LIFTED UNIT PROPAGATION

by

Pashootan Vaezipoor

B.Sc., Amirkabir University of Technology, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the
School of Computing Science
Faculty of Applied Sciences

© Pashootan Vaezipoor 2012
SIMON FRASER UNIVERSITY
Spring 2012

All rights reserved. However, in accordance with the *Copyright Act of Canada*, this work may be reproduced without authorization under the conditions for "Fair Dealing." Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

APPROVAL

Name: Pashootan Vaezipoor

Degree: Master of Science

Title of Thesis: Lifted Unit Propagation

Examining Committee: Dr. Oliver Schulte
Chair

____

Dr. David G. Mitchell, Senior Supervisor

____

Dr. Evgenia Ternovska, Supervisor

____

Dr. James Delgrande, SFU Examiner

Date Approved: March 13, 2012 ______________________________

Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website (www.lib.sfu.ca) at http://summit/sfu.ca and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2011

Abstract

Recent emergence of effective solvers for propositional satisfiability (SAT) and related problems has led to new methods for solving computationally challenging industrial problems, such as NP-hard search problems in planning, software design, and hardware verification. This has produced a demand for tools which allow users to write high level problem specifications which are automatically reduced to SAT. We consider the case of specifications in first order logic with reduction to SAT by grounding. For realistic problems, the resulting SAT instances can be prohibitively large. A key technique in SAT solvers is unit propagation, which often significantly reduces instance size before search for a solution begins. We define ”lifted unit propagation”, which is executed before grounding. We show that instances produced by a grounding algorithm with lifted unit propagation are never larger than those produced by normal grounding followed by UP, and demonstrate experimentally that they are sometimes much smaller.

To an everlasting friend,
Karan Vaezipoor

*“Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it.
And to make matters worse: complexity sells better. ’*

— Edsger W. Dijkstra

Acknowledgments

I hereby would like to thank, my senior supervisor, David Mitchell for introducing this line of research and more importantly teaching me how to think and write in terms of logic. I also want to thank my colleagues and lab-mates, specifically Amir Aavani, for their valuable comments and support. Lastly I want to thank my parents for demonstrating by example, the true definition humanity.

Contents

Approval ii

Abstract iii

Dedication iv

Quotation v

Acknowledgments vi

Contents vii

List of Tables ix

List of Figures x

1 Introduction 1

1.1 Structure of the Text 4

2 Preliminaries 6

2.1 First-order Logic 6

2.1.1 Syntax 7

2.1.2 Semantics 8

2.1.3 Equivalence of Formulas and Negation Normal Form 9

2.1.4 Partial Structures 11

2.2 Datalog 14

2.2.1 Syntax 14

2.2.2 Semantics . . . 15
2.3 Some Notes on Computational Complexity . . . 16
2.3.1 NP Search Problems . . . 17

3 FO Model Expansion and Grounding **18**
3.1 Formal Definition of Model Expansion and Grounding . . . 19
3.2 Top-Down Grounding . . . 20
3.3 Relational Algebraic Grounding . . . 21
3.4 Transformation to CNF and Unit Propagation . . . 25
3.4.1 Autarkies and Autark Subformulas . . . 27
3.5 Conclusion . . . 27

4 Lifting Unit Propagation **29**
4.1 Bound Structures . . . 29
4.2 Top-down Grounding over a Bound Structure . . . 31
4.3 LUP Structures . . . 31
4.3.1 A Datalog Program for LUP Structure Construction . . . 36
4.3.2 An Algorithm for Construction of LUP Structures . . . 38
4.3.3 Bottom Up Grounding with LUP Bounds . . . 40

5 Experiments **41**

6 Discussion **45**

Bibliography **47**

List of Tables

4.1 Rules for Bounds Computation 37

5.1 Impact of LUP on the size of the grounding. 42
5.2 Impact of LUP on reduction in grounding and (SAT) solving time. 42
5.3 Comparison between the effectiveness of LUP and GIDL Bounds on reduction in grounding size. 44
5.4 Comparison of solving time for Enfragmo and IDP, with and without LUP/GIDL bounds. 44

List of Figures

1.1 Overview of a typical ground-based system (like Enfragmo). 2

2.1 Ordering on truth values for partial structures. 12

3.1 Top-down grounding steps. 22

3.2 Result of reduced grounding on ground formula tree. 23

4.1 Structures and the vocabularies on which they are defined. 30

Chapter 1

Introduction

In the course of the last decade, we have witnessed a giant leap in the performance of propositional solvers, such as *Satisfiability (SAT) solvers* and *Answer Set Programming (ASP) Solvers*, thanks to research targeted at devising effective methods and heuristics, and also solver competitions held by the respective research communities. Having these efficient solvers, one can now aim for solving real-world problems that arise in industry. But the intrinsic complexity of these problems makes low-level modelling of them, say at SAT level, extremely hard or in some cases impossible. This, along with other incentives, has led to the emergence of systems that employ the *declarative programming paradigm*, to specify the problem at a higher level and use these propositional solvers to compose an answer for it.

The declarative nature of these languages implies that instead of describing the steps required to find the answer, the programmer, in a rigorous manner, states what the answer should look like. A declarative programming language for search problems provides a syntax to describe the relationship between the problem instance and the solution.

Fagin's theorem [12] states that the set of all properties expressible in existential second-order logic ($\exists$SO) is precisely the complexity class NP. The theorem suggests a natural declarative problem solving approach for NP-complete problems: Represent a problem with an $\exists$SO formula φ, and solve instances by reduction to SAT or some other fixed NP-complete problem. In the case of search problems, we must find interpretations of the existentially quantified second order variables, which provide a solution. So, the task becomes that of expanding a given structure to give suitable interpretations for those relation symbols.

As an example, consider the classical NP-complete problem of graph 3-colouring. The instance graph is a finite structure and the problem specification is described by the following first-order (FO)

formula (Note that the ∃SO quantifiers are omitted from the specification):

$$\begin{aligned} &\forall x(R(x) \vee B(x) \vee G(x)) \ \wedge \\ &\forall x[\neg(R(x) \wedge B(x)) \wedge \neg(R(x) \wedge G(x)) \wedge \neg(B(x) \wedge G(x))] \ \wedge \\ &\forall x \forall y[E(x,y) \supset (\neg(R(x) \wedge R(y)) \wedge \neg(B(x) \wedge B(y)) \wedge \neg(G(x) \wedge G(y)))]. \end{aligned} \tag{1.1}$$

Now the task is to expand the instance to containt an interpretation for the $\{R, B, G\}$ relations, which correspond to a possible colouring of the instance graph. For a specific logic $\mathcal{L}$, the task is called $\mathcal{L}$-Model Expansion, abbreviated $\mathcal{L}$-MX. In this thesis our focus is on specifications in the form of first-order formulas, so the model expansion task would be FO-MX.

Many systems have specifications given in extensions or restrictions of classical FO, including: IDP [**?**], MXG [25], Enfragmo [6, 7], ASPPS [11], and Kodkod [28]. Specifications for ASP systems, such as DLV [20] and clingo [13], are (extended) normal logic programs under stable model semantics. The focus of this thesis is on Enfragmo which was built in our team at SFU.

Conversion to SAT, or *grounding*, is central in many of these systems. Given the problem specification φ and the problem instance $\mathcal{A}$ (see Figure 1.1), the grounder, roughly, must produce a ground formula ψ which is logically equivalent to φ over the domain of $\mathcal{A}$. Then ψ can be transformed into a propositional CNF formula, and given as input to a SAT solver. If a satisfying assignment is found, a solution to $\mathcal{A}$ can be constructed from it in terms of a new structure $\mathcal{B}$. ASP systems use an analogous process.

A "naive" grounding of φ over a finite domain A can be obtained by replacing each sub-formula of the form $\exists x\, \psi(x)$ with $\bigvee_{a \in A} \psi(\tilde{a})$, where $\tilde{a}$ is a constant symbol which denotes domain element a, and similarly replacing each subformula $\forall x\, \psi(x)$ with a conjunction. For a fixed FO formula φ, this can be done in time polynomial in $|A|$. Most grounders use refinements of this method, implemented top-down or bottom-up, and perform well on simple benchmark problems and small instances. However, as we tackle more realistic problems with complex specifications and instances

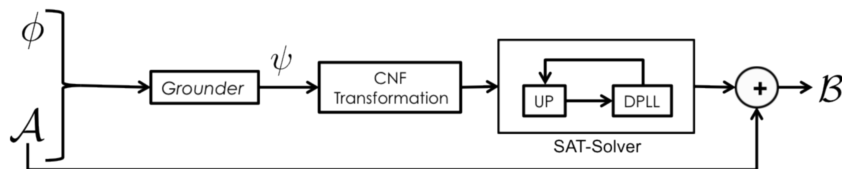

Figure 1.1: Overview of a typical ground-based system (like Enfragmo).

having large domains, the produced groundings can become prohibitively large. This can be the case even when the formulas are "not too hard". That is, the system performance is poor because of time spent generating and manipulating this large ground formula, yet an essentially equivalent but smaller formula can be solved in reasonable time. In this thesis we present one direction to develop techniques which scale effectively to complex specifications and large instances.

Most efficient SAT solvers employ a version of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm which begins by executing unit propagation (UP) on the input formula (note that other pre-processings might be done by SAT solvers but that is not a part of the original DPLL algorithm). This initial application of UP often eliminates a large number of variables and clauses, and is done very fast. However, it may be too late: the system has already spent a good deal of time generating large but rather uninteresting (parts of) ground formulas, transforming them to CNF, moving them from the grounder to the SAT solver, building the SAT solver's data structures, etc. This suggests trying to execute a process similar to UP *before or during grounding*.

One version of this idea was introduced in [34, 36]. The method presented there involves computing a *symbolic* and *incomplete* representation of the information that UP could derive, obtained from φ alone without reference to a particular instance structure. For brevity, we refer to that method as GWB, for "Grounding with Bounds". In [34, 36], the top-down grounder GIDL [33] is modified to use this information, and experiments indicate it significantly reduces the size of groundings without taking unreasonable time.

In [31] we proposed an alternative approach to this method which involves constructing a *concrete* and *complete* representation of the information that UP can derive about a grounding of φ over $\mathcal{A}$, and then using this information during grounding to reduce grounding size. This thesis elaborates more on that method, which is called *lifted unit propagation* (LUP).

The LUP method is roughly as follows.

1. Modify instance structure $\mathcal{A}$ to produce a new (partial) structure which contains information equivalent to that derived by executing UP on the CNF formula obtained from a grounding of φ over $\mathcal{A}$. We call this new partial structure the LUP structure for φ and $\mathcal{A}$, denoted $\mathcal{LUP}(\varphi,\mathcal{A})$.

2. Run a modified (top-down or bottom-up) grounding algorithm which takes as input φ and $\mathcal{LUP}(\varphi,\mathcal{A})$, and produces a grounding of φ over $\mathcal{A}$.

The modification in step 2 relies on the idea that a tuple in $\mathcal{LUP}(\varphi,\mathcal{A})$ indicates that a particular subformula has the same (known) truth value in every model. Thus, that subformula may be replaced

with its truth value. So the intuition is that the CNF formula obtained by grounding over $\mathcal{LUP}(\varphi,\mathcal{A})$ is at most as large as the formula that results from producing the naive grounding and then executing UP on it. Our experiments not only confirm that claim, but also show that in some cases the size of CNF formula is even much smaller than this, because the grounding method eliminates some "autark sub-formulas" which UP does not eliminate, as explained in Section 3.4.1 and Chapter 5.

