain an upper bound for the minimal length. More precisely, we know that the minimal length $k_{min}$ is in the closed interval from $m$ to $|A|_1 - m$ where $|\cdot|_1$ denotes the number of 1s in a matrix.

Without further heuristic knowledge about the typical length of shortest programs, the obvious thing to do is to use a bisecting approach for refining the interval. That is, one selects the middle element of the current interval and calls a decision procedure based on our encoding from Section 7.2 for this parameter. If there is a model, the interval is restricted to the lower half of the previous interval and we continue bisecting. If there is no model and $\delta$ is unsatisfiable, the interval is restricted to the upper half of the previous interval. When the interval becomes empty, the lower limit indicates the minimal parameter $k_{min}$.

The above approach requires a logarithmic number of calls to the decision procedure, approximately half of which will return the result “unsatisfiable”. This approach is very efficient if we can assume that our decision procedure takes approximately the same time for a positive answer as for a negative answer. As we will see in the case study of the following section, though, for realistic problem instances the negative answers may require orders of magnitude more time.

Thus, to minimize the number of calls to the decision procedure resulting in a negative

answer, we propose the following algorithm for refining the length $k$.

(i) Start with $k := |A|_1 - m - 1$.

(ii) Call the decision procedure with $k$.

(iii) If `UNSAT`, return $k + 1$ and exit.

(iv) If `SAT`, compute used length $k_{used}$ from $B$ and $C$.

(v) Set $k := k_{used} - 1$ and go to Step ii.

Here, the *used length* of a program is the number of variables that are needed directly or indirectly to compute the $m$ linear forms. For given matrices $B$ and $C$ and a function $f$, the set of used variables *used* is the least set such that:

- if $f(\ell) = i$, then $v_i \in \mathit{used}$ and
- if $v_i \in \mathit{used}$ and $c_{i,j} = 1$, then $v_j \in \mathit{used}$.

The used length can then be obtained as the cardinality of the set *used*.

This algorithm obviously only results in exactly one call to UNSAT—directly before finding the minimal solution. The price we pay for this is that in the worst case we have to call the decision procedure a linear number of times. In practice, though, for $k > k_{min}$, there are many solutions and the solution returned by the SAT solver typically has $k_{used} < k$. Consequently, at the beginning the algorithm typically approaches $k_{min}$ in rather large steps.

While it seems natural to use MaxSAT for our optimization problem instead of calling the SAT solver repeatedly, the decision problems close to the optimum are already so hard that solving these as part of a larger instance seems infeasible.

## 7.4 Case Study: Advanced Encryption Standard

As mentioned in the introduction, a major motivation for our work is the minimization of circuits for implementing cryptographic algorithms. In this section, we study how our contributions can be applied to optimize an important component of the *Advanced Encryption Standard (AES)* [Fed01].

The AES algorithm consists of the (repeated) application of four steps. The main step for introducing non-linearity is the SubBytes step that is based on a so-called S-box. This S-box is a transformation based on multiplicative inverses in $\mathrm{GF}(2^8)$ combined with an invertible affine transformation. This step can be decomposed into two linear parts and a minimal non-linear part.

For our case study, we consider the first of the linear parts (called the "top matrix" in [BP10]) which is represented by the following $21 \times 8$ matrix $A$:

$$A = \begin{pmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0
\end{pmatrix}$$

The following matrices $B$ and $C$ represent a solution with length $k = 23$. This solution was found in less than one minute using our decision procedure from Section 7.2 with MiniSAT v2.1 as back-end on a 2.67 GHz Intel Core i7.

$$B = \begin{pmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

$$C = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0
\end{pmatrix}$$

This is also the length of the previously shortest known linear straight-line program for the linear forms described by the matrix $A$, which has length $k = 23$ as well [BP10]. This shows that our SAT-based optimization method is able to find very good solutions in reasonable time. The UNSAT case is harder, though. For $k = 20$ (which is trivially unsatisfiable due to the pigeonhole problem), without the tunings from Section 7.2 we cannot show unsatisfiability within 4 days. But with the tunings enabled we can show unsatisfiability in less than one second.

Unfortunately, showing the conjecture $k_{min} = 23$ via the unsatisfiability for $k = 22$ proves to be much more challenging. Indeed, we have run many different SAT solvers (including but not limited to glucose, ManySat, MiniSAT, MiraXT with 8 threads, OKsolver, PicoSAT, PrecoSAT, RSat, SAT4J) on the CNF file for this instance of our decision problem. Some of the more promising solvers for this instance were run for more than 40 days without returning either SAT or UNSAT.

In this effort to prove unsatisfiability of this hard SAT instance, we have also asked for and received a lot of support and good advice from the SAT community (see the Acknowledgments at the end of this chapter). Still, the unsatisfiability of this instance remained a conjecture for several months, even though pre-processing techniques can reduce the number of variables of this instance from more than 45000 to less than 5000 in a matter of minutes.[76]

Thanks to a pointer at the SAT '10 conference, we invoked version 2.5.1 of the SAT solver cryptominisat [SNC09] on this challenging instance. Using cryptominisat, we finally

[76]The reader is cordially invited to try his favorite SAT solver on one of the instances available from: `http://aprove.informatik.rwth-aachen.de/eval/slp.zip`

succeeded in showing unsatisfiability of this instance within less than 106 hours on an Opteron 848 at 2.2 GHz. It is interesting to note that this SAT solver has been designed specifically to extract encoded XOR operations and to treat them specially during the solving process. The motivation for doing so is that XOR operations are often used within cryptographic ciphers.

In [HJN11], Hyvärinen, Junttila and Niemelä report on the fastest wall-time proof known for this instance to date, found using a grid-based parallel approach in approx. 45 hours.[77]

To analyze how difficult these problems really are, we consider a small subset of the linear forms to be computed for the top matrix. The table to the right shows how the runtimes in seconds of the SAT solver are affected by the choice of $k$ for the case that we consider only the first 8 out of 21 linear forms from $A$. In order to keep runtimes manageable we already incorporated the symmetry breaking improvement described in Section 7.5. Note that unsatisfiability for $k = 12$ is still much harder to show than satisfiability for $k_{min} = 13$.

| $k$ | result | time |
|---|---|---|
| 8 | UNSAT | 0.4 |
| 9 | UNSAT | 0.5 |
| 10 | UNSAT | 1.2 |
| 11 | UNSAT | 5.0 |
| 12 | UNSAT | 76.8 |
| 13 | SAT | 1.0 |
| 14 | SAT | 3.4 |
| 15 | SAT | 2.8 |
| 16 | SAT | 1.5 |
| 17 | SAT | 4.3 |
| 18 | SAT | 2.7 |
| 19 | SAT | 2.5 |
| 20 | SAT | 3.0 |
| 21 | SAT | 3.0 |
| 22 | SAT | 3.5 |
| 23 | SAT | 3.6 |
| 24 | SAT | 5.5 |
| 25 | SAT | 5.9 |

To conclude this case study, we see that while finding (potentially) minimal solutions is obviously feasible, proving their optimality (i.e., unsatisfiability of the associated decision problem for $k = k_{min} - 1$) is challenging. This observation confirms observations made in [KKY09]. In the following section we present some of our attempts to improve the efficiency of our encoding for the UNSAT case.

## 7.5 Handling the UNSAT Case

Satisfiability of propositional logic is an NP-complete problem and, thus, we can expect that at least some instances are computationally expensive. While SAT solvers have proven to be a Swiss army knife for solving practically relevant instances of many different NP-complete problems, our kind of program synthesis problems seems to be a major challenge for today's SAT solvers even on instances with "just" 1500 variables.

In this section we discuss three different approaches based on unary SAT encodings, on

[77] Since this case study provides for SAT instances which are at the same time relatively small, surprisingly hard to solve, and at the same time stem from a practically highly relevant application, we submitted several instances for different program lengths for the AES top matrix as benchmarks to the SAT competition 2011, cf. `http://www.satcompetition.org/`. Moreover, [BP10] presents a counterpart to the top matrix, viz. the "bottom matrix", which gives the linear forms of the second linear part used by [BP10] for the implementation of the AES S-box. This matrix contains 8 linear forms over 18 variables, and the resulting SAT instances turn out to be even harder for modern SAT solvers than those from the top matrix.

Pseudo-Boolean satisfiability, and on symmetry breaking.

## Unary encodings

As remarked by [GLP06], encoding arithmetic in unary representation instead of the more common binary (CPU-like) representation can be very beneficial for the performance of modern CDCL-based [MLM09] SAT solvers on the resulting instances. This observation is confirmed by our work presented in Chapter 5. One possible approach could be to encode the computations not via XOR on GF(2), but rather in unary representation on $\mathbb{N}$ with a deferred parity check (i.e., first compute the value of the variables in the program using addition on $\mathbb{N}$ instead of GF(2), and perform the check whether the number is even or odd only when needed). Unfortunately, this approach turned out to be prohibitively expensive as the (integer) values for the $i$-th line are bounded only by $\mathcal{O}(\mathit{fib}(i))$ where $\mathit{fib}$ is the Fibonacci function.

## Encoding to Pseudo-Boolean Constraints

In addition to optimizing and tuning our encoding to SAT, we also implemented a straightforward encoding to Pseudo-Boolean constraints. The hope was that, e.g., cutting plane approaches could be useful for showing unsatisfiability.

We experimented with MiniSAT+, Pueblo, SAT4J, and SCIP but were not able to obtain any improvements for, e.g., the first 8 linear forms of the top matrix.

## Symmetry Breaking

In general, having many solutions is considered good for SAT instances as the SAT solver is more likely to "stumble" upon one of them. For UNSAT instances, though, having many potential solutions usually means that the search space to exhaust is very large.

One of the main reasons for having many solutions is symmetry. For example, it does not matter if we first compute $v_1 = x_1 \oplus x_2$ and then $v_2 = x_3 \oplus x_4$ or the other way around. Limiting these kinds of symmetries can be expected to significantly reduce the runtimes for UNSAT instances.

In our concrete setting, being able to reorder independent program lines is one of the major sources of symmetry. Two outputs in a straight-line programs are said to be *independent* if neither of them depends on the other (directly through the matrix $C$ or indirectly).

Now, the idea for breaking symmetry is to impose an order on these lines: the line which computes the "smaller" linear form (w.r.t. a total order on linear forms, which can, e.g., be obtained by lexicographic comparison of the coefficient vectors) must occur before the line which computes the greater linear form.

We can encode the direct dependence of $v_i$ on $v_p$:

$$\bigwedge_{1\leq i\leq k}\bigwedge_{1\leq p<i} c(i,p) \rightarrow dep(i,p)$$

Likewise, the indirect dependence of $v_i$ on $v_p$ can be encoded by transitivity:

$$\bigwedge_{1\leq i\leq k}\bigwedge_{1\leq p<i}\bigwedge_{p<q<i} c(i,q) \wedge dep(q,p) \rightarrow dep(i,p)$$

We also need to encode the reverse direction, i.e.:

$$\bigwedge_{1\leq i\leq k}\bigwedge_{1\leq p<i}\left(dep(i,p) \rightarrow \left(c(i,p) \ \vee \bigvee_{p<q<i} (c(i,q) \wedge dep(q,p))\right)\right)$$

Now we can enforce that for $i > p$, the output $v_i$ depends on the output $v_p$ or $v_i$ encodes a greater linear form than $v_p$:

$$\bigwedge_{1\leq i\leq k}\bigwedge_{1\leq p<i} (dep(i,p) \ \vee \ [\psi(1,i),\ldots,\psi(n,i)] >_{lex} [\psi(1,p),\ldots,\psi(n,p)])$$

Here lexicographic comparison of formula tuples is encoded in the usual way (see for example the encodings given in the papers [FGM+07, CLS08], which deal with SAT encodings for automated termination analysis).

While this approach eliminates some otherwise valid solutions of length $k$ and thus reduces the set of admissible solutions, obviously there is at least one solution of length $k$ which satisfies our constraints whenever solutions of length $k$ exist at all. This way, we greatly reduce the search space by breaking symmetries that are not relevant for the result, but may slow down the search considerably.

Consider again the restriction of our S-box top matrix to the first 8 linear forms. With symmetry breaking, we can show unsatisfiability for the "hard" case $k = 12$ in 76.8 seconds. In contrast, without symmetry breaking, we cannot show unsatisfiability within 4 days.

## 7.6 Summary and Outlook

In this chapter we have shown how shortest linear straight-line programs for given linear forms (or equivalently, XOR-circuits with a minimal number of gates) can be synthesized using SAT solvers. We have given a formalization of this problem in Section 7.1. To this end, in Section 7.2 we have presented a novel polynomial-size encoding of the associated decision problem to SAT and a customized white-box method for again turning this decision procedure into an optimization algorithm. Our approach is based on methodology

from constraint-based automated termination analysis such as the parametric approach from [CMTU05] to modeling the problem and on SAT encoding patterns for subproblems that arise in diverse applications.

In Section 7.4, we have evaluated the feasibility of this approach by a case study where we minimize an important part of the S-box for the Advanced Encryption Standard. This study shows that our SAT-based approach is indeed able to synthesize shortest-known programs for realistic problem settings within reasonable time.

Proving the optimality of the programs found by showing unsatisfiability of the associated decision problem leads to very challenging SAT problems. To improve the performance for the UNSAT case, in Section 7.5 we have discussed three approaches based on unary encodings, on a port to Pseudo-Boolean satisfiability, and on symmetry breaking. We have shown that symmetry breaking significantly reduces runtimes in the UNSAT case. This way, we have managed to obtained an optimality proof for our case study via the SAT solver cryptominisat, which has in the mean time been confirmed independently by [HJN11].

As direct future work, one could of course apply our method to other problems from cryptography. Apart from that, also further enhancements and variations of the encoding seem possible. In 2010, Michael Codish and Olaf Owe independently suggested an encoding where the lines of the parametric program do not reflect the order of the instructions, but rather the order of the desired linear forms. This way, one would not need the explicit mapping from linear forms to variables holding their values anymore. Also a different representation of this mapping, e.g., using order encoding [CB94, TTKB09] would be worth investigating. Moreover, Harald Zankl suggested encoding the problem to SMT-LIA. Indeed, via our description of the problem on a high level of abstraction in Section 7.2, one can easily port the concrete formulation of the problem from propositional logic to other formalisms. For instance, in [Ban10], Mutsunori Banbara reports on his implementation of the encoding in the input language of the CSP solver Sugar [TTKB09] based on our description of the encoding in [FS10].

One could also extend the encoding to optimize with respect to further properties. For instance, for digital circuits, not only their size, but also their depth is an important measure of optimality. The reason is that in physical systems, circuit depth directly affects the total runtime of a signal from the circuit input to its output. This in turn determines the possible clock rate for the circuit as part of a greater system. Therefore, it is not only desirable to obtain circuits with few gates, but the resulting circuits should at the same time also have a small depth. Thus, it is not surprising that (using incomplete methods) [BP11] addresses the challenge of searching for XOR circuits with a minimal gate count for a given maximal depth. Lifting the complete SAT-based approach of this chapter to this refined setting would thus provide an interesting extension of the present work.

Finally, another direction of work could be to specialize existing SAT solvers to further improve performance in the UNSAT case.

## Acknowledgments

Regarding this chapter, our sincere thanks go to Erika Ábrahám, Daniel Le Berre, Armin Biere, Youssef Hamadi, Oliver Kullmann, Matthew Lewis, Lakhdar Saïs, and Laurent Simon for input on and help with the experiments. Furthermore, we thank Joan Boyar and René Peralta for providing us with information on their work and Michael Codish for pointing out similarities to common subexpression elimination.

# 8 Conclusion

This thesis presents several contributions to the state of the art of automated termination analysis and circuit synthesis by means of SAT and SMT encodings. In Chapter 3, we present a novel automation for polynomial interpretations with negative constants [HM07] based on an encoding to SMT-NIA. This then allows us to select an arbitrary SMT solver for quantifier-free NIA formulas, e.g., the one arising from our SAT encoding for SMT-NIA in [FGM+07, Fuh07]. Moreover, we prove that given a set of term constraints, it suffices to consider negative constants for only a subset of the function symbols. Using these contributions, we obtain speedups by orders of magnitude for synthesis of this useful class of orders.

In Chapter 4, we present max-polynomial interpretations, which allow to combine polynomial interpretations by the max- and min-operations. This way, we obtain a very flexible class of weakly monotone algebras. Moreover, we provide transformation rules that lead to an SMT-NIA encoding for the search for suitable max-polynomial interpretations. We point out a class of terms where an exponential blowup in constraint size occurs for a structurally simple max-polynomial interpretation. Since additionally, it is not clear *a priori* which shape one should use for a parametric interpretation, for successful automation we provide syntactic heuristics indicating for each function symbol which shape of max-polynomial interpretations should be used. Moreover, we provide novel transformation rules to alleviate the exponential blowup for certain cases. As our experiments indicate, the integration of max-polynomial interpretations using the contributions of Chapter 4 increases the power of the termination tools AProVE and $\mathsf{T_TT_2}$ notably. Finally, we also provide experimental evidence that the approach presented in Chapter 3 is significantly more suitable for polynomial interpretations with a negative constant than an approach based on the setting of Chapter 4.

In Chapter 5, we give a novel SAT encoding for arctic interpretations, which are particularly useful, e.g., for string rewrite systems. This SAT encoding is based on a unary representation of numbers via order encoding, which is beneficial for propagation in CDCL-based SAT solvers. Our experiments with the implementation in AProVE indicate notable improvements in practice over the encoding from [KW08, KW09], which is based on a binary representation of numbers. These improvements lead to a notable increase in power for AProVE, which, according to our experiments, was most probably crucial in winning the category *SRS Standard* of the Termination Competition in the years 2009 – 2011.

Chapter 6 presents an adaption of SCNP [BC08], a sufficient and in practice "almost complete" criterion for *size-change termination* [LJB01] to term rewriting. This adaption via SCNP reduction pairs works very smoothly and requires only a very natural extension of the DP framework [GTSF06] to support *tuple typing* for our correctness proof. This extension is likely to be useful also for adaption of further termination techniques from other formal settings. For automation, we contribute an extension of the SAT encoding of [BC08] to allow for the automatic search for both the underlying base reduction pair used to abstract to size-change graphs and for the size-change termination argument for these size-change graphs. This way, for the first time we apply size-change termination analysis in a setting where the abstraction is not fixed beforehand, but instead we search for a suitable abstraction only when needed. We also provide and solve a challenge example which highlights that SCNP can add, e.g., max-comparison on top of a class of base reduction pairs that does not support such comparisons. Our experiments reveal significant performance improvements over the previous integration of size-change termination [TG05].