We compute the LUP structure by constructing, from φ, an inductive definition of the relations of the LUP structure for φ and $\mathcal{A}$ (see Section 4.3.1). A semi-naive method for evaluating this inductive definition, based on relational algebra, has been implemented that works within the grounder Enfragmo. We also computed the definitions using the ASP grounders Gringo and DLV, but these were not faster.

The computed LUP structure can be used during grounding, with small modifications to the grounding algorithms. For top-down grounding (see Section 4.2), we modify the naive recursive algorithm to check the derived information in $\mathcal{LUP}(\varphi, \mathcal{A})$ at the time of instantiating each sub-formula of φ.

For bottom-up grounding (see Section 4.3.3), we revise the bottom-up grounding method based on the extended relational algebra described in [23, 27], which is the basis of grounders our group has been developing. The change required to ground using $\mathcal{LUP}(\varphi, \mathcal{A})$ is a simple revision to the base case.

1.1 Structure of the Text

What follows is the structure of the thesis:

- In Chapter 2, we provide some background material on classical logic, partial structures and theory of Datalog. These are theoretical foundations on which the following chapters are established. At the end of the chapter we provide some complexity background regarding the NP search problems.

- In Chapter 3, we first formally define the notion of Model Expansion and then we give two grounding algorithms that can be used for MX, namely: top-down grounding, and relational algebraic grounding, where the later is an example of bottom-up grounding. Finally, we present methods for CNF transformation and formally define unit propagation and Autarkies and we designate the notations that are going to be used in later chapters to refer to these notions.

- We begin Chapter 4 by introducing a special kind of partial structures, bound structures, and we define the notion of grounding over such structures. In Section 4.2 we provide a top-down grounding algorithm capable of grounding over bound structures and we use that algorithm as a theoretical touchstone for the actual algorithm that we have used in practice. In Section 4.3 we introduce LUP structures as a special type of bound structure and we prove that grounding over such structure results in ground formulas that are at most as large as the result of conventional grounding algorithm followed by unit propagation. Later, in section 4.3.1, we use Datalog programs to construct LUP structures and we provide an algorithm to evaluate those programs. Finally in Section 4.3.3, we present a modification to the bottom-up algorithm of Section 3.3 so that it can ground over LUP structures.

- In Chapter 5 we present an experimental evaluation of the performance of our grounder Enfragmo with LUP. This evaluation is limited by the fact that our LUP implementation does not support specifications with arithmetic or aggregates, and a shortage of interesting benchmarks which have natural specifications without these features. Within the limited domains we have tested, we found that:

  1. CNF formulas produced by Enfragmo with LUP are always smaller than the result of running UP on the CNF formula produced by Enfragmo without LUP, and in some cases much smaller.
  2. CNF formulas produced by Enfragmo with LUP are always smaller than the ground formulas produced by GIDL, with or without GWB turned on.
  3. Grounding over $\mathcal{LUP}(\varphi, \mathcal{A})$ is always slower than grounding without, but CNF transformation with LUP is almost always faster than without.
  4. Total solving time for Enfragmo with LUP is sometimes significantly less than that of Enfragmo without LUP, but in other cases is somewhat greater.
  5. Enfragmo with LUP and the SAT solver MiniSat always runs faster than the IDP system (GIDL with ground solver MINISAT(ID)), with or without the GWB method turned on in GIDL .

- Determining the extent to which these observations generalize is future work which is discussed in Chapter 6.

Chapter 2

Preliminaries

In this chapter we review some of the background material from mathematical logic and introduce the notation that will be used throughout this thesis. First we introduce classical first-order logic (FO), then we introduce the notion of partial structures. After that, we introduce Datalog, which is a particular logic programming language. Finally, we present some standard material on SAT solving and conversion to SAT.

2.1 First-order Logic

Definition 2.1.1. A *vocabulary* is composed of a set of *constant* symbols, like $\{c_1, c_2, \ldots, c_n, \ldots\}$, *predicate* symbols, like $\{P_1, P_2, \ldots, P_n, \ldots\}$, and *functions* symbols, like $\{f_1, f_2, \ldots, f_n, \ldots\}$, where each predicate and function symbol has a specific *arity*, which is the number of its arguments.

We use P/n to emphasise that the arity of predicate symbol P is n. Note that the constant symbols are in fact 0-ary function symbols.

Throughout the text we use capital letters to refer to predicate symbols and lower case letters to refer to function symbols.

Definition 2.1.2 (Structure)**.** If σ is a vocabulary, a σ-structure $\mathcal{A} = \langle A, \{c_i^{\mathcal{A}}\}, \{P_i^{\mathcal{A}}\}, \{f_i^{\mathcal{A}}\}\rangle$ consists of a *domain* A, along with an interpretation for constant, predicate and function symbols, as follows:

- each constant symbol c_i of σ is mapped to an element $c_i^{\mathcal{A}}$ from the domain A,

- each k-ary predicate symbol P_i of σ is mapped to a k-ary relation $P_i^{\mathcal{A}}$ over A, i.e. $P_i^{\mathcal{A}} \subseteq A^k$,
- each k-ary function symbol f_i of σ is mapped to a k-ary function $f_i^{\mathcal{A}}$ over A, i.e. $f_i^{\mathcal{A}} : A^k \rightarrow A$.

We use upper case script letters (e.g. $\mathcal{A}$, $\mathcal{B}$) for structures, and the domain of a structure is denoted by a Roman letter corresponding to the name of the structure, for example, domains of structures $\mathcal{A}$ and $\mathcal{B}$ are denoted by A and B, respectively. Throughout the thesis, all structures are finite (a structure is finite if its associated domain is finite).

Example 2.1.1. Consider vocabulary σ containing two constant symbols s and d, a binary relation E and a binary function symbol w. Then a flow graph with weighted edges, $\mathcal{G} = \langle V, s^{\mathcal{G}}, d^{\mathcal{G}}, E^{\mathcal{G}}, w^{\mathcal{G}} \rangle$ with source node $s^{\mathcal{G}}$ and destination $d^{\mathcal{G}}$ is a structure for σ, where V is the set of vertices, and $E^{\mathcal{G}}$ is the edge relation.

Typically when we are describing structures, we drop the superscript, so the graph structure of Example 2.1.1 would be denoted by $\mathcal{G} = \langle V, s, d, E, w \rangle$.

2.1.1 Syntax

We inductively define terms and formulas of the first-order logic over vocabulary σ as follows. We assume an infinite set of variables:

- Each variable x is a term.
- Each constant symbol c is a term.
- If f is a k-ary function symbol and $t_1, \ldots, t_k$ are terms, then $f(t_1, \ldots, t_k)$ is also a term.
- if t_1 and t_2 are terms, then $t_1 = t_2$ is an atomic formula.
- If P is a k-ary predicate symbol and $t_1, \ldots, t_k$ are terms, then $P(t_1, \ldots, t_k)$ is an atomic formula.
- If ψ_1 and ψ_2 are two formulas, then $\neg\psi_1$, $\psi_1 \wedge \psi_2$ and $\psi_1 \vee \psi_2$ are formulas.
- If ψ is a formula, then $\forall x \psi$ and $\exists x \psi$ are formulas.

Symbols $\wedge$, $\vee$ and $\neg$ are called *connectives* and $\forall$ and $\exists$ are called *quantifiers*. When a formula does not have any connectives or quantifiers we call it an *atom*. A *literal* is an atom (positive literal) or the negation of an atom (negative literal).

Definition 2.1.3 (Bound Variable)**.** A variable occurrence x is *bound* if it occurs in a sub-formula of the form $\forall x\varphi$ or $\exists x\varphi$ and *free* otherwise.

A formula is called a *sentence* if it does not have any free variables. We denote the set of free variables $\bar{x}$ of formula φ by $\varphi(\bar{x})$, where $\bar{x}$ is a tuple of variables. We also write $\bar{c}$) to refer to tuples of constant symbols. We sometimes use $\forall\bar{x}\varphi$ and $\exists\bar{x}\varphi$ as shorthands for formulas $\forall x_1, \ldots, \forall x_n\varphi$ and $\exists x_1, \ldots, \exists x_n\varphi$ respectively, if $\bar{x}$ is a tuple of variables $x_1, \ldots, x_n$. A formula that does not have any quantifiers is call it a *quantifier-free formula*.

We use $\varphi \supset \psi$, $\varphi \leftrightarrow \psi$ and $t_1 \neq t_2$ as abbreviations for formulas $\neg\varphi \vee \psi$, $(\varphi \supset \psi) \wedge (\psi \supset \varphi)$ and $\neg(t_1 = t_2)$, respectively.

A *substitution* is a set of pairs (t/t'), where t and t' are terms. If θ is a substitution, then $\varphi[\theta]$ denotes the result of substituting t' for each occurrence of t in φ, for every (t/t') in θ. We assume that all of the variables of t' are free in $\varphi[t/t']$.

2.1.2 Semantics

The value of a first-order formula, with respect to structure $\mathcal{A}$, can be determined in an inductive manner based on the value of its constituting terms:

The value of a k-ary term t with free variables $\bar{x}$ under structure $\mathcal{A}$ at $\bar{a} \in A^n$ is denoted by $t^{\mathcal{A}}(\bar{a})$ and is defined as follows [1]:

- if t is a constant symbol c, then $t^{\mathcal{A}}$ is $c^{\mathcal{A}}$.
- if t is a variable x_i, then the value of $t^{\mathcal{A}}(\bar{a})$ is a_i.
- if t is of the form $f(t_1, \ldots, t_k)$, where f is a k-ary function symbol, then the value of $t^{\mathcal{A}}(\bar{a})$, in which $\bar{a} \in A^k$, is $f^{\mathcal{A}}(t_1^{\mathcal{A}}(\bar{a}), \ldots, t_k^{\mathcal{A}}(\bar{a}))$.

The fact that subformula $\varphi(\bar{x})$ is true according to structure $\mathcal{A}$ at point $\bar{a}$ is denoted by $\mathcal{A} \models \varphi(\bar{x})[\bar{x}/\bar{a}]$ and is determined as follows:

- if φ is $(t_1 = t_2)$, then $\mathcal{A} \models \varphi(\bar{x})[\bar{x}/\bar{a}]$ iff $t_1^{\mathcal{A}}(\bar{a}) = t_2^{\mathcal{A}}(\bar{a})$.
- if φ is $P(t_1, \ldots, t_k)$, then $\mathcal{A} \models \varphi(\bar{x})[\bar{x}/\bar{a}]$ iff $(t_1^{\mathcal{A}}(\bar{a}), \ldots, t_k^{\mathcal{A}}(\bar{a})) \in P^{\mathcal{A}}$.

[1]Note that we assume an ordering on the set of variables, and tuples of domain elements evaluate those variables based on that ordering.

- if φ is $\neg\psi$, then $\mathcal{A} \models \varphi$ iff $\mathcal{A} \not\models \psi$.
- if φ is $(\psi_1 \wedge \psi_2)$, then $\mathcal{A} \models \varphi$ iff $\mathcal{A} \models \psi_1$ and $\mathcal{A} \models \psi_2$.
- if φ is $(\psi_1 \vee \psi_2)$, then $\mathcal{A} \models \varphi$ iff $\mathcal{A} \models \psi_1$ or $\mathcal{A} \models \psi_2$.
- if $\varphi(\bar{x})$ is $\forall y \psi(\bar{x}, y)$, then $\mathcal{A} \models \varphi(\bar{x})[\bar{x}/\bar{a}]$ iff $\mathcal{A} \models \psi(\bar{x}, y)[(\bar{x}/\bar{a}) \cup (y/a')]$ for every $a' \in A$.
- if $\varphi(\bar{x})$ is $\exists y \psi(\bar{x}, y)$, then $\mathcal{A} \models \varphi(\bar{x})[\bar{x}/\bar{a}]$ iff $\mathcal{A} \models \psi(\bar{x}, y)[(\bar{x}/\bar{a}) \cup (y/a')]$ for some $a' \in A$.

Let $\tau = \mathrm{vocab}(\varphi)$, where $\mathrm{vocab}(\varphi)$ is the vocabulary under which sentence φ is defined. A sentence φ is ***satisfiable*** iff there is a τ-structure $\mathcal{A}$ in which φ is true. In that case, $\mathcal{A}$ is a *model* for φ. A formula $\varphi(\bar{x})$ is satisfiable iff the sentence $\exists \bar{x} \varphi(\bar{x})$ is satisfiable. A sentence φ is *valid* iff it is true in every τ-structure. Validity of sentence φ is denoted by: $\models \varphi$.

Examples of valid and unsatisfiable formulas are $\forall x(E(x) \vee \neg E(x))$ and $\exists x(E(x) \wedge \neg E(x))$, respectively. We respectively use the notations $\top$ and $\bot$ to denote a valid and unsatisfiable formula.

2.1.3 Equivalence of Formulas and Negation Normal Form

Sentences φ_1 and φ_2 are *logically equivalent* iff $\varphi_1 \leftrightarrow \varphi_2$ is valid. We use "$\Leftrightarrow$" to indicate that two formulas are logically equivalent. It is immediate from this definition that if two formulas are equivalent, then they have identical sets of models.

The correctness of the following equivalences can be determined from the FO semantics:

1. Distributivity Laws:

$$\varphi \wedge (\psi \vee \mu) \Leftrightarrow (\varphi \wedge \psi) \vee (\varphi \wedge \mu) \tag{2.1}$$

$$\varphi \vee (\psi \wedge \mu) \Leftrightarrow (\varphi \vee \psi) \wedge (\varphi \vee \mu) \tag{2.2}$$

2. Negation:

$$\neg\neg\varphi \Leftrightarrow \varphi \tag{2.3}$$

$$\neg(\varphi \vee \psi) \Leftrightarrow \neg\varphi \wedge \neg\psi \tag{2.4}$$

$$\neg(\varphi \wedge \psi) \Leftrightarrow \neg\varphi \vee \neg\psi \tag{2.5}$$

$$\neg\forall\varphi \Leftrightarrow \exists\neg\varphi \tag{2.6}$$

$$\neg\exists\varphi \Leftrightarrow \forall\neg\varphi \tag{2.7}$$

3. Flattening Terms (where φ is an atomic formula):

$$\varphi \Leftrightarrow \exists x\ (x = t \wedge \varphi[t/x]) \tag{2.8}$$

$$\varphi \Leftrightarrow \forall x\ (x = t \supset \varphi[t/x]) \tag{2.9}$$

Often systems witch have their input expressed in first order logic, transform the input into an equivalent formula that tends to be syntactically more manageable. In later chapters we require our formula to be in *Negation Normal Form (NNF)*, meaning that the only negated subformula are atoms. Every τ-sentence can be transformed to an equivalent τ-sentence in NNF by repeated use of equivalences 2.3 to 2.7.

In addition, throughout the thesis, we always assume that terms containing $n(>0)$-ary function symbols do not appear as arguments for predicate symbols, except for equality. For example, the formula $\forall x, \bar{y}\ P(x, f(\bar{y}))$ is not acceptable, due to existence of function symbol $f(\bar{y})$ inside predicate symbol P. If a formula does not have any such predicates, we say that it is in *Term Normal Form*. In fact any formula can be transformed to an equivalent formula in TNF, by the application of equivalences 2.8 and 2.9. In case of the above formula, if we apply equivalence 2.8, we obtain $\forall x, y\ \exists z(z = f(\bar{y}) \wedge P(x, z))$ in TNF.

Graph of a Function

Another assumption that we consider about vocabularies in the later chapters of this thesis is for them to be function-free. This assumption does not restrict our domain of discourse, since one can replace functions with their *graphs* and add some *safety sentences* to the formula. The graph of a function like $f(x)$ is a binary relation $F(x, y)$, where y is the value of f at x. For instance if f is the square function, then $F(2, 4)$ is true while $F(2, 5)$ is false.

The safety sentences are added to reflect the essence of function symbols and their totality, over to the predicate symbols that are modelling them. In other words, the fact that the result of a function at some point is a *unique* element of the domain and also the fact that a function is defined over all domain elements, should be hard coded into the formula, by means of the safety sentences:
For every function symbol $f(\bar{x})$ in the vocabulary, we add the following sentences about it's graph to our formula:

$$\forall \bar{x}, y_1, y_2\ (F(\bar{x}, y_1) \wedge F(\bar{x}, y_2) \supset y_1 = y_2) \tag{2.10}$$

$$\forall \bar{x}\, \exists y\, F(\bar{x}, y) \tag{2.11}$$

Example 2.1.2. Consider the formula $\forall x, \bar{y}\, P(x, f(\bar{y}))$ that we had before. After bringing the function out of the predicate symbol we have $\forall x, \bar{y}\, \exists z(z = f(\bar{y}) \wedge P(x, z))$. At this point the subformula $z = f(\bar{y})$ can be replaced by the graph of f, provided that we add the safety sentences. The final formula would be:

$$\begin{aligned}
&\forall x, \bar{y}\, (\exists z P(x, z) \wedge F(\bar{y}, z)) \wedge \\
&\forall \bar{y}, z_1, z_2\, (F(\bar{y}, z_1) \wedge F(\bar{y}, z_2) \supset z_1 = z_2) \wedge \\
&\forall \bar{y}\, \exists z\, F(\bar{y}, z).
\end{aligned}$$

A vocabulary that does not contain any function symbols is called a *relational vocabulary* and a structure of such vocabulary is called a *relational structure*.

2.1.4 Partial Structures

A relational τ-structure $\mathcal{A}$ consists of a domain A together with a relation $R^{\mathcal{A}} \subseteq A^k$ for each k-ary relation symbol of τ. We will make use of partial structures, in which the interpretation of a relation symbol may be only partially defined. For this, it is convenient to view a structure in terms of the characteristic functions of the relations. Partial τ-structure $\mathcal{A}$ consists of a domain A together with a k-ary function $\chi_R^{\mathcal{A}} : A^k \to \{\top, \bot, \infty\}$, for each k-ary relation symbol R of τ. Here and throughout the text, $\top$ denotes true, $\bot$ denotes false, and ∞ denotes undefined. If each of these characteristic functions is total, then $\mathcal{A}$ is total. We may sometimes abuse terminology and call a relation partial, meaning the characteristic function interpreting the relation symbol in question is partial.

For any (total) τ-structure $\mathcal{A}$, each τ-sentence φ is either true or false in $\mathcal{A}$ ($\mathcal{A} \models \varphi$ or $\mathcal{A} \not\models \varphi$), and each τ-formula $\varphi(\bar{x})$ with free variables $\bar{x}$, defines a relation

$$\varphi^{\mathcal{A}} = \{\bar{a} \in A^{|\bar{x}|} : \mathcal{A} \models \varphi(\bar{x})[\bar{x}/\bar{a}]\}. \tag{2.12}$$

Similarly, for any partial τ-structure, each τ-sentence is either true, false or undetermined in $\mathcal{A}$, and each τ-formula $\varphi(\bar{x})$ with free variables $\bar{x}$ defines a partial function

$$\chi_\varphi^{\mathcal{A}} : A^k \to \{\top, \bot, \infty\}. \tag{2.13}$$

In the case that $\chi_\varphi^{\mathcal{A}}$ is total, it is the characteristic function of the relation (2.12).

Figure 2.1: Ordering on truth values for partial structures.

Semantics of Partial Structures

To grasp the above concept of evaluating the sentences and formulas under partial structures, we provide the natural adaptation of standard FO semantics to the case of partial relations. Our semantics adhere to Kleene's 3-valued semantics [19].

Considering the lattice of Figure 2.1, the value of subformula $\varphi(\bar{x})$ according to structure $\mathcal{A}$ at point $\bar{a}$ is determined as follows:

- if φ is $(t_1 = t_2)$, then $\chi_\varphi^\mathcal{A}(\bar{a}) = \begin{cases} \top & \text{if } t_1^\mathcal{A}(\bar{a}) = t_2^\mathcal{A}(\bar{a}) \\ \bot & \text{if } t_1^\mathcal{A}(\bar{a}) \neq t_2^\mathcal{A}(\bar{a}). \end{cases}$
- if φ is $P(t_1, \ldots, t_k)$, then $\chi_\varphi^\mathcal{A} = \chi_P^\mathcal{A}$.
- if φ is $\neg\psi$, then $\chi_\varphi^\mathcal{A}(\bar{a}) = \neg\chi_\psi^\mathcal{A}(\bar{a})$.
- if φ is $(\psi_1 \wedge \psi_2)$, then $\chi_\varphi^\mathcal{A}(\bar{a}) = \mathrm{Min}\{\chi_{\psi_1}^\mathcal{A}(\bar{a}), \chi_{\psi_2}^\mathcal{A}(\bar{a})\}$.
- if φ is $(\psi_1 \vee \psi_2)$, then $\chi_\varphi^\mathcal{A}(\bar{a}) = \mathrm{Max}\{\chi_{\psi_1}^\mathcal{A}(\bar{a}), \chi_{\psi_2}^\mathcal{A}(\bar{a})\}$.
- if $\varphi(\bar{x})$ is $\forall y \psi(\bar{x}, y)$, then $\chi_\varphi^\mathcal{A}(\bar{a}) = \prod_{a' \in A} \chi_\psi^\mathcal{A}(\bar{a}, a')$.
- if $\varphi(\bar{x})$ is $\exists y \psi(\bar{x}, y)$, then $\chi_\varphi^\mathcal{A}(\bar{a}) = \sum_{a' \in A} \chi_\psi^\mathcal{A}(\bar{a}, a')$.

Where Max and Min are respectively the *greatest lower bound* and *least upper bound* operators . We used the curly dot notations to distinguish them from regular $\wedge$ and $\vee$.