In Chapter 7 we show that the methods applied for termination proving in this thesis also carry over to the—at first glance rather distinct—application domain of circuit synthesis. Given a specification, we automatically implement a Boolean function between tuples of inputs and outputs which is solely composed of XOR operations (or equivalently, a linear straight-line program over GF(2)). This class of circuits/programs frequently occurs, e.g., as part of symmetric ciphers in cryptography. The goal is to minimize the number of XOR gates used by the implementation. To attain these goals, we adapt the parametric approach [CMTU05] from termination analysis used throughout this thesis. To represent the needed constraints on the parameters, we propose a suitable SAT encoding, which includes several optimizations to make the approach scale to practical instances for the current generation of SAT solvers. To achieve this SAT encoding, we reuse ideas and encoding patterns introduced for termination analysis. We conduct a case study to implement an important part of the *Advanced Encryption Standard* following [BP10], which shows that we can indeed obtain optimal implementations quickly and prove their optimality within reasonable time also for highly practical examples.

## Future Work

While this thesis already provides several advances for the state of the art of termination proving and circuit synthesis, of course there are still further possibilities for future research. In addition to the possible directions for future work that we provide at the end of the individual chapters, also improvements seem possible concerning the bigger overall picture of SAT and SMT encodings and termination analysis.

One topic that still seems under-explored is the question which redundant constraints

one should add to a "minimal" problem encoding to facilitate the satisfiability check for CDCL SAT solvers. For instance, in Chapter 7 we observe that adding redundant constraints (and thus increasing the size of the SAT instance) renders otherwise very hard SAT instances from applications like AES feasible. The reason is that adding clauses beforehand which otherwise would have to be learned by the SAT solver in a time-consuming process can drastically reduce the time needed for the overall search. Obviously, the question arises *which* additional redundant constraints can actually be helpful and which constraints merely inflate the SAT instance without aiding in the search process. In general, it takes a lot of expertise on SAT encodings and on the inner workings of the class of SAT solvers used (such as CDCL solvers) to anticipate which additional redundant constraints are going to speed up the search. Hence, automation is desirable in order to make efficient SAT-based constraint solving also available to the non-expert user who only wishes to describe the problem at hand without worrying about details of the search process.

One possible approach to achieve this could be to analyze the clauses that are learned in the solving process for related instances. Here one could, e.g., keep track of occurrences of those variables in learned clauses that arise directly from actual parameters of the underlying higher-level constraint model, such as variables encoding parametric coefficients as in Chapters 3 – 5, or variables encoding a mapping between finite sets as in Chapter 7. Then one could for instance apply data mining algorithms to propose candidates for (preferably small) clauses that are already entailed by the original SAT problem. By correlation between different SAT instances where one marks the role of variables, one could try to detect common patterns in the inferred clauses and lift them from the concrete arising instances in a "minimal" encoding to the abstract setting of the parameterized encoding.

Another point is that devising a SAT encoding entirely by hand as done in this thesis can be a cumbersome and error-prone task because of all the details one has to pay attention to on this low level of abstraction. Instead, it would be preferable to give a specification of the desired constraints on a suitably high level of abstraction. Then one could invoke a dedicated *SAT compiler* that generates an (optimized) SAT encoding for these particular constraints. Steps in this direction have been done already. For instance, in the paper [MJ10], Marić and Janičić present the tool URBiVA, which takes a checker program for a solution candidate as an input and synthesizes a SAT or SMT encoding based on this program. In [TTKB09], Tamura *et al.* present the dedicated SAT encoder Sugar for linear finite constraint satisfaction problems which is based on order encoding (cf. Chapter 5). Moreover, in [MCLS11], Metodi *et al.* present a tool that encodes a constraint problem from a declarative description of the constraints in a generic language to a SAT instance. Here, additionally information from the high-level description is used to improve the quality of the resulting SAT encoding.

Via such translators as part of the tool chain, the role of SAT encodings as a low-level "assembly language" for constraint programming becomes even more pronounced. While decades ago, it was customary for programmers to write hand-optimized assembly programs, nowadays it has become commonplace to write programs in a higher-level programming language. These are not executable as such, but they are processed further by a compiler which applies optimizations and then generates the actual assembly code. Likewise, also for constraint programming it is thus desirable to abstract from the low-level implementation details via *SAT compilers*, This way, the constraint programmer can focus purely on the problem statement, not on means of speeding up the solution process.

Finally, thanks to the modularity of encoding-based techniques, further improvements can of course also be achieved via speedups in SAT and SMT solvers. Apart from improvements to the solvers that are beneficial in general (e.g., more efficient data structures), one can also try to perform optimizations for the class of SAT or SMT instances that arises during termination analysis. For instance, for the SMT-NIA instances arising from the search for polynomial orders, empirical evidence [FGM+07] shows that in practice, it suffices to search for rather small coefficients. Thus, this class of instances would benefit particularly if SMT-NIA solvers were optimized specifically to find satisfiability proofs for small search spaces quickly.

On a broader scope for termination, it is interesting to remark that successful work on termination analysis has been conducted by several scientific communities with distinct background. Concretely, for instance the present thesis is set up with term rewriting as the underlying model of computation, and a lot of work both on the theoretical foundations and on automation has given rise to a number of strong termination tools for rewriting. Likewise, also approaches based on software model checking and transition invariants have been applied in significant research on automated termination analysis in recent years, which has also led to several termination tools operating on imperative input languages, e.g., Terminator [CPR06, CPR09] or Loopfrog [KST+09, TSWK11].

However, so far there has only been little cross-fertilization between these lines of research which pursue the common goal of automated termination analysis, but are based on different backgrounds. Thus, a possible direction for future research is to look for synergies between these distinct settings. For instance, it would be interesting to investigate if there exist proof techniques which are successfully used in one of the settings, but are unknown in the other. By adapting such techniques (in both directions), one may be able to combine the strengths of both approaches and thus obtain a "best of both worlds" setting. Another option could be a modular coupling of termination tools on the level of subproblems. Here some parts of a termination problem would be solved by termination analysis based on rewriting, and other parts would be dealt with by termination analysis based on software model checking.

# Bibliography


[AAC+08] Elvira Albert, Puri Arenas, Michael Codish, Samir Genaim, Germán Puebla, and Damiano Zanardini. Termination analysis of Java Bytecode. In *Proc. 10th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS '08)*, volume 5051 of *LNAI*, pages 2–18. Springer, 2008.

[AAG+07] Elvira Albert, Puri Arenas, Samir Genaim, Germán Puebla, and Damiano Zanardini. Cost analysis of Java Bytecode. In *Proc. 16th European Symposium on Programming (ESOP '07)*, volume 4421 of *LNCS*, pages 157–172. Springer, 2007.

[Abe04] Andreas Abel. Termination checking with types. *RAIRO - Theoretical Informatics and Applications*, 38(4):277–319, 2004.

[ACG+06] Elena Annov, Michael Codish, Jürgen Giesl, Peter Schneider-Kamp, and René Thiemann. A SAT-based implementation for RPO termination. In *Short Papers of LPAR 2006*, 2006.

[AEF+08] Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, and René Thiemann. Improving context-sensitive dependency pairs. In *Proc. 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '08)*, volume 5330 of *LNAI*, pages 636–651. Springer, 2008.

[AEM11a] Martin Avanzini, Naohi Eguchi, and Georg Moser. A path order for rewrite systems that compute exponential time functions. In *Proc. 22nd International Conference on Rewriting Techniques and Applications (RTA '11)*, volume 10 of *LIPIcs*, pages 123–138. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[AEM11b] Martin Avanzini, Naohi Eguchi, and Georg Moser. A path order for rewrite systems that compute exponential time functions (technical report). *Computing Research Repository*, abs/1010.1128, 2011.

[AG00] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.

[AG01] Thomas Arts and Jürgen Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, Germany, September 2001.

[ALN09] Beatriz Alarcón, Salvador Lucas, and Rafael Navarro-Marset. Proving termination with matrix interpretations over the reals. In *Proc. 10th International Workshop on Termination (WST '09)*, pages 12–15, 2009.

[AM08] Martin Avanzini and Georg Moser. Complexity analysis by rewriting. In *Proc. 9th International Symposium on Functional and Logic Programming (FLOPS '08)*, volume 4989 of *LNCS*, pages 130–146. Springer, 2008.

[AM09] Martin Avanzini and Georg Moser. Dependency pairs and polynomial path orders. In *Proc. 20th International Conference on Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *LNCS*, pages 48–62. Springer, 2009.

[Ama05] Roberto M. Amadio. Synthesis of max-plus quasi-interpretations. *Fundamenta Informaticae*, 65(1–2):29–60, 2005.

[ANOR09] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In *Proc. 12th International Conference on Theory and Applications of Satisfiability Testing (SAT '09)*, volume 5584 of *LNCS*, pages 167–180. Springer, 2009.

[Ava10] Martin Avanzini. POP* and semantic labeling using SAT. In *Selected Papers of Interfaces: Explorations in Logic, Language and Computation, ESSLLI 2008 and ESSLLI 2009 Student Sessions (ESSLLI '08/09)*, volume 6211 of *LNAI*, pages 155–166. Springer, 2010.

[Ave06] James Avery. Size-change termination and bound analysis. In *Proc. 8th International Symposium on Functional and Logic Programming (FLOPS '06)*, volume 3945 of *LNCS*, pages 192–207. Springer, 2006.

[AY09] Takahito Aoto and Toshiyuki Yamada. Argument filterings and usable rules for simply typed dependency pairs. In *Proc. 7th International Symposium on Frontiers of Combining Systems (FroCoS '09)*, volume 5749 of *LNAI*, pages 117–132. Springer, 2009.

[Ban10] Mutsunori Banbara. Personal communication, 2010.

[BC08] Amir M. Ben-Amram and Michael Codish. A SAT-based approach to size change termination with global ranking functions. In *Proc. 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08)*, volume 4963 of *LNCS*, pages 218–232. Springer, 2008.

[BCDO06] Josh Berdine, Byron Cook, Dino Distefano, and Peter W. O'Hearn. Automatic termination proofs for programs with shape-shifting heaps. In *Proc. 18th International Conference on Computer Aided Verification (CAV '06)*, volume 4144 of *LNCS*, pages 386–400. Springer, 2006.

[BCG+07] Maurice Bruynooghe, Michael Codish, John P. Gallagher, Samir Genaim, and Wim Vanhoof. Termination analysis of logic programs through combination of type-based norms. *ACM Transactions on Programming Languages and Systems*, 29(2), 2007.

[BK11] Frédéric Blanqui and Adam Koprowski. CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 21(4):827–859, 2011.

[BL87] Ahlem Ben Cherifa and Pierre Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, 1987.

[BL07] Amir M. Ben-Amram and Chin Soon Lee. Program termination analysis in polynomial time. *ACM Transactions on Programming Languages and Systems*, 29(1):1–37, 2007.

[BLN+09] Cristina Borralleras, Salvador Lucas, Rafael Navarro-Marset, Enric Rodríguez-Carbonell, and Albert Rubio. Solving non-linear polynomial arithmetic via SAT modulo linear arithmetic. In *Proc. 22nd International Conference on Automated Deduction (CADE '09)*, volume 5663 of *LNAI*, pages 294–305. Springer, 2009.

[BLO+12] Cristina Borralleras, Salvador Lucas, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. SAT modulo linear arithmetic for solving polynomial constraints. *Journal of Automated Reasoning*, 48(1):107–131, 2012.

[BMP08] Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In *Proc. 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS '08)*, volume 5162 of *LNCS*, pages 168–179. Springer, 2008.

[BMS05] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Termination of polynomial programs. In *Proc. 6th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '05)*, volume 3385 of *LNCS*, pages 113–129. Springer, 2005.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[BOEG10] Marc Brockschmidt, Carsten Otto, Christian von Essen, and Jürgen Giesl. Termination graphs for Java Bytecode. In *Verification, Induction, Termination Analysis*, volume 6463 of *LNAI*, pages 17–37. Springer, 2010.

[BOG11] Marc Brockschmidt, Carsten Otto, and Jürgen Giesl. Modular termination proofs of recursive Java Bytecode programs by term rewriting. In *Proc. 22nd International Conference on Rewriting Techniques and Applications (RTA '11)*, volume 10 of *LIPIcs*, pages 155–170. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[BP09] Joan Boyar and René Peralta. A new technique for combinational circuit optimization and a new circuit for the S-Box for AES. Patent Application Number 61089998 filed with the U.S. Patent and Trademark Office, 2009.

[BP10] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In *Proc. 9th International Symposium on Experimental Algorithms (SEA '10)*, volume 6049 of *LNCS*, pages 178–189. Springer, 2010.

[BP11] Joan Boyar and René Peralta. A depth-16 circuit for the AES S-box. Technical Report 2011/332, Cryptology ePrint Archive, 2011.

[BSOG12] Marc Brockschmidt, Thomas Ströder, Carsten Otto, and Jürgen Giesl. Automated detection of non-termination and `NullPointerException`s for Java Bytecode. In *Proc. 2nd International Conference on Formal Verification of Object-Oriented Software (FoVeOOS '11)*, volume 7421 of *LNCS*, pages 123–141. Springer, 2012.

[CB94] James M. Crawford and Andrew B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proc. 12th National Conference on Artificial Intelligence (AAAI '94)*, volume 2, pages 1092–1097. AAAI Press, 1994.

[CCF+07] Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Certification of automated termination proofs. In *Proc. 6th International Symposium on Frontiers of Combining Systems (FroCoS '07)*, volume 4720 of *LNAI*, pages 148–162. Springer, 2007.

[CCF+11] Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Automated certified proofs with C*i*ME3. In *Proc. 22nd International*

*Conference on Rewriting Techniques and Applications (RTA '11)*, volume 10 of *LIPIcs*, pages 21–30. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[CD96] Philippe Codognet and Daniel Diaz. Compiling constraints in clp(FD). *Journal of Logic Programming*, 27(3):185–226, 1996.

[CFFS11] Michael Codish, Yoav Fekete, Carsten Fuhs, and Peter Schneider-Kamp. Optimal base encodings for pseudo-Boolean constraints. In *Proc. 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '11)*, volume 6605 of *LNCS*, pages 189–204. Springer, 2011.

[CFGS10] Michael Codish, Carsten Fuhs, Jürgen Giesl, and Peter Schneider-Kamp. Lazy abstraction for size-change termination. In *Proc. 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR (Yogyakarta) '10)*, volume 6397 of *LNCS (ARCoSS)*, pages 217–232. Springer, 2010.

[CGB+11] Michael Codish, Igor Gonopolskiy, Amir Ben-Amram, Carsten Fuhs, and Jürgen Giesl. SAT-based termination analysis using monotonicity constraints over the integers. *Theory and Practice of Logic Programming, Proc. 27th International Conference on Logic Programming (ICLP '11)*, 11(4–5):503–520, 2011.

[CGST12] Michael Codish, Jürgen Giesl, Peter Schneider-Kamp, and René Thiemann. SAT solving for termination proofs with recursive path orders and dependency pairs. *Journal of Automated Reasoning*, 49(1):53–93, 2012.

[CLS05] Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Testing for termination with monotonicity constraints. In *Proc. 21st International Conference on Logic Programming (ICLP '05)*, volume 3668 of *LNCS*, pages 326–340. Springer, 2005.

[CLS06] Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Solving partial order constraints for LPO termination. In *Proc. 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *LNCS*, pages 4–18. Springer, 2006.

[CLS08] Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Solving partial order constraints for LPO termination. *Journal on Satisfiability, Boolean Modelling and Computation*, 5:193–215, 2008.

[CLSS06] Michael Codish, Vitaly Lagoon, Peter Schachte, and Peter J. Stuckey. Size-change termination analysis in $k$-bits. In *Proc. 15th European Symposium on Programming (ESOP '06)*, volume 3924 of *LNCS*, pages 230–245. Springer, 2006.

[CMTU05] Evelyne Contejean, Claude Marché, Ana Paula Tomás, and Xavier Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.

[Cod08] Michael Codish. Proving termination with (Boolean) satisfaction. In *Proc. 17th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR '07)*, volume 4915 of *LNCS*, pages 1–7. Springer, 2008.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*, pages 151–158. ACM Press, 1971.

[Coq10] The Coq Development Team. The Coq proof assistant, reference manual, version 8.3, 2010.

[CPR05] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Abstraction refinement for termination. In *Proc. 12th International Symposium on Static Analysis (SAS '05)*, volume 3672 of *LNCS*, pages 87–101. Springer, 2005.

[CPR06] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Terminator: Beyond safety. In *Proc. 18th International Conference on Computer Aided Verification (CAV '06)*, volume 4144 of *LNCS*, pages 415–418. Springer, 2006.

[CPR09] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Summarization for termination: no return! *Formal Methods in System Design*, 35(3):369–387, 2009.

[CPU+10] Evelyne Contejean, Andrey Paskevich, Xavier Urbain, Pierre Courtieu, Olivier Pons, and Julien Forest. A3PAT, an approach for certified automated termination proofs. In *Proc. 17th ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM '10)*, pages 63–72. ACM Press, 2010.

[CS02] Michael Colón and Henny Sipma. Practical methods for proving program termination. In *Proc. 14th International Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *LNCS*, pages 442–454. Springer, 2002.

[CSL+06] Michael Codish, Peter Schneider-Kamp, Vitaly Lagoon, René Thiemann, and Jürgen Giesl. SAT solving for argument filterings. In *Proc. 13th International*

*Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '06)*, volume 4246 of *LNAI*, pages 30–44. Springer, 2006.

[CT99] Michael Codish and Cohavit Taboch. A semantic basis for termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999.

[DD94] Danny De Schreye and Stefaan Decorte. Termination of logic programs: The never-ending story. *Journal of Logic Programming*, 19/20:199–260, 1994.

[Der79] Nachum Dershowitz. A note on simplification orderings. *Information Processing Letters*, 9(5):212–215, 1979.

[Der82] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.