Ordering on Partial Structures

There is a natural partial order on partial structures for any vocabulary τ, which we may denote by $\leq$, where $\mathcal{A} \leq \mathcal{B}$ iff $\mathcal{A}$ and $\mathcal{B}$ agree at all points where they are both defined, and $\mathcal{B}$ is defined at every point $\mathcal{A}$ is. If $\mathcal{A} \leq \mathcal{B}$, we may say that $\mathcal{B}$ is a *strengthening of* $\mathcal{A}$. When convenient, if the vocabulary of $\mathcal{A}$ is a proper subset of that of $\mathcal{B}$, we may still call $\mathcal{B}$ a strengthening of $\mathcal{A}$, taking $\mathcal{A}$ to leave all symbols not in its vocabulary, completely undefined. We will call $\mathcal{B}$ a *conservative strengthening of* $\mathcal{A}$ *with respect to formula* φ if $\mathcal{B}$ is a strengthening of $\mathcal{A}$ and in addition every total structure which is a strengthening of $\mathcal{A}$ and a model of φ is also a strengthening of $\mathcal{B}$.

Example 2.1.3. Let φ be the sentence:

$$\forall x, y, z\ (Father(x, y) \wedge Father(y, z)) \supset GParent(x, z)$$

over vocabulary $\mathcal{P} = \{Father/2, GParent/2\}$ and consider partial $\mathcal{P}$-structures $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, all defined over domain $\{Lohraspa, Vishtaspa, Peshotanu, Spentodata\}$, as follows:

$$\chi^{\mathcal{A}}_{Father}(x, y) = \chi^{\mathcal{B}}_{Father}(x, y) = \chi^{\mathcal{C}}_{Father}(x, y) = \begin{cases} \top & (x, y) \in \{(Lohraspa, Vishtaspa), \\ & \qquad (Vishtaspa, Peshotanu)\} \\ \bot & \text{o.w.} \end{cases}$$

$$\chi^{\mathcal{A}}_{GParent}(x, y) = \infty$$

$$\chi^{\mathcal{B}}_{GParent}(x, y) = \begin{cases} \top & (x, y) \in \{(Lohraspa, Peshotanu)\} \\ \infty & \text{o.w.} \end{cases}$$

$$\chi^{\mathcal{C}}_{GParent}(x, y) = \begin{cases} \top & (x, y) \in \{(Lohraspa, Spentodata)\} \\ \infty & \text{o.w.} \end{cases}$$

Both $\mathcal{B}$ and $\mathcal{C}$ are strengthenings of $\mathcal{A}$, but only $\mathcal{B}$ is a conservative strengthening of $\mathcal{A}$ w.r.t. φ, since $GParent(Lohraspa, Peshotanu)$ is true in every model of φ that is a strengthening of $\mathcal{A}$, while there is a strengthening of $\mathcal{A}$ that is a model of φ in which $GParent(Lohraspa, Spentodata)$ is false.

2.2 Datalog

Later in the thesis we perform a computation that can be conviniently described in the *logic programming paradigm*. In fact the programs that we deal with are in a very restricted logic programming language that is equivalent to the *deductive database* query language *Datalog* [9, 8]. Datalog has emerged from the integration of logic programming and databases and it is used to infer additional facts about the data stored in a database.

2.2.1 Syntax

A Datalog program Δ consists of a set of *facts* and a set of *rules*, denoted respectively by $\mathrm{Fact}(\Delta)$ and $\mathrm{Rule}(\Delta)$. The facts are a set of assertions about the entities under consideration, for example "Lohraspa is the father of Vishtaspa". The rules impose some relations on these entities and allow for further deduction of facts from other facts. For example "If X is the father of Y and Y is the father of Z, then X is the grandparent of Z" constitutes a rule. Obviously, $\mathrm{Fact}(\Delta) \cup \mathrm{Rule}(\Delta) = \Delta$. The set of constant symbols occurring in a Datalog program Δ is denoted by $adom(\Delta)$.

Rules and facts of Datalog are given as *Horn clauses*, of the form:

$$L_0(\bar{x}_0) \;\leftarrow\; L_1(\bar{x}_1), \ldots, L_n(\bar{x}_n) \tag{2.14}$$

where $\bar{x}_i$ are tuples of variables. The left-hand side is called the *head* and the right-hand side is called the *body*. The facts can be thought of as rules with empty body. For instance the aforementioned fact and rule about fathers and grandparents can be represented in Datalog as:

$$Father(Lohraspa, Vishtaspa).$$
$$GParent(x, z) \;\leftarrow\; Father(x, y), Father(y, z).$$

Where x, y and z are variables. An atom, fact or rule that contains no variables is called *ground*. Any Datalog program Δ must satisfy the following *safety conditions*:

- Each fact of Δ is ground.
- Every variable appearing in the head of a rule r of Δ must also appear in the body of the r.

2.2.2 Semantics

A Datalog program Δ can be associated to a set of FO formulas: Every fact is a ground predicate and every rule like 2.14 can be converted to a FO formula like:

$$\forall \bar{x}_1, \ldots, \bar{x}_n (L_1(\bar{x}_1) \wedge \ldots \wedge L_n(\bar{x}_n) \supset L_0(\bar{x}_0)) \tag{2.15}$$

The vocabulary of Δ is denoted by $\mathrm{vocab}(\Delta)$. A ground fact $L(\bar{a})$ is satisfied under a $\mathrm{vocab}(\Delta)$-structure $\mathscr{A}$ iff $\bar{a} \in L^{\mathcal{A}}$. Note that in the Datalog literature, a structure is regarded as a set of facts, that's why we used a different notation to emphasize this distinction. One can turn a Datalog structure $\mathscr{A}$ to a first order structure $\mathcal{A}$ by setting the value of a relation $P^{\mathcal{A}}$ true at $\bar{a}$, if $P(\bar{a})$ is a fact in $\mathscr{A}$ and setting it to false otherwise. A Datalog rule r is true under $\mathscr{A}$ iff its corresponding FO formula is satisfied by $\mathscr{A}$ for every evaluation of the free variables of r. Structure $\mathscr{A}$ is a model for Δ (denoted by $\mathscr{A} \models \Delta$) iff every fact and rule of Δ is true under $\mathscr{A}$.

We denote by $\mathrm{HB}(\Delta)$ the set of all possible atoms that we can express in the language of Datalog program Δ, i.e. the set of all atoms, $L(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n$ are constant symbols in $adom(\Delta)$. A structure, then, is a subset of this set, containing the facts that are true under itself.

A fact $L(\bar{a}) \in \mathrm{HB}(\Delta)$ is a logical consequence of (or *follows from*) Δ iff every structure $\mathscr{A}$ that satisfies Δ also satisfies $L(\bar{a})$. If $L(\bar{a})$ follows from Δ, we write $\Delta \models L(\bar{a})$. Intuitively, facts that follow from a Datalog program are the answers obtained by running that program.

Definition 2.2.1 ($\mathrm{cons}(\Delta)$)**.** The set of all logical consequence facts of a datalog program Δ is denoted by $\mathrm{cons}(\Delta)$, which is equal to the intersection of all models of Δ:

$$\mathrm{cons}(\Delta) = \{L(\bar{a}) \in \mathrm{HB}(\Delta) \mid \Delta \models L(\bar{a})\} \tag{2.16}$$

$$= \cap \{\mathscr{A} \mid \mathscr{A} \models \Delta\}. \tag{2.17}$$

The set $\mathrm{cons}(\Delta)$ is called the *minimal model* of Δ.

Fixpoint Characterization of $\mathrm{cons}(\Delta)$

There is a fixpoint characterization for $\mathrm{cons}(\Delta)$ which, to a good extent, mimics the actual implementation of this operator in practice (see Chapter 4). Let $\mathrm{Infer}_1(\Delta)$ be the set of facts that can be inferred from the set of Datalog clauses Δ in a *single* step. It means that the result of this function is the set of heads of rules in Δ whose bodies are satisfied by only the facts in Δ. Then $\mathrm{cons}(\Delta)$

can be characterized by the least fixpoint of a mapping $\mathrm{T}_\Delta : 2^{\mathrm{HB}(\Delta)} \rightarrow 2^{\mathrm{HB}(\Delta)}$. The mapping is defined as:

$$\mathrm{T}_\Delta(\mathcal{A}) = \mathcal{A} \cup \mathrm{Fact}(\Delta) \ \cup \ \mathrm{Infer}_1(\mathrm{Rule}(\Delta) \cup \mathcal{A}). \tag{2.18}$$

where $\mathcal{A}$ can be viewed both as a set of facts and a structure.

The set of models of Δ coincides with the set of fixpoints of T_Δ. It is easy to see that the least fixpoint of T_Δ is the same as the minimal model of Δ. So $\mathrm{cons}(\Delta)$ can be computed by a least fixpoint operation on T_Δ, i.e. computing $\mathrm{T}_\Delta(\emptyset)$, $\mathrm{T}_\Delta(\mathrm{T}_\Delta(\emptyset))$, $\mathrm{T}_\Delta(\mathrm{T}_\Delta(\mathrm{T}_\Delta(\emptyset)))$, ... until the result of an application of T_Δ is equal to its predecessor.

2.3 Some Notes on Computational Complexity

Throughout the thesis we focus on a class of problems known as *NP search problems*. In this section we formally define this complexity class.

The *input* of a computational problem can be regarded as a string over a set of elements, called the *domain*, and the *output* should have a certain property. Usually the problem is described in terms of that property. Formula 1 of the Introduction section, for instance, gives the property of interest for the graph 3-colouring problem.

Among different types of computational problems are *decision problems* and *search problems* [29]:

The output of a decision problem, given the input, is just a *Yes/No* answer. In other words we are only interested to see if the input satisfies the property. For instance, again for the graph 3-colouring problem, given an undirected graph $\mathcal{G} = \{V, E\}$, we are asked to verify wether there is a way to assign colours (i.e. find the mapping $c : V \rightarrow \{R, B, G\}$), picked from the set $\{R, B, G\}$, to the vertices of $\mathcal{G}$ such that no two adjacent vertices are unicoloured.

For the search version, we have to come up with an answer that is in some relation with the input. More formally, given input x and a relation R, we want to find an "answer" y such that $(x, y) \in R$. This way the problem is even more challenging, because we not only are asked to determine wether an answer exists, we have to construct it also. For the graph 3-colouring problem, the mapping that assigns colours to vertices and adheres to the aforementioned criteria, itself, is the answer, if such a mapping exists.

2.3.1 NP Search Problems

If a decision problem asymptotically can be solved in polynomial time, we say that the problem lies in class P. Conventionally, we call the problems of this class to be solvable in a "feasible" time, where feasible refers to the rate of growth rather than the actual time needed to solve specific instances.

A decision problem L is a *NP decision problem* if there exists a polynomial time computable relation R and a constant c s.t. $x \in L$ iff there exists a y, $|y| \leq |x|^c$ such that $R(x, y)$ accepts.

A NP search problem can be though of as the search version of a decision problem, where for any given input x, an answer y (where $|y| \leq |x|^c$ for a constant c) can be checked in polytime.

Chapter 3

FO Model Expansion and Grounding

As mentioned in previous chapters, the combinatorial search problems targeted in this thesis all reside in the class of NP problems. Based on Cook's theorem [10] that any problem in NP can be reduced in polytime to SAT, one can devise a framework to solve NP problems by conversion to SAT and employment of efficient SAT-solvers as an off-the-shelf component of the framework.

A natural formalization of these NP search problems and their specifications is the logical task of *model expansion*. Several benefits can be achieved by formalizing in logic, one of which is access to the rich theoretical results in the field of *descriptive complexity* [17] where the correspondence between expressiveness of logics and computational complexity is established. This field provides tools to control the expressive power of the modelling language, which in turn calibrates the solver framework to target the problems at a specific complexity class.