[Der87] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1–2):69–115, 1987.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[DM79] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[ES04] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proc. 6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004. See also `http://minisat.se/`.

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modelling and Computation*, 2:1–26, 2006.

[EWZ06] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. In *Proc. 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *LNAI*, pages 574–588. Springer, 2006.

[EWZ08] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2–3):195–220, 2008.

[Fal09] Stephan Falke. *Term Rewriting with Built-In Numbers and Collection Data Structures*. PhD thesis, University of New Mexico, Albuquerque, NM, USA, 2009.

[Fed01] Federal Information Processing Standard 197. The advanced encryption standard. Technical report, National Institute of Standards and Technology, 2001.

[FGM+07] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. 10th International Conference on Theory and Applications of Satisfiability Testing (SAT '07)*, volume 4501 of *LNCS*, pages 340–354. Springer, 2007.

[FGM+08] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. Maximal termination. In *Proc. 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *LNCS*, pages 110–125. Springer, 2008.

[FGP+09] Carsten Fuhs, Jürgen Giesl, Martin Plücker, Peter Schneider-Kamp, and Stephan Falke. Proving termination of integer term rewriting. In *Proc. 20th International Conference on Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *LNCS*, pages 32–47. Springer, 2009.

[FGP+11] Carsten Fuhs, Jürgen Giesl, Michael Parting, Peter Schneider-Kamp, and Stephan Swiderski. Proving termination by dependency pairs and inductive theorem proving. *Journal of Automated Reasoning*, 47(2):133–160, 2011.

[FK08] Stephan Falke and Deepak Kapur. Dependency pairs for rewriting with built-in numbers and semantic data structures. In *Proc. 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *LNCS*, pages 94–109. Springer, 2008.

[FK09] Stephan Falke and Deepak Kapur. A term rewriting approach to the automated termination analysis of imperative programs. In *Proc. 22nd International Conference on Automated Deduction (CADE '09)*, volume 5663 of *LNAI*, pages 277–293. Springer, 2009.

[FK10] Stephan Falke and Deepak Kapur. Termination of context-sensitive rewriting with built-in numbers and collection data structures. In *Proc. 18th International Workshop on Functional and Constraint Logic Programming (WFLP '09)*, volume 5979 of *LNCS*, pages 44–61. Springer, 2010.

[FK11] Carsten Fuhs and Cynthia Kop. Harnessing first order termination provers using higher order dependency pairs. In *Proc. 8th International Symposium Frontiers of Combining Systems (FroCoS '11)*, volume 6989 of *LNAI*, pages 147–162. Springer, 2011.

[FKS11a] Stephan Falke, Deepak Kapur, and Carsten Sinz. Termination analysis of C programs using compiler intermediate languages. In *Proc. 22nd International Conference on Rewriting Techniques and Applications (RTA '11)*, volume 10 of *LIPIcs*, pages 41–50. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[FKS11b] Stephan Falke, Deepak Kapur, and Carsten Sinz. Termination analysis of C programs using compiler intermediate languages. Technical Report Karlsruhe Report in Informatics 2011-6, Karlsruhe Institute of Technology, 2011.

[FNO+08] Carsten Fuhs, Rafael Navarro-Marset, Carsten Otto, Jürgen Giesl, Salvador Lucas, and Peter Schneider-Kamp. Search techniques for rational polynomial orders. In *Proc. 9th International Conference on Artificial Intelligence and Symbolic Computation (AISC '08)*, volume 5144 of *LNAI*, pages 109–124. Springer, 2008.

[FS10] Carsten Fuhs and Peter Schneider-Kamp. Synthesizing shortest straight-line programs over GF(2) using SAT. In *Proc. 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *LNCS*, pages 71–84. Springer, 2010.

[Fuh07] Carsten Fuhs. SAT-based methods for automated termination analysis with polynomial orderings. Diploma thesis, April 2007. RWTH Aachen, Germany.

[Fuh09] Carsten Fuhs. SAT instances for termination analysis with AProVE. In Daniel Le Berre, Olivier Roussel, Laurent Simon, Vasco Manquinho, Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes, editors, *SAT 2009 competitive events booklet: preliminary version*, pages 63–67. 2009. Available at `http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf`.

[Gat06] Andreas Gathmann. Tropical algebraic geometry. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 108:3–32, 2006.

[Ges90] Alfons Geser. *Relative Termination*. PhD thesis, Universität Passau, Germany, 1990.

[GHW04] Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting systems. *Applicable Algebra in Engineering, Communication and Computing*, 15(3–4):149–171, 2004.

[GHW07] Andreas Gebhardt, Dieter Hofbauer, and Johannes Waldmann. Matrix evolutions. In *Proc. 9th International Workshop on Termination (WST '07)*, pages 4–8, 2007.

[Gie95] Jürgen Giesl. Termination analysis for functional programs using term orderings. In *Proc. 2nd International Symposium on Static Analysis (SAS '95)*, volume 983 of *LNCS*, pages 154–171. Springer, 1995.

[GLP06] Olga Grinchtein, Martin Leucker, and Nir Piterman. Inferring network invariants automatically. In *Proc. 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *LNAI*, pages 483–497. Springer, 2006.

[Goo98] Joshua T. Goodman. *Parsing inside-out*. PhD thesis, Harvard University, 1998.

[GRS+11] Jürgen Giesl, Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, and René Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM Transactions on Programming Languages and Systems*, 33(2):1–39, 2011.

[GSST06] Jürgen Giesl, Stephan Swiderski, Peter Schneider-Kamp, and René Thiemann. Automated termination analysis for Haskell: From term rewriting to programming languages. In *Proc. 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *LNCS*, pages 297–312. Springer, 2006.

[GST06] Jürgen Giesl, Peter Schneider-Kamp, and René Thiemann. AProVE 1.2: automatic termination proofs in the dependency pair framework. In *Proc. 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *LNAI*, pages 281–286. Springer, 2006.

[GTS05a] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '04)*, volume 3452 of *LNAI*, pages 301–331. Springer, 2005.

[GTS05b] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. 5th International Workshop on Frontiers of Combining Systems (FroCoS '05)*, volume 3717 of *LNAI*, pages 216–231. Springer, 2005.

[GTSF06] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.

[GTSS07] Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp. Proving termination by bounded increase. In *Proc. 21st International Conference on Automated Deduction (CADE '07)*, volume 4603 of *LNAI*, pages 443–459. Springer, 2007.

[GWB98] Jürgen Giesl, Christoph Walther, and Jürgen Brauburger. Termination analysis for functional programs. *Automated Deduction - A Basis for Applications*, 3:135–164, 1998.

[HJ98] Hoon Hong and Dalibor Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.

[HJN11] Antti Eero Johannes Hyvärinen, Tommi A. Junttila, and Ilkka Niemelä. Grid-based SAT solving with iterative partitioning and clause learning. In *Proc. 17th International Conference on Principles and Practice of Constraint Programming (CP '11)*, volume 6876 of *LNCS*, pages 385–399. Springer, 2011.

[HKW10] Dieter Hofbauer, Adam Koprowski, and Johannes Waldmann. Tropical termination. Talk by Johannes Waldmann at the TeReSe workshop 2010/1, Aachen, Germany, May 2010.

[HL86] Thérèse Hardin and Alain Laville. Proof of termination of the rewriting system SUBST on CCL. *Theoretical Computer Science*, 46(2–3):305–312, 1986.

[HM05] Nao Hirokawa and Aart Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1–2):172–199, 2005.

[HM06] Nao Hirokawa and Aart Middeldorp. Predictive labeling. In *Proc. 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *LNCS*, pages 313–327. Springer, 2006.

[HM07] Nao Hirokawa and Aart Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.

[HM08] Nao Hirokawa and Georg Moser. Automated complexity analysis based on the dependency pair method. In *Proc. 4th International Joint Conference on Automated Reasoning (IJCAR '08)*, volume 5195 of *LNAI*, pages 364–379. Springer, 2008.

[HMZ08] Nao Hirokawa, Aart Middeldorp, and Harald Zankl. Uncurrying for termination. In *Proc. 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '08)*, volume 5330 of *LNAI*, pages 667–681. Springer, 2008.

[Hof01] Dieter Hofbauer. Termination proofs by context-dependent interpretations. In *Proc. 12th International Conference on Rewriting Techniques and Applications (RTA '01)*, volume 2051 of *LNCS*, pages 108–121. Springer, 2001.

[Hol95] Christian Holzbaur. OFAI clp(q,r) manual, edition 1.3.3. Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna, Austria, 1995.

[HW06] Dieter Hofbauer and Johannes Waldmann. Termination of string rewriting with matrix interpretations. In *Proc. 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *LNCS*, pages 328–342. Springer, 2006.

[JB04] Neil D. Jones and Nina Bohr. Termination analysis of the untyped $\lambda$-calculus. In *Proc. 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *LNCS*, pages 1–23. Springer, 2004.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proc. 14th Annual ACM Symposium on Principles of Programming Languages (POPL '87)*, pages 111–119. ACM Press, 1987.

[JLN+10] Michael Jünger, Thomas Liebling, Denis Naddef, George Nemhauser, William Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence Wolsey, editors. *50 Years of Integer Programming 1958–2008*. Springer, 2010.

[KB70] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. *Computational Problems in Abstract Algebra*, pages 263–297, 1970.

[KH11] Dominik Klein and Nao Hirokawa. Maximal completion. In *Proc. 22nd International Conference on Rewriting Techniques and Applications (RTA '11)*, volume 10 of *LIPIcs*, pages 71–80. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[KK99a] Hisashi Kondo and Masahito Kurihara. Design and heuristics for BDD-based automated termination verification system for rule-based programs. In *Proc. 12th IEEE International Conference on Systems, Man, and Cybernetics (SMC '99)*, volume 5, pages 738–743. IEEE Computer Society, 1999.

[KK99b] Masahito Kurihara and Hisashi Kondo. Heuristics and experiments on BDD representation of boolean functions for expert systems in software verification domains. In *Proc. 12th Australian Joint Conference on Artificial Intelligence (AI '99)*, volume 1747 of *LNAI*, pages 353–364. Springer, 1999.

[KK04] Masahito Kurihara and Hisashi Kondo. Efficient BDD encodings for partial order constraints with application to expert systems in software verification. In *Proc. 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE '04)*, volume 3029 of *LNAI*, pages 827–837. Springer, 2004.

[KKY09] Arist Kojevnikov, Alexander S. Kulikov, and Grigory Yaroslavtsev. Finding efficient circuits using SAT-solvers. In *Proc. 12th International Conference on Theory and Applications of Satisfiability Testing (SAT '09)*, volume 5584 of *LNCS*, pages 32–44. Springer, 2009.

[KL80] Sam Kamin and Jean-Jacques Lévy. Two generalizations of the recursive path ordering. Unpublished Manuscript, University of Illinois, Urbana, IL, USA, 1980.

[KM07] Adam Koprowski and Aart Middeldorp. Predictive labeling with dependency pairs using SAT. In *Proc. 21st International Conference on Automated Deduction (CADE '07)*, volume 4603 of *LNAI*, pages 410–425. Springer, 2007.

[Kop06] Adam Koprowski. TPA: Termination proved automatically. In *Proc. 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *LNCS*, pages 257–266. Springer, 2006. `http://www.win.tue.nl/tpa/`.

[KOR93] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121(1–2):279–308, 1993.

[Kro98] Daniel Krob. Some automata-theoretic aspects of min-max-plus semirings. In Jeremy Gunawardena, editor, *Idempotency*, pages 70–79. Cambridge University Press, 1998.

[KST+09] Daniel Kroening, Natasha Sharygina, Stefano Tonetta, Aliaksei Tsitovich, and Christoph M. Wintersteiger. Loopfrog: A static analyzer for ANSI-C programs. In *Proc. 24th IEEE/ACM International Conference on Automated Software Engineering (ASE '09)*, pages 668–670. IEEE Computer Society, 2009.

[KST+11] Alexander Krauss, Christian Sternagel, René Thiemann, Carsten Fuhs, and Jürgen Giesl. Termination of Isabelle functions via termination of rewriting. In *Proc. 2nd International Conference on Interactive Theorem Proving (ITP '11)*, volume 6898 of *LNCS*, pages 152–167. Springer, 2011.

[KSZM09] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In *Proc. 20th International Conference on Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *LNCS*, pages 295–304. Springer, 2009.

[KV03] Konstantin Korovin and Andrei Voronkov. Orienting rewrite rules with the Knuth-Bendix order. *Information and Computation*, 183(2):165–186, 2003.

[KW08] Adam Koprowski and Johannes Waldmann. Arctic termination ...below zero. In *Proc. 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *LNCS*, pages 202–216. Springer, 2008.

[KW09] Adam Koprowski and Johannes Waldmann. Max/plus tree automata for termination of term rewriting. *Acta Cybernetica*, 19(2):357–392, 2009.

[LA04] Chris Lattner and Vikram S. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. 2nd IEEE / ACM International Symposium on Code Generation and Optimization (CGO '04)*, pages 75–88. IEEE Computer Society, 2004.

[Lan79] Dallas Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

[Lee09] Chin Soon Lee. Ranking functions for size-change termination. *ACM Transactions on Programming Languages and Systems*, 31(3):1–42, 2009.

[Les83] Pierre Lescanne. Computer experiments with the REVE term rewriting system generator. In *Proc. 10th ACM Symposium on Principles of Programming Languages (POPL '83)*, pages 99–108. ACM Press, 1983.

[LJB01] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *Proc. 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '01)*, volume 36 of *ACM SIGPLAN Notices*, pages 81–92. ACM Press, 2001.

[LM08] Salvador Lucas and José Meseguer. Order-sorted dependency pairs. In *Proc. 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP '08)*, pages 108–119. ACM Press, 2008.

[LMS03] Vitaly Lagoon, Frédéric Mesnard, and Peter J. Stuckey. Termination analysis with types is more accurate. In *Proc. 19th International Conference on Logic Programming (ICLP '03)*, volume 2916 of *LNCS*, pages 254–268. Springer, 2003.

[LP10] Daniel Le Berre and Anne Parrain. The SAT4J library, release 2.2. *Journal on Satisfiability, Boolean Modelling and Computation*, 7:59–64, 2010.

[Luc98] Salvador Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1):1–61, 1998.

[Luc02] Salvador Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.

[Luc05] Salvador Lucas. Polynomials over the reals in proofs of termination: from theory to practice. *RAIRO - Theoretical Informatics and Applications*, 39(3):547–586, 2005.

[Luc07] Salvador Lucas. Practical use of polynomials over the reals in proofs of termination. In *Proc. 9th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP '07)*, pages 39–50. ACM Press, 2007.

[Luc10] Salvador Lucas. From matrix interpretations over the rationals to matrix interpretations over the naturals. In *Proc. 10th International Conference on Artificial Intelligence and Symbolic Computation (AISC '10)*, volume 6167 of *LNAI*, pages 116–131. Springer, 2010.

[Mar08] Joao Marques-Silva. Practical applications of boolean satisfiability. In *Proc. 9th International Workshop on Discrete Event Systems (WODES '08)*, pages 74–80. IEEE Computer Society, 2008.

[Mat70] Yuri Matiyasevich. Enumerable sets are Diophantine. *Soviet Mathematics (Dokladi)*, 11(2):354–357, 1970.

[MCLS11] Amit Metodi, Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Boolean equi-propagation for optimized SAT encoding. In *Proc. 17th International Conference on Principles and Practice of Constraint Programming (CP '11)*, volume 6876 of *LNCS*, pages 621–636. Springer, 2011.

[MJ10] Filip Marić and Predrag Janičić. URBiVA: Uniform reduction to bit-vector arithmetic. In *Proc. 5th International Joint Conference on Automated Reasoning (IJCAR '10)*, volume 6173 of *LNAI*, pages 346–352. Springer, 2010.

[MLM09] João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press, 2009.

[MN70] Zohar Manna and Steven Ness. On the termination of Markov algorithms. In *Proc. 3rd Hawaii International Conference on System Science (HICSS '70)*, pages 789–792, 1970.

[MP09] Jean-Yves Marion and Romain Péchoux. Sup-interpretations, a semantic method for static analysis of program resources. *ACM Transactions on Computational Logic*, 10(4):1–31, 2009.

[MS08] Georg Moser and Andreas Schnabl. Proving quadratic derivational complexities using context dependent interpretations. In *Proc. 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *LNCS*, pages 276–290. Springer, 2008.

[MSW08] Georg Moser, Andreas Schnabl, and Johannes Waldmann. Complexity analysis of term rewriting based on matrix and context dependent interpretations. In *Proc. 28th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '08)*, volume 2 of *LIPIcs*, pages 304–315. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008.

[MV06] Panagiotis Manolios and Daron Vroon. Termination analysis with calling context graphs. In *Proc. 18th International Conference on Computer-Aided Verification (CAV '06)*, volume 4144 of *LNCS*, pages 401–414. Springer, 2006.

[MZ07] Claude Marché and Hans Zantema. The termination competition. In *Proc. 18th International Conference on Rewriting Techniques and Applications (RTA '07)*, volume 4533 of *LNCS*, pages 303–313. Springer, 2007. See also `http://termination-portal.org/wiki/Termination_Competition`.

[NDGS11] Manh Thang Nguyen, Danny De Schreye, Jürgen Giesl, and Peter Schneider-Kamp. Polytool: Polynomial interpretations as a basis for termination analysis of logic programs. *Theory and Practice of Logic Programming*, 11(1):33–63, 2011.

[NEG11] Lars Noschinski, Fabian Emmes, and Jürgen Giesl. A dependency pair framework for innermost complexity analysis of term rewrite systems. In *Proc. 23rd International Conference on Automated Deduction (CADE '11)*, volume 6803 of *LNAI*, pages 422–438. Springer, 2011.

[NGSD08] Manh Thang Nguyen, Jürgen Giesl, Peter Schneider-Kamp, and Danny De Schreye. Termination analysis of logic programs based on dependency graphs. In *Proc. 17th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR '07)*, volume 4915 of *LNCS*, pages 8–22. Springer, 2008.