For example Fagin's theorem [12] states such a relationship between ∃SO and complexity class NP. In [24], this theorem is rephrased to the statement that FO-MX can express every problem in NP. This result has two practical consequences: First, it provides assurance that all NP problems can be expressed in FO-MX; secondly, the statement that no problem outside of complexity class NP is expressible in FO-MX guarantees a polytime algorithm to convert (i.e. ground) the expressed problems to SAT.

In this chapter we first formally define the notion of MX and grounding. Then we provide algorithms to perform the grounding. The algorithms are categorized into *top-down* and *bottom-up* depending on where in the FO formula tree they start the conversion. Section 3.2 describes top-down grounding, while section 3.3 gives a bottom-up version based on *relational algebra*. Finally, in section 3.4 we discuss some technical issues regarding the transformation of the ground formula to *conjunctive normal form (CNF)*.

3.1 Formal Definition of Model Expansion and Grounding

Here, we define MX [22] for the special case of FO. A structure $\mathcal{B}$ for vocabulary $\sigma \cup \varepsilon$ is an expansion of σ-structure $\mathcal{A}$ iff $\mathcal{A}$ and $\mathcal{B}$ have the same domain ($A = B$), and interpret their common vocabulary identically, i.e., for each symbol R of σ, $R^{\mathcal{B}} = R^{\mathcal{A}}$. Also, if $\mathcal{B}$ is an expansion of σ-structure $\mathcal{A}$, then $\mathcal{A}$ is the reduct of $\mathcal{B}$ defined by σ. Note that expansion is a special case of strengthening. In fact, if $\mathcal{B}$ is a complete strengthening of $\mathcal{A}$, then it is an expansion for $\mathcal{A}$.

Definition 3.1.1 (Model Expansion for FO)**.**

Given: A FO formula φ on vocabulary $\sigma \cup \varepsilon$ and a σ-structure $\mathcal{A}$,

Find: an expansion $\mathcal{B}$ of $\mathcal{A}$ that satisfies φ.

In the present context, the formula φ constitutes a problem specification, the structure $\mathcal{A}$ a problem instance, and expansions of $\mathcal{A}$ which satisfy φ are solutions for $\mathcal{A}$. Thus, we call the vocabulary of $\mathcal{A}$, the *instance* vocabulary, denoted by σ, and ε the *expansion* vocabulary. We sometimes say φ is $\mathcal{A}$-satisfiable if there exists an expansion $\mathcal{B}$ of $\mathcal{A}$ that satisfies φ.

Example 3.1.1. Let φ be formula 1.1 of Chapter 1. A finite structure $\mathcal{A}$ over vocabulary $\sigma = \{E\}$, where E is a binary relation symbol, is a graph. Given graph $\mathcal{A} = \mathcal{G} = (V; E)$, there is an expansion $\mathcal{B}$ of $\mathcal{A}$ that satisfies φ, iff $\mathcal{G}$ is 3-colourable. So φ constitutes a specification of the problem of graph 3-colouring. To illustrate:

$$\underbrace{(\overbrace{V; E^{\mathcal{A}}}^{\mathcal{A}}, R^{\mathcal{B}}, B^{\mathcal{B}}, G^{\mathcal{B}})}_{\mathcal{B}} \models \varphi$$

An interpretation for the expansion vocabulary $\varepsilon := \{R, B, G\}$ given by structure $\mathcal{B}$ is a colouring of $\mathcal{G}$, and the proper 3-colourings of $\mathcal{G}$ are the interpretations of ε in structures $\mathcal{B}$ that satisfy φ.

Given φ and $\mathcal{A}$, we want to produce a CNF formula (for input to a SAT solver), which represents the solutions to $\mathcal{A}$. This is done in two steps: grounding, followed by transformation to CNF. The grounding step produces a ground formula ψ which is equivalent to φ over expansions of $\mathcal{A}$. To produce ψ, we bring domain elements into the syntax by expanding the vocabulary with a new constant symbol for each domain element. For A, the domain of $\mathcal{A}$, we denote this set of constants by $\tilde{A}$. For each $a \in A$, we write $\tilde{a}$ for the corresponding symbol in $\tilde{A}$. We also write $\tilde{\bar{a}}$, where $\bar{a}$ is a tuple.

Definition 3.1.2 (Grounding of φ over $\mathcal{A}$)**.** Let φ be a formula of vocabulary $\sigma \cup \varepsilon$, $\mathcal{A}$ be a finite σ-structure, and ψ be a ground formula of vocabulary μ, where $\mu \supseteq \sigma \cup \varepsilon \cup \tilde{A}$. Then ψ is a grounding of φ over $\mathcal{A}$ if and only if:

1. if φ is $\mathcal{A}$-satisfiable then ψ is $\mathcal{A}$-satisfiable;
2. if $\mathcal{B}$ is a μ-structure which is an expansion of $\mathcal{A}$ and gives $\tilde{A}$ the intended interpretation, and $\mathcal{B} \models \psi$, then $\mathcal{B} \models \varphi$.

We call ψ a reduced grounding if it contains no symbols of the instance vocabulary σ.

Definition 3.1.2 is a slight generalization of that used in [23, 27], in that it allows ψ to have vocabulary symbols not in $\sigma \cup \varepsilon \cup \tilde{A}$. This generalization allows us to apply a Tseitin-style CNF transformation (see Section 3.4) in such a way that the resulting CNF formula is still a grounding of φ over $\mathcal{A}$. If $\mathcal{B}$ is an expansion of $\mathcal{A}$ satisfying ψ, then the reduct of $\mathcal{B}$ defined by $\sigma \cup \varepsilon$ is an expansion of $\mathcal{A}$ that satisfies φ. For the remainder of the paper, we assume that φ is in NNF (see Section 2.1.3) and that $\supset$ and $\leftrightarrow$ symbols are replaced with their equivalences, as described in Section 2.1.3.

3.2 Top-Down Grounding

Algorithm 1 produces the "naive grounding" of φ over $\mathcal{A}$ mentioned in Chapter 1. We allow conjunction and disjunction to be connectives of arbitrary arity. That is $(\wedge\ \varphi_1\ \varphi_2 \ldots \varphi_i)$ is a formula, not just an abbreviation for some parenthesization of $(\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_i)$. The initial call to Algorithm 1 is $\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi, \emptyset)$, where $\emptyset$ is the empty substitution, after that the algorithm works in a top-down manner.

Algorithm 1 Top-Down Naive Grounding of NNF formula φ over $\mathcal{A}$

$$\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi,\theta) = \begin{cases} P(\bar{x})[\theta] & \text{if } \varphi \text{ is an atom } P(\bar{x}) \\ \neg P(\bar{x})[\theta] & \text{if } \varphi \text{ is a negated atom } \neg P(\bar{x}) \\ \bigwedge_i \mathrm{NaiveGnd}_{\mathcal{A}}(\psi_i,\theta) & \text{if } \varphi = \bigwedge_i \psi_i \\ \bigvee_i \mathrm{NaiveGnd}_{\mathcal{A}}(\psi_i,\theta) & \text{if } \varphi = \bigvee_i \psi_i \\ \bigwedge_{a\in A} \mathrm{NaiveGnd}_{\mathcal{A}}(\psi,[\theta \cup (x/\tilde{a})]) & \text{if } \varphi = \forall x\ \psi \\ \bigvee_{a\in A} \mathrm{NaiveGnd}_{\mathcal{A}}(\psi,[\theta \cup (x/\tilde{a})]) & \text{if } \varphi = \exists x\ \psi \end{cases}$$

The ground formula produced by Algorithm 1 is not a grounding of φ over $\mathcal{A}$ (according to Definition 3.1.2), because it does not take into account the interpretations of σ given by $\mathcal{A}$. To

produce a grounding of φ over $\mathcal{A}$, we may conjoin a set of atoms giving that information with $\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi, \emptyset)$:

$$\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi, \emptyset) \wedge \bigwedge_{\substack{P \in \sigma \\ \bar{a} \in A^{|P|}}} P(\bar{x})[\bar{x}/\tilde{\bar{a}}] \tag{3.1}$$

We can further produce a *reduced grounding* from 3.1 by "evaluating out" all atoms of the instance vocabulary. From now on we use $\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi)$ to refer to the resulting reduced grounding of formula φ over $\mathcal{A}$.

Example 3.2.1. Let φ be the formula $\forall x, y\ (\neg Edge(x, y) \vee (C(x) \wedge C(y)))$ defined over domain $A = \{1, 2, 3\}$. Figure 3.1 shows the progress of Algorithm 1 on this formula. If we have $\mathcal{A} = \mathcal{G} = \{(1, 2), (2, 1), (2, 3), (3, 2)\}$ as our problem instance, the set of atoms to be conjoined with $\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi, \emptyset)$ to produce the naive grounding is:

$$\begin{gathered}\{E(1, 2), E(2, 1), E(2, 3), E(3, 2), \\ \neg E(1, 1), \neg E(2, 2), \neg E(3, 3), \\ \neg E(1, 3), \neg E(3, 1)\}.\end{gathered}$$

Using this, we can evaluate out some parts of the tree and remove them from the final ground formula, hence producing a reduced grounding of φ with respect to $\mathcal{A}$ (see Figure 3.2). For instance, in the left-most branch of Figure 3.2, since we know that $E(1, 1)$ is false, we can replace it accordingly. Now, the value of the $\vee$ operator can be set to true, regardless of the value of the other branch. But true is the zero element of $\wedge$, so we can remove the $\vee$ branch all together.

3.3 Relational Algebraic Grounding

The grounding algorithm used in Enfragmo constructs a grounding by a bottom-up process that parallels database query evaluation, based on an extension of the relational algebra. We give a rough sketch of the method here: further details can be found in, e.g., [25, 27]. Given a structure (database) $\mathcal{A}$, a boolean query is a formula φ over the vocabulary of $\mathcal{A}$, and query answering is evaluating whether φ is true, i.e., $\mathcal{A} \models \varphi$. In the context of grounding, φ has some additional vocabulary beyond that of $\mathcal{A}$, and producing a reduced grounding involves evaluating out the instance vocabulary, and producing a ground formula representing the expansions of $\mathcal{A}$ for which φ is true.

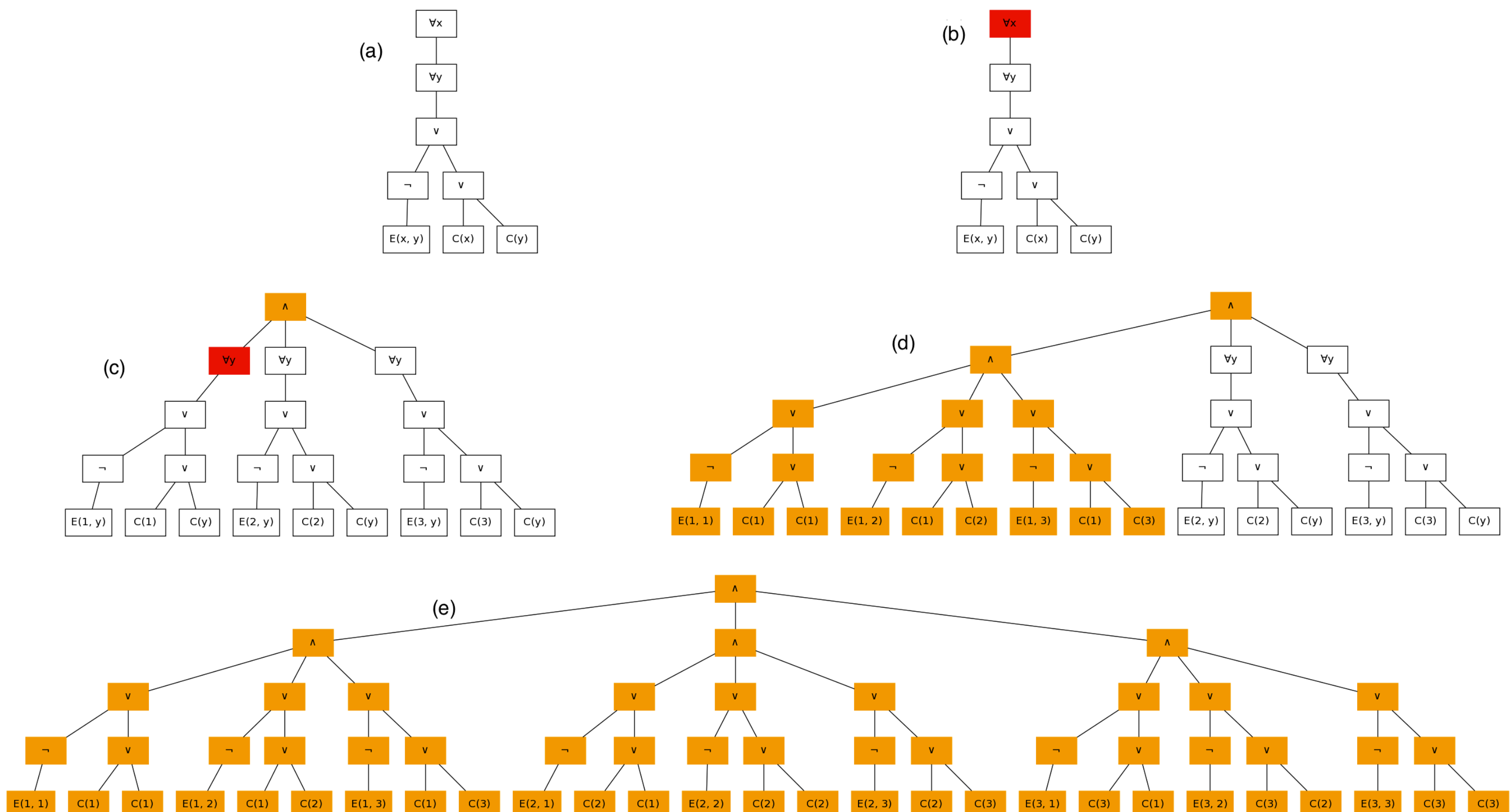

Figure 3.1: Top-down grounding steps of formula $\forall x, y\ (\neg Edge(x, y) \vee (C(x) \wedge C(y)))$ over domain $A = \{1, 2, 3\}$: (a) The FO formula tree. (b) & (c) & (d) Evaluation of quantifiers from top to bottom (red nodes denote the next quantifier to be evaluated). (e) The final ground formula tree.

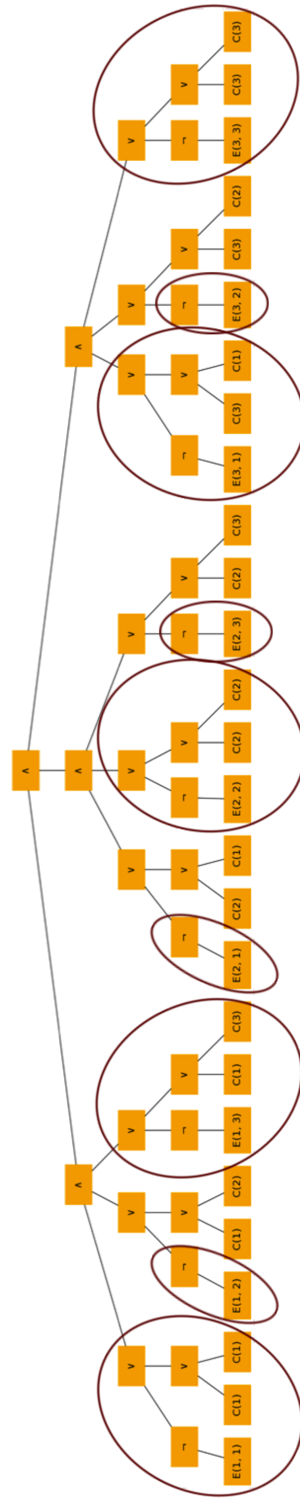

Figure 3.2: Circled subtrees show the subformulas that can be removed from the ground formula in reduced grounding.

For each sub-formula $\alpha(\bar{x})$ with free variables $\bar{x}$, we call the set of reduced groundings for α under all possible ground instantiations of $\bar{x}$ an *answer to* $\alpha(\bar{x})$. We represent answers with tables on which the extended algebra operates. An X-relation, in databases, is a k-ary relation associated with a k-tuple of variables X, representing a set of instantiations of the variables of X. The grounding method uses extended X-relations, in which each tuple $\bar{a}$ is associated with a formula. In particular, if R is the answer to $\alpha(\bar{x})$, then R consists of the pairs $(\bar{a}, \alpha(\tilde{\bar{a}}))$. Since a sentence has no free variables, the answer to a sentence φ is a zero-ary extended X-relation, containing a single pair $(\langle\rangle, \psi)$, associating the empty tuple with formula ψ, which is a reduced grounding of φ.

The relational algebra has operations corresponding to each connective and quantifier in FO: complement (negation); join (conjunction); union (disjunction), projection (existential quantification); division or quotient (universal quantification). Each generalizes to extended X-relations. If $(\bar{a}, \alpha(\tilde{\bar{a}})) \in \mathcal{R}$ then we write $\delta_{\mathcal{R}}(\bar{a}) = \alpha(\tilde{\bar{a}})$. For example, the join of two extended relations is defined as follows:

Definition 3.3.1 ($\mathcal{R} \bowtie \mathcal{S}$)**.** The join of extended relations X-relation $\mathcal{R}$ and extended Y-relation $\mathcal{S}$ (both over domain A), denoted $\mathcal{R} \bowtie \mathcal{S}$, is the extended $X \cup Y$-relation $\{(\bar{a}, \psi) \mid \bar{a} : X \cup Y \to A, \bar{a}|_X \in \mathcal{R}, \bar{a}|_Y \in \mathcal{S}, \text{ and } \psi = \delta_{\mathcal{R}}(\bar{a}|_X) \wedge \delta_{\mathcal{S}}(\bar{a}|_Y)\}$;

It is easy to show that, if $\mathcal{R}$ is an answer to $\alpha_1(\bar{x})$ and $\mathcal{S}$ is an answer to $\alpha_2(\bar{y})$ (both wrt $\mathcal{A}$), then $\mathcal{R} \bowtie \mathcal{S}$ is an answer to $\alpha_1(\bar{x}) \wedge \alpha_2(\bar{y})$. The analogous property holds for the other operators.

To ground a $\sigma \cup \varepsilon$ formula φ on a σ-structure $\mathcal{A}$ with this algebra, we define the answer to atomic formula $P(\bar{x})$ as $(\bar{a}, \rho)$, where:

$$\rho = \begin{cases} P(\tilde{\bar{a}}) & \text{if } \; P \in \varepsilon \\ \top & \text{if } \; P \in \sigma \; \text{ and } \; \bar{a} \in P^{\mathcal{A}} \\ \bot & \text{if } \; P \in \sigma \; \text{ and } \; \bar{a} \notin P^{\mathcal{A}}. \end{cases} \tag{3.2}$$

Then we apply the algebra inductively, bottom-up, on the structure of the formula. At the top, we obtain the answer to φ, which is a relation containing only the pair $(\langle\rangle, \psi)$, where ψ is a reduced grounding of φ wrt $\mathcal{A}$.

Proposition 3.3.1. *Let $(\langle\rangle, \psi)$ be the answer to sentence φ after initialization (3.2) of atomic formulas according to structure $\mathcal{A}$, then:*

$$\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi) \equiv \psi$$

Note that if we set the answer to the instance predicates the same way as the expansion predicates, then what we get as the answer to φ (i.e. $(\langle\rangle, \psi)$) is a naive grounding of φ over $\mathcal{A}$ which is equivalent to 3.1.

Example 3.3.1. Let $\sigma = \{P\}$ and $\varepsilon = \{E\}$, and let $\mathcal{A}$ be a σ-structure with $P^{\mathcal{A}} = \{(1,2,3),(3,4,5)\}$. The following extended relation $\mathcal{R}$ is an answer to $\varphi_1 \equiv P(x,y,z) \wedge E(x,y) \wedge E(y,z)$:

x	y	z	ψ
1	2	3	$E(1,2) \wedge E(2,3)$
3	4	5	$E(3,4) \wedge E(4,5)$

Observe that $\delta_{\mathcal{R}}(1,2,3) = E(1,2) \wedge E(2,3)$ is a reduced grounding of $\varphi_1[(1,2,3)] = P(1,2,3) \wedge E(1,2) \wedge E(2,3)$, and $\delta_{\mathcal{R}}(1,1,1) = \bot$ is a reduced grounding of $\varphi_1[(1,1,1)]$.

The following extended relation is an answer to $\varphi_2 \equiv \exists z \varphi_1$:

x	y	ψ
1	2	$E(1,2) \wedge E(2,3)$
3	4	$E(3,4) \wedge E(4,5)$

Here, $E(1,2) \wedge E(2,3)$ is a reduced grounding of $\varphi_2[(1,2)]$. Finally, the following represents an answer to $\varphi_3 \equiv \exists x \exists y \varphi_2$, where the single formula is a reduced grounding of φ_3.

ψ
$[E(1,2) \wedge E(2,3)] \vee [E(3,4) \wedge E(4,5)]$

3.4 Transformation to CNF and Unit Propagation

Transformation of a formula in NNF to CNF is achievable by repeated application of the distributivity law (Equivalence 2.2) to move the disjunctions deeper in the formula tree, and as a result moving conjunctions up at the same time. But this technique might produce a CNF that is exponentially larger than the size of the input formula.

The method of Tseitin [30] for CNF transformation guarantees a linear increase in the size of the CNF formula. The resulting formula is not equivalent to the first formula, but they are equisatisfiable, meaning that if one is satisfiable the other one is also satisfiable. Moreover, every model for the resulting CNF formula is also a model for the original formula, which is important for model expansion.

We use Tseitin's method with two modifications. The method, usually presented for propositional formulas, involves adding a new atom corresponding to each sub-formula. Here, we use a

version for ground FO formulas, so the resulting CNF formula is also a ground FO formula, over vocabulary $\tau = \sigma \cup \varepsilon \cup \tilde{A} \cup \omega$, where ω is a set of new relation symbols which we call "Tseitin symbols". To be precise, ω consists of a new k-ary relation symbol $\lceil \psi \rceil$ for each subformula ψ of φ with k free variables. We also formulate the transformation for formulas in which conjunction and disjunction may have arbitrary arity.