[NMZ10] Friedrich Neurauter, Aart Middeldorp, and Harald Zankl. Monotonicity criteria for polynomial interpretations over the naturals. In *Proc. 5th International Joint Conference on Automated Reasoning (IJCAR '10)*, volume 6173 of *LNAI*, pages 502–517. Springer, 2010.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[OBEG10] Carsten Otto, Marc Brockschmidt, Christian von Essen, and Jürgen Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *Proc. 21st International Conference on Rewriting Techniques and Applications (RTA '10)*, volume 6 of *LIPIcs*, pages 259–276. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[Ohl01] Enno Ohlebusch. Termination of logic programs: Transformational methods revisited. *Applicable Algebra in Engineering, Communication and Computing*, 12(1–2):73–116, 2001.

[PR04a] Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In *Proc. 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '04)*, volume 2937 of *LNCS*, pages 239–251. Springer, 2004.

[PR04b] Andreas Podelski and Andrey Rybalchenko. Transition invariants. In *Proc. 19th IEEE Symposium on Logic in Computer Science (LICS '04)*, pages 32–41. IEEE Computer Society, 2004.

[PS97] Sven Eric Panitz and Manfred Schmidt-Schauß. TEA: Automatically proving termination of programs in a non-strict higher-order functional language. In *Proc. 4th International Symposium on Static Analysis (SAS '97)*, volume 1302 of *LNCS*, pages 345–360. Springer, 1997.

[Sch08] Peter Schneider-Kamp. Static termination analysis for Prolog using term rewriting and SAT solving. PhD thesis AIB-2008-17, RWTH Aachen, December 2008.

[Sch11] Peter Schneider-Kamp. Personal communication, 2011.

[SES+12] Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, and Carsten Fuhs. A linear operational semantics for termination and complexity analysis of ISO Prolog. In *Proc. 21st International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR '11)*, volume 7225 of *LNCS*, pages 237–252. Springer, 2012.

[SGN10] Peter Schneider-Kamp, Jürgen Giesl, and Manh Thang Nguyen. The dependency triple framework for termination of logic programs. In *Proc. 19th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR '09)*, volume 6037 of *LNCS*, pages 37–51. Springer, 2010.

[SGS+10] Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, and René Thiemann. Automated termination analysis for logic programs with cut. *Theory and Practice of Logic Programming, Proc. 26th International Conference on Logic Programming (ICLP '10)*, 10(4–6):365–381, 2010.

[SGST09] Peter Schneider-Kamp, Jürgen Giesl, Alexander Serebrenik, and René Thiemann. Automated termination proofs for logic programs by term rewriting. *ACM Transactions on Computational Logic*, 11(1):1–52, 2009.

[SJ05] Damien Sereni and Neil D. Jones. Termination analysis of higher-order functional programs. In *Proc. 3rd Asian Symposium on Programming Languages and Systems (APLAS '05)*, volume 3780 of *LNCS*, pages 281–297. Springer, 2005.

[SM08] Christian Sternagel and Aart Middeldorp. Root-labeling. In *Proc. 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *LNCS*, pages 336–350. Springer, 2008.

[Sma04] Jan-Georg Smaus. Termination of logic programs using various dynamic selection rules. In *Proc. 20th International Conference on Logic Programming (ICLP '04)*, volume 3132 of *LNCS*, pages 43–57. Springer, 2004.

[SMP10] Fausto Spoto, Fred Mesnard, and Étienne Payet. A termination analyser for Java Bytecode based on path-length. *ACM Transactions on Programming Languages and Systems*, 32(3):1–70, 2010.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Proc. 12th International Conference on Theory and Applications of Satisfiability Testing (SAT '09)*, volume 5584 of *LNCS*, pages 244–257. Springer, 2009.

[SPG+09] Stephan Swiderski, Michael Parting, Jürgen Giesl, Carsten Fuhs, and Peter Schneider-Kamp. Termination analysis by dependency pairs and inductive theorem proving. In *Proc. 22nd International Conference on Automated Deduction (CADE '09)*, volume 5663 of *LNAI*, pages 322–338. Springer, 2009.

[ST10] Christian Sternagel and René Thiemann. Certification extends termination techniques. In *Proc. 11th International Workshop on Termination (WST '10)*, 2010.

[ST11a] Christian Sternagel and René Thiemann. Generalized and formalized uncurrying. In *Proc. 8th International Symposium Frontiers of Combining Systems (FroCoS '11)*, volume 6989 of *LNAI*, pages 243–258. Springer, 2011.

[ST11b] Christian Sternagel and René Thiemann. Modular and certified semantic labeling and unlabeling. In *Proc. 22nd International Conference on Rewriting Techniques and Applications (RTA '11)*, volume 10 of *LIPIcs*, pages 329–344. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[STA+07] Peter Schneider-Kamp, René Thiemann, Elena Annov, Michael Codish, and Jürgen Giesl. Proving termination using recursive path orders and SAT solving. In *Proc. 6th International Symposium on Frontiers of Combining Systems (FroCoS '07)*, volume 4720 of *LNAI*, pages 267–282. Springer, 2007.

[Ste95] Joachim Steinbach. Simplification orderings: History of results. *Fundamenta Informaticae*, 24(1-2):47–87, 1995.

[TG05] René Thiemann and Jürgen Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 16(4):229–270, 2005.

[Thi07] René Thiemann. The DP framework for proving termination of term rewriting. PhD thesis AIB-2007-17, RWTH Aachen, October 2007.

[Thi11] René Thiemann. Personal communication, 2011.

[Tiw04] Ashish Tiwari. Termination of linear programs. In *Proc. 16th International Conference on Computer Aided Verification (CAV '04)*, volume 3114 of *LNCS*, pages 70–82. Springer, 2004.

[TS09] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In *Proc. 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs '09)*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009.

[Tse68] Grigori Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125. 1968. Reprinted in Jörg Siekmann and Graham Wrightson (editors), *Automation of Reasoning*, 2:466–483, 1983.

[TSWK11] Aliaksei Tsitovich, Natasha Sharygina, Christoph M. Wintersteiger, and Daniel Kroening. Loop summarization and termination analysis. In *Proc. 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '11)*, volume 6605 of *LNCS*, pages 81–95. Springer, 2011.

[TTKB09] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

[Tur49] Alan M. Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69, 1949.

[Wal94] Christoph Walther. On proving the termination of algorithms by machine. *Artificial Intelligence*, 71(1):101–157, 1994.

[Wal07] Johannes Waldmann. Weighted automata for proving termination of string rewriting. *Journal of Automata, Languages and Combinatorics*, 12(4):545–570, 2007.

[Xi02] Hongwei Xi. Dependent types for program termination verification. *Higher-Order and Symbolic Computation*, 15(1):91–131, 2002.

[Yam01] Toshiyuki Yamada. Confluence and termination of simply typed term rewriting systems. In *Proc. 12th International Conference on Rewriting Techniques and Applications (RTA'01)*, volume 2051 of *LNCS*, pages 338–352. Springer, 2001.

[Zan94] Hans Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17(1):23–50, 1994.

[Zan95] Hans Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24(1–2):89–105, 1995.

[Zan03] Hans Zantema. Termination. In Terese, editor, *Term Rewriting Systems*, chapter 6, pages 181–259. Cambridge University Press, 2003.

[Zan09] Harald Zankl. *Lazy Termination Analysis*. PhD thesis, University of Innsbruck, Austria, 2009.

[ZHM07] Harald Zankl, Nao Hirokawa, and Aart Middeldorp. Constraints for argument filterings. In *Proc. 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '07)*, volume 4362 of *LNCS*, pages 579–590. Springer, 2007.

[ZHM09] Harald Zankl, Nao Hirokawa, and Aart Middeldorp. KBO orientability. *Journal of Automated Reasoning*, 43(2):173–201, 2009.

[ZM06] Harald Zankl and Aart Middeldorp. KBO as a satisfaction problem. In *Proc. 8th International Workshop on Termination (WST '06)*, pages 55–59, 2006.

[ZM07] Harald Zankl and Aart Middeldorp. Satisfying KBO constraints. In *Proc. 18th International Conference on Rewriting Techniques and Applications (RTA '07)*, volume 4533 of *LNCS*, pages 389–403. Springer, 2007.

[ZM08] Harald Zankl and Aart Middeldorp. Increasing interpretations. In *Proc. 9th International Conference on Artificial Intelligence and Symbolic Computation (AISC '08)*, volume 5144 of *LNAI*, pages 191–205. Springer, 2008.

[ZM09] Harald Zankl and Aart Middeldorp. Increasing interpretations. *Annals of Mathematics and Artificial Intelligence*, 56(1):87–108, 2009.

[ZM10] Harald Zankl and Aart Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In *Proc. 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR (Dakar) '10)*, volume 6355 of *LNAI*, pages 481–500. Springer, 2010.

[ZSHM10] Harald Zankl, Christian Sternagel, Dieter Hofbauer, and Aart Middeldorp. Finding and certifying loops. In *Proc. 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '10)*, volume 5901 of *LNCS*, pages 755–766. Springer, 2010.

[ZW07] Hans Zantema and Johannes Waldmann. Termination by quasi-periodic interpretations. In *Proc. 18th International Conference on Rewriting Techniques and Applications (RTA '07)*, volume 4533 of *LNCS*, pages 404–418. Springer, 2007.