Let $\gamma = \mathrm{NaiveGnd}_{\mathcal{A}}(\varphi, \emptyset)$. Each subformula α of γ is a grounding over $\mathcal{A}$ of a substitution instance $\psi(\bar{x})[\theta]$, of some subformula ψ of φ with free variables $\bar{x}$. To describe the CNF transformation, it is useful to think of labelling the subformulas of γ during grounding as follows. If α is a grounding of formula $\psi(\bar{x})[\theta]$, label α with the ground atom $\lceil \psi \rceil(\bar{x})[\theta]$. To minimize notation, we will denote this atom by $\widehat{\alpha}$, setting $\widehat{\alpha}$ to α if α is an atom. Now, we have for each sub-formula α of the ground formula ψ, a unique ground atom $\widehat{\alpha}$, and we carry out the Tseitin transformation to CNF using these atoms.

Definition 3.4.1. For ground formula ψ, we denote by $\mathrm{CNF}(\psi)$ the following set of ground clauses. For each sub-formula α of ψ of form:

1. $(\wedge_i \alpha_i)$, include in $\mathrm{CNF}(\psi)$ the set of clauses $\{(\neg\widehat{\alpha} \vee \widehat{\alpha_i})\} \cup \{(\vee_i \neg\widehat{\alpha_i} \vee \widehat{\alpha})\}$
2. $(\vee_i \alpha_i)$, include in $\mathrm{CNF}(\psi)$ the set of clauses $\{(\widehat{\alpha} \vee \neg\alpha_i)\} \cup \{(\vee_i \alpha_i \vee \neg\widehat{\alpha})\}$
3. $(\neg\alpha_1)$, include in $\mathrm{CNF}(\psi)$ the set of clauses $\{(\neg\widehat{\alpha} \vee \alpha_1)\} \cup \{(\widehat{\alpha} \vee \alpha_1)\}$.

If ψ is a grounding of φ over $\mathcal{A}$, then CNF(ψ) is also. The models of ψ are exactly the reducts of the models of CNF(ψ) defined by $\sigma \cup \varepsilon \cup \tilde{A}$. CNF($\psi$) can trivially be viewed as a propositional CNF formula. This propositional formula can be sent to a SAT solver, and if a satisfying assignment is found, a model of φ which is an expansion of $\mathcal{A}$ can be constructed from it.

Definition 3.4.2 (UP(γ))**.** Let γ be a ground FO formula in CNF. Define $\mathrm{UP}(\gamma)$, the result of applying unit propagation to γ, to be the fixed point of the following operation:

> If γ contains a unit clause (l), delete from each clause of γ every occurrence of $\neg l$, and delete from γ every clause containing l.

Now, $\mathrm{CNF}(\mathrm{NaiveGND}_{\mathcal{A}}(\varphi))$ is the result of producing the naive grounding of φ over $\mathcal{A}$, and transforming it to CNF in the standard way, and $\mathrm{UP}(\mathrm{CNF}(\mathrm{NaiveGND}_{\mathcal{A}}(\varphi)))$ is the formula obtained after simplifying it by executing unit propagation. These two formulas provide reference points for measuring the reduction in ground formula size obtained by LUP in later chapters.

3.4.1 Autarkies and Autark Subformulas

In the literature, an *autarky* [26] is informally a “self-sufficient“ model for some clauses which does not affect the remaining clauses of the formula. An autark subformula is a subformula which is satisfied by an autarky. To see how an autark subformula may be produced during grounding, let $\lambda = \gamma_1 \vee \gamma_2$ and imagine that the value of subformula γ_1 is true according to our bound structure. Then λ will be true, regardless of the value of γ_2, and the grounder will replace its subformula with its truth value, whereas in the case of naive grounding, the grounder does not have that information during the grounding. So it generates the set of clauses for this subformula as: $\{(\neg\lambda \vee \gamma_1 \vee \gamma_2), (\neg\gamma_1 \vee \lambda), (\neg\gamma_2 \vee \lambda)\}$. Now the propagation of the truth value of λ_1 and subsequently λ, results in elimination of all the three clauses, but the set of clauses generated for γ_2 will remain in the CNF formula. We call γ_2 and the clauses made from that subformula autarkies.

The example suggests that this is a common phenomena and that the number of autarkies might be quite large in many groundings, as will be seen in Chapter 5.

3.5 Conclusion

In this chapter we discussed the notion of model expansion and grounding in theory and then provided two grounding algorithms: a top-down and a bottom-up algorithm. MXG grounder [25], and Enfragmo [6, 7] are examples of bottom-up grounders that employ algebraic database theory to do the grounding. This enables them to use the best practices and optimization techniques from database theory.

Another example of a bottom-up grounder is the grounder of the Alloy system [18] , called Kodkod [28]. Alloy uses a mixture of FO logic and relational algebraic operators in its specification language. The system is used for model-checking of software abstractions expressed in Alloy’s specification language. Kodkod is the program that translates the specification directly to SAT. Kodkod represents the predicates (atomic formulas) as sparse matrices and translates logical operators on those atomic formulas to matrix operations.

Top-down grounding is the technique used in the grounder of the IDP system [35], GIDL [33]. The system first tries to find some bounds on the subformulas of the input theory (formula) and then uses that information to reduce the size of the ground formula. Finally the ground formula is give to a SAT solver with a richer syntax (with sets, aggregates and inductive definitions) than typical CNF solvers.

In later chapters we focus on Enfragmo, and in Chapter 5 we compare the bounds computation technique of GIDL with LUP.

Chapter 4

Lifting Unit Propagation

In this chapter we present a grounding algorithm that produces groundings of φ over a specific class of partial structures, which we call *bound structures*, related to φ and $\mathcal{A}$. These partial structures are called bound structures because they bound the sets of tuples of domain elements on which the subformulas are true/false. For instance, if a bound structure sets the value of subformula ψ to true (resp. false) at $\bar{a}$, then it is bounding the set of tuples on which ψ is false (resp. true). This naming was first introduced in [34]. One gets smaller groundings by replacing the subformulas with their values during the grounding. These values are obtained from the bound structure.

Building on the notion of bound structures we define Lifted Unit Propagation (LUP) structures and we show that top-down grounding over this special bound structure produces ground formulas which, after CNF generation, unit propagation cannot simplify any further. In other words, the ground formula is as small as it can be relative to doing normal grounding and UP afterwards. Intuitively, we bring forward the work of UP before the grounding and we do it in a higher level, hence the word "lifted".

Later we show how to use Datalog programs for construction of LUP structures and finally we claim that the same result as top-down grounding over LUP structures can be obtained by bottom-up grounding over the same structure. In fact, the bottom-up adaptation is the method that was implement in the grounder Enfragmo, since it is a bottom-up grounder.

4.1 Bound Structures

The specific structures of interest are over a vocabulary expanding the vocabulary of φ in a certain way. We will call a vocabulary τ a Tseitin vocabulary for φ if it contains, in addition to the symbols

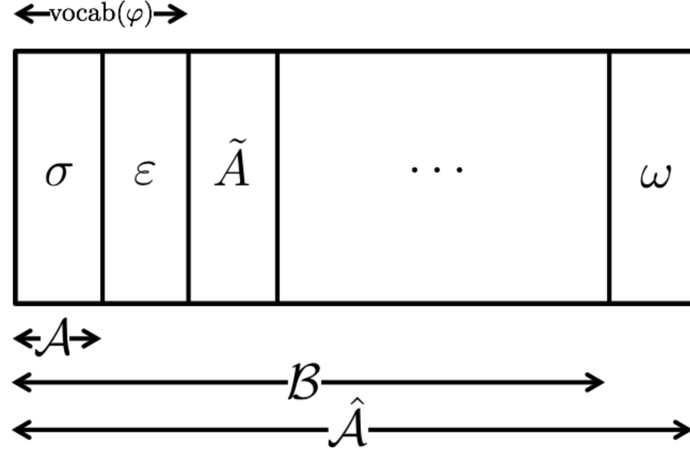

Figure 4.1: Structures and the vocabularies on which they are defined.

of φ, the set ω of Tseitin symbols for φ (see Section 3.4). We call a τ-structure a "Tseitin structure for φ" if the interpretations of the Tseitin symbols respect the special role of those symbols in the Tseitin transformation. For example, if α is $\alpha_1 \wedge \alpha_2$, then $\hat{\mathcal{A}}$ must not make $\widehat{\alpha}^{\mathcal{A}} \leftrightarrow \widehat{\alpha_1}^{\mathcal{A}} \wedge \widehat{\alpha_2}^{\mathcal{A}}$ false (according to the FO semantics for partial structures of Section 2.1.4). The vocabulary of the formula $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$ is a Tseitin vocabulary for φ, and every model of that formula is a Tseitin structure for φ.

Definition 4.1.1 (Bound Structures)**.** Let φ be a formula, and $\mathcal{A}$ be a structure for a subset of the vocabulary of φ. A bound structure $\hat{\mathcal{A}}$ for φ and $\mathcal{A}$ is a partial Tseitin structure for φ that is a conservative strengthening of $\mathcal{A}$ with respect to φ.

Intuitively, a bound structure provides a way to represent the information from the instance together with additional information, including information about the Tseitin symbols in a grounding of φ, that we may derive (by any means), provided that the information does not eliminate any possible expansions of the instance that model φ (see Figure 4.1).

Let τ be the minimum vocabulary for bound structures for φ and $\mathcal{A}$ (i.e. the dotted part of Figure 4.1 is empty). The bound structures for φ and $\mathcal{A}$ with vocabulary τ form a lattice under the partial order $\leq$. The minimum element of this lattice is $\mathcal{A}$ (expanded with empty $\tau - \sigma$ relations), while the maximum element is defined exactly for the atoms of $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$ which have the same

truth value in every Tseitin τ-structure that satisfies φ, i.e. the true atoms in the intersection of all the Tseitin τ-structures that satisfy φ[1].

Definition 4.1.2 (Grounding over a bound structure)**.** Let $\hat{\mathcal{A}}$ be a bound structure for φ and $\mathcal{A}$. A formula ψ, over a Tseitin vocabulary for φ which includes $\tilde{A}$, is a grounding of φ over $\hat{\mathcal{A}}$ iff

1. if there is a total strengthening of $\hat{\mathcal{A}}$ that satisfies φ, then there is one that satisfies ψ;
2. if $\mathcal{B}$ is a total Tseitin structure for φ which strengthens $\hat{\mathcal{A}}$, gives $\tilde{A}$ the intended interpretation and satisfies ψ, then it satisfies φ.

A grounding ψ of φ over $\hat{\mathcal{A}}$ need not be a grounding of φ over $\mathcal{A}$. If we conjoin with ψ ground atoms representing the information contained in $\hat{\mathcal{A}}$, then we do obtain a grounding of φ over $\mathcal{A}$. In practice, for ψ which is a reduced grounding, we just send CNF(ψ) to the SAT solver, and if a satisfying assignment is found, we add the missing information back in at the time we construct a model for φ (see Figure 1.1 of the Chapter 1).

4.2 Top-down Grounding over a Bound Structure

Algorithm 2 produces a grounding of φ over a bound structure $\hat{\mathcal{A}}$ for $\mathcal{A}$ and φ. Gnd and $Simpl$ are defined by mutual recursion. Gnd performs expansions and substitutions, while $Simpl$ performs lookups in $\hat{\mathcal{A}}$ to see if the grounding of a sub-formula may be left out. $Eval$ provides the base cases, evaluating ground atoms over $\sigma \cup \varepsilon \cup \tilde{A} \cup \omega$ in $\hat{\mathcal{A}}$.