# Index


$(\mathcal{P},\mathcal{R})$-chain, 26
    minimal, 26
$DOM$, 24
$\mathcal{P}os(\cdot)$, 23
$\oplus$, 78
$\mathbb{A}_{\mathbb{N}}$, 78
$\mathbb{A}_{\mathbb{Z}}$, 78
$\otimes$, 78
$\circ$, 28
$||\cdot||_{bin}$, 86
$||\cdot||_{un}$, 91
$\oplus$, 35
$root(\cdot)$, 24
$\mathcal{F}(\cdot)$, 23
$\trianglerighteq$, 24
$\triangleright$, 24
$[\cdot]^{left}$, 41, 43
$[\cdot]^{right}$, 41, 43
$\rightarrow_{\mathcal{R}}$, 24
$\mathcal{V}ar(\cdot)$, 23
$con$, 41
$ncon$, 41
$t[s]_{\pi}$, 24

algebra
    max/plus, 78
    weakly monotone, 30
arctic addition, 78
arctic constraint, 80
arctic multiplication, 78
arctic semi-ring, 78
arctic variable, 80
arity, 23

base term, 110

CDCL, 90
chain, 26
    minimal, 26
compatible, 28
composition, 28
constant, 23
constant part, 41
constant symbol, 23
Constraint Satisfaction Problem, 19
CSP, 19

decreasing
    strictly, 28
    weakly, 28
defined symbol, 25
dependency graph, 29
dependency pair, 25
dependency pair framework, 27
dependency pair problem, 26
    tuple-typed, 110
dependency pair processor, 27
dimension, 79
Diophantine constraint, 34
domain, 24
DP, 25
DP framework, 27
DP problem, 26
    tuple-typed, 110
DP processor, 27

finite, 79
formula
    SMT-NIA, 34
function symbol, 23

global constraint, 90
ground term, 23

independent, 50
infinity bit, 86, 91
interpretation, 30
    arctic matrix, 79
    polynomial, 31

KBO, 5

left-hand side, 24
level mapping, 117
lhs, 24
LPO, 4

max/plus algebra, 78
max/plus semi-ring, 78
monotonic, 24

non-constant part, 41
normal form, 24

orient, 28

parametric coefficient, 33
PB constraint, 3
polynomial interpretation, 31
    with negative constants, 40
position, 23
positive, 79
processor, 27
Pseudo-Boolean constraint, 3

reduce, 24
reduction pair, 28
    strongly monotonic, 28
    weakly monotonic, 28
reduction pair processor, 28
rewrite relation, 24
rewrite rule, 24
rewrite sequence, 24
rewrite step, 24
rhs, 24
right-hand side, 24
root position, 24
root symbol, 24
rule, 24

SAT problem, 3
SAT solver, 3
satisfiability solver, 3
satisfy, 113
SCNP property, 113
SCNP reduction pair, 118
SCT, 103
SCT property, 103
signature, 23
size-change termination, 103
SMT, 3
SMT-NIA formula, 34
SRS, 3, 83
stable, 24
standard DP problem, 108
string rewrite system, 3, 83
substitution, 24
subterm, 24
    strict, 24

term, 23
term rewrite relation, 24
term rewrite step, 24
term rewrite system, 24
term size, 24
terminating, 24
Termination Competition, 2
TPDB, 2
TRS, 24

tuple property, 108
tuple symbol, 25, 109
tuple term, 110
tuple typing, 110
tuple-typed reduction pair, 112

variable, 23
variable coefficient, 33

# Curriculum Vitae

| | |
|---|---|
| Name | Carsten Fuhs |
| Geburtsdatum | 6. November 1978 |
| Geburtsort | Düren |

## Bildungsgang

| | |
|---|---|
| 1989–1998 | Städt. Gymnasium am Wirteltor Düren<br>Abschluss: Allgemeine Hochschulreife |
| 1998–1999 | Zivildienst am St.-Augustinus-Krankenhaus Düren |
| 1999–2007 | Studium der Informatik an der RWTH Aachen<br>Abschluss: Diplom |
| 2007–2011 | Wissenschaftlicher Angestellter am Lehr- und Forschungsgebiet Informatik 2 (Prof. Dr. Jürgen Giesl), RWTH Aachen |
| 2012– | Post-Doctoral Research Assistant, Queen Mary, University of London und University College London |

# Aachener Informatik-Berichte

**This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:**

`http://aib.informatik.rwth-aachen.de/`

**To obtain copies please consult the above URL or send your request to:**

**Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,**
**Email:** `biblio@informatik.rwth-aachen.de`

2009-01 * Fachgruppe Informatik: Jahresbericht 2009

2009-02 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications

2009-03 Alexander Nyßen: Model-Based Construction of Embedded Real-Time Software - A Methodology for Small Devices

2009-05 George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs

2009-06 George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete

2009-07 Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I

2009-08 Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs

2009-09 Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem

2009-10 Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm

2009-11 Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs

2009-12 Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes

2009-13 Martin Zimmermann: Time-optimal Winning Strategies for Poset Games

2009-14 Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)

2009-15 Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems

2009-16 George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs

2009-17 Carsten Kern: Learning Communicating and Nondeterministic Automata

2009-18 Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

2010-01 * Fachgruppe Informatik: Jahresbericht 2010

2010-02 Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time

2010-03 Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering

2010-04 René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme

2010-05 Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme

2010-06 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata

2010-07 George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms

2010-08 Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting

2010-09 George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs

2010-10 Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut

2010-11 Martin Zimmermann: Parametric LTL Games

2010-12 Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut

2010-13 Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems

2010-14 Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination

2010-15 Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode

2010-16 Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles

2010-17 Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten

2010-18 Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit

2010-19 Felix Reidl: Experimental Evaluation of an Independent Set Algorithm

2010-20 Wladimir Fridman, Christof Löding, Martin Zimmermann: Degrees of Lookahead in Context-free Infinite Games

2011-01 * Fachgruppe Informatik: Jahresbericht 2011

2011-02 Marc Brockschmidt, Carsten Otto, Jürgen Giesl: Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting

2011-03 Lars Noschinski, Fabian Emmes, Jürgen Giesl: A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems

2011-04 Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, Thomas Noll: A Local Greibach Normal Form for Hyperedge Replacement Grammars

2011-06 Johannes Lotz, Klaus Leppkes, and Uwe Naumann: dco/c++ - Derivative Code by Overloading in C++

2011-07 Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe: An Operational Semantics for Activity Diagrams using SMV

2011-08 Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, Carsten Fuhs: A Linear Operational Semantics for Termination and Complexity Analysis of ISO Prolog

2011-09 Markus Beckers, Johannes Lotz, Viktor Mosenkis, Uwe Naumann (Editors): Fifth SIAM Workshop on Combinatorial Scientific Computing

2011-10 Markus Beckers, Viktor Mosenkis, Michael Maier, Uwe Naumann: Adjoint Subgradient Calculation for McCormick Relaxations

2011-11 Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, Bernd Becker: Hierarchical Counterexamples for Discrete-Time Markov Chains

2011-12 Ingo Felscher, Wolfgang Thomas: On Compositional Failure Detection in Structured Transition Systems

2011-13 Michael Förster, Uwe Naumann, Jean Utke: Toward Adjoint OpenMP

2011-14 Daniel Neider, Roman Rabinovich, Martin Zimmermann: Solving Muller Games via Safety Games

2011-16 Niloofar Safiran, Uwe Naumann: Toward Adjoint OpenFOAM

2011-18 Kamal Barakat: Introducing Timers to pi-Calculus

2011-19 Marc Brockschmidt, Thomas Ströder, Carsten Otto, Jürgen Giesl: Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode

2011-24 Callum Corbett, Uwe Naumann, Alexander Mitsos: Demonstration of a Branch-and-Bound Algorithm for Global Optimization using McCormick Relaxations

2011-25 Callum Corbett, Michael Maier, Markus Beckers, Uwe Naumann, Amin Ghobeity, Alexander Mitsos: Compiler-Generated Subgradient Code for McCormick Relaxations

2011-26 Hongfei Fu: The Complexity of Deciding a Behavioural Pseudometric on Probabilistic Automata

2012-01 * Fachgruppe Informatik: Annual Report 2012

2012-02 Thomas Heer: Controlling Development Processes

2012-03 Arne Haber, Jan Oliver Ringert, Bernhard Rumpe: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems

2012-04 Marcus Gelderie: Strategy Machines and their Complexity

2012-05 Thomas Ströder, Fabian Emmes, Jürgen Giesl, Peter Schneider-Kamp, and Carsten Fuhs: Automated Complexity Analysis for Prolog by Term Rewriting

2012-06 Marc Brockschmidt, Richard Musiol, Carsten Otto, Jürgen Giesl: Automated Termination Proofs for Java Programs with Cyclic Data

2012-07 André Egners, Björn Marschollek, and Ulrike Meyer: Hackers in Your Pocket: A Survey of Smartphone Security Across Platforms

2012-08 Hongfei Fu: Computing Game Metrics on Markov Decision Processes

2012-09 Dennis Guck, Tingting Han, Joost-Pieter Katoen, and Martin R. Neuhäußer: Quantitative Timed Analysis of Interactive Markov Chains

2012-10 Uwe Naumann and Johannes Lotz: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Direct Solvers for Systems of Linear Equations

2012-12 Jürgen Giesl, Thomas Ströder, Peter Schneider-Kamp, Fabian Emmes, and Carsten Fuhs: Symbolic Evaluation Graphs and Term Rewriting — A General Methodology for Analyzing Logic Programs

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.