The stronger $\hat{\mathcal{A}}$ is, the smaller the ground formula produced by Algorithm 2. For instance, if we set $\hat{\mathcal{A}}$ to be undefined everywhere (i.e., to just give the domain), then Algorithm 2 produces $\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi, \emptyset)$. On the other hand, if $\hat{\mathcal{A}}$ is set to $\mathcal{A}$, we get the reduced grounding obtained by $\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi)$. Ultimately, if we set $\hat{\mathcal{A}}$ to be the intersection of all the Tseitin τ-structures that satisfy φ (the aforementioned maximum bound structure), we get the smallest possible grounding φ over $\mathcal{A}$ using Algorithm 2.

Proposition 4.2.1. *Algorithm 2 produces a grounding of φ over $\hat{\mathcal{A}}$.*

4.3 LUP Structures

Section 4.2 introduced bound structures and how to ground over them. $\mathcal{LUP}(\varphi, \mathcal{A})$ is a special kind of bound structure in the sense that grounding over this bound structure and subsequent CNF

[1]This is the structure produced by "Most Optimum Propagator" in [36].

Algorithm 2 Top-Down Grounding over Bound Structure $\hat{\mathcal{A}}$ for φ and $\mathcal{A}$

$$Gnd_{\hat{\mathcal{A}}}(\varphi,\theta) = \begin{cases} Eval_{\hat{\mathcal{A}}}(P,\theta) & \varphi \text{ is an atom } P(\bar{x}) \\ \neg Eval_{\hat{\mathcal{A}}}(P,\theta) & \varphi \text{ is a negated atom } \neg P(\bar{x}) \\ \bigwedge_i Simpl_{\hat{\mathcal{A}}}(\psi_i,\theta) & \varphi = \bigwedge_i \psi_i \\ \bigvee_i Simpl_{\hat{\mathcal{A}}}(\psi_i,\theta) & \varphi = \bigvee_i \psi_i \\ \bigwedge_{a\in A} Simpl_{\hat{\mathcal{A}}}(\psi,\theta \cup (x/\tilde{a})) & \varphi = \forall x\ \psi \\ \bigvee_{a\in A} Simpl_{\hat{\mathcal{A}}}(\psi,\theta \cup (x/\tilde{a})) & \varphi = \exists x\ \psi \end{cases}$$

$$Eval_{\hat{\mathcal{A}}}(P,\theta) = \begin{cases} \top & \hat{\mathcal{A}} \models P[\theta] \\ \bot & \hat{\mathcal{A}} \models \neg P[\theta] \\ P(\bar{x})[\theta] & \text{o.w} \end{cases}$$

$$Simpl_{\hat{\mathcal{A}}}(\psi,\theta) = \begin{cases} \top & \hat{\mathcal{A}} \models \lceil\psi\rceil[\theta] \\ \bot & \hat{\mathcal{A}} \models \neg\lceil\psi\rceil[\theta] \\ Gnd_{\hat{\mathcal{A}}}(\psi,\theta) & \text{o.w.} \end{cases}$$

transformation makes unit propagation on the resulting ground formula redundant. In other words, the resulting ground formula is as small as it can get with respect to UP, so doing UP on it would not make it any smaller. In this section we define $\mathcal{LUP}(\varphi,\mathcal{A})$, and a method for constructing it.

Definition 4.3.1 ($\mathcal{LUP}(\varphi,\mathcal{A})$)**.** Let Units denote the set of unit clauses that appear during the execution of UP on $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$. If Units is:

- Unsatisfiable: then no LUP structure exists.
- Satisfiable: then the LUP structure for φ and $\mathcal{A}$ is the unique bound structure for φ and $\mathcal{A}$ for which:

$$\chi^{\mathcal{A}}_{\lceil\psi\rceil}(\bar{a}) = \begin{cases} \top & \lceil\psi\rceil(\tilde{\bar{a}}) \in \mathrm{Units} \\ \bot & \neg\lceil\psi\rceil(\tilde{\bar{a}}) \in \mathrm{Units} \\ \infty & \text{o.w.} \end{cases} \tag{4.1}$$

for every subformula ψ of φ.

Theorem 4.3.2. *If* Units *is satisfiable, then the LUP structure of Definition* 4.3.1 *exists, and it is a unique bound structure for* φ *and* $\mathcal{A}$.

Proof. (Sketch) We have to show that the above construction method yields a unique bound structure, provided that Units is satisfiable. In other words, we have to show that given the precondition,

there exists a partial structure $\hat{\mathcal{A}}$ that is a unique Tseitin structure and a conservative strengthening of the instance structure $\mathcal{A}$. Uniqueness is immediate, due to the construction method. Now, since Units is satisfiable, it means that no two unit clauses are contradictory, i.e. for any atom α, not both α and $\neg\alpha$ can be in Units. Based on this, and based on the fact that unit propagation does not violate semantical relation between the Tseitin atoms, one can prove, in a case by case manner, that there are no inconsistencies in the values of the Tseitin predicates and hence $\hat{\mathcal{A}}$ is a Tseitin structure.

Now it only remains to prove that $\hat{\mathcal{A}}$ is a conservative strengthening of $\mathcal{A}$. It is a strengthening, because it has the same value as $\mathcal{A}$ wherever $\mathcal{A}$ is defined (remember from Section 3.2 that the interpretations of instance vocabulary are included as units in $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$). And finally $\hat{\mathcal{A}}$ is a *conservative* strengthening of φ and $\mathcal{A}$, because every possible model of φ that is a strengthening of $\mathcal{A}$ is also a strengthening of $\hat{\mathcal{A}}$. To see this, observe that no possible satisfying assignments for $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$ is lost by unit propagation. In other words, the value of any atom that UP infers remains the same in all possible satisfying assignments. By the same token, every possible model of φ that is a strengthening of $\mathcal{A}$ has the same value as $\mathcal{LUP}(\varphi, \mathcal{A})$ wherever they are both defined. □

It is instructive to talk about the boundary conditions regarding the above definition. For instance, satisfiability of Units plays a crucial role in the existence of LUP structure. Unsatisfiability of that set means that the given formula φ is not $\mathcal{A}$-satisfiable, because UP could find the unsatisfiability of $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$. In that case no LUP structure can exists, by the definition. In another case, the formula might still not be $\mathcal{A}$-satisfiable but UP cannot detect that. In this case an LUP structure does exists, but the resulting ground formula obtained by grounding over that structure would not be satisfiable.

On the other hand, if unit propagation alone can determine that $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$ is satisfiable, then φ's Tseitin predicate in the LUP structure would be set to $\top$. In that case the result of $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\varphi, \emptyset)$ would be just an empty formula.

Since Algorithm 2 produces a grounding, according to Definition 4.1.2, for any bound structure, it produces a grounding for φ over $\mathcal{LUP}(\varphi, \mathcal{A})$. Now we have to show that grounding over a structure with above properties, frees us from doing unit propagation on the ground formula. In other words, we must show that $\mathrm{CNF}(Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\varphi, \emptyset))$ produces a smaller grounding compared to $\mathrm{UP}(\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi)))$ without eliminating any satisfying assignments (and ultimately models for φ). While conservation of models is immediate from the fact that $\mathcal{LUP}(\varphi, \mathcal{A})$ is a bound structure for $\mathcal{A}$ and φ, containment of $\mathrm{CNF}(Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\varphi, \emptyset))$ inside $\mathrm{UP}(\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi)))$

requires some elaboration. The following theorem states this more formally:

Theorem 4.3.3. *Let φ be a formula over $\tau (= \sigma \cup \epsilon)$ and $\mathcal{A}$ a σ-structure. Then:*

$$\mathrm{CNF}(Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\varphi, \emptyset)) \subseteq \mathrm{UP}(\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))). \tag{4.2}$$

Remark 4.3.4. The "$\subseteq$" symbol in Theorem 4.3.3 is an extension of the same symbol in set theory. If A and B are two sets of clauses, then $A \subseteq B$ iff every clause c in A is also in B or there is a clause c' in B which c is a subset of.

Proof. From Proposition 4.3.2 and Algorithm 2, and from the fact that $\mathcal{LUP}(\varphi, \mathcal{A})$ is a conservative strengthening of $\mathcal{A}$ we already know that:

$$\mathrm{CNF}(Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\varphi, \emptyset)) \subseteq \mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi)). \tag{4.3}$$

To prove the theorem we must show that whatever UP can eliminate from $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$ is already eliminated from $\mathrm{CNF}(Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\varphi, \emptyset))$. In order to do that we have to identify the type of clauses that are eliminated by UP. Based on Definition 3.4.2 there are two types of clauses that are removed by unit propagation: The unit clause itself, and any other clause that contains that unit literal. The unit clauses are removed from $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\varphi, \emptyset)$ by either $Simpl$ or $Eval$ functions. Now for non-unit clauses removed by UP, observe that they belong to the set of clauses generated for a $\vee$, $\wedge$ or $\neg$ connective in the ground formula. We prove the claim for $\vee$ connective, which is either the result of unwinding an existential quantifier or a simple disjunction in the input formula. Similar arguments work for other connectives.

Without loss of generality let the connective have only two oprands and let it be a disjunction in the input formula (and not an unwinding of some existential quantifier): $\psi(\bar{x}) = \psi_1(\bar{x}) \vee \psi_2(\bar{x})$. The set of clauses created for this connective in $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$ would be:

1. $(\neg\lceil\psi\rceil \vee \lceil\psi_1\rceil \vee \lceil\psi_2\rceil)[\theta]$
2. $(\neg\lceil\psi_1\rceil \vee \lceil\psi\rceil)[\theta]$
3. $(\neg\lceil\psi_2\rceil \vee \lceil\psi\rceil)[\theta]$

The following cases show the scenarios in which one of these clauses can be eliminated. For each case we show that the removal of that clause by UP from $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$ is accounted for in $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\psi, \theta)$ (We remove the substitution function $[\theta]$ for better readability, but its existence is assumed):

i. $(\lceil\psi\rceil)$ is a unit. In this case UP removes the second and third clause, turning the first clause into a binary clause of $(\lceil\psi_1\rceil \vee \lceil\psi_2\rceil)$, which might lead to an autarky (see Section 3.4.1). Better result is achieved by $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\psi,\theta)$. $Simpl$ leaves out the whole branch for $\lceil\psi\rceil$ and none of the above clauses are generated. So in this case $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\psi,\theta)$ is even performing better, by not generating an autarky.

ii. $(\neg\lceil\psi\rceil)$ is a unit. Same as in (i), $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\psi,\theta)$ does not generate any clauses for the subformula.

iii. $(\lceil\psi_1\rceil)$ is a unit. UP eliminates the first clause, makes the second clause a unit clause $(\lceil\psi\rceil)$ and finally eliminates the third clause because of $\lceil\psi\rceil$. Now to argue that $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\psi,\theta)$ produces the same result, we have to take into account that $(\lceil\psi\rceil)$ becomes a unit clause so it is in Unit and as in (i) $Simpl$ leaves out the whole branch for it. So again $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\psi,\theta)$ does not generate any clauses for this subformula.

iv. $(\neg\lceil\psi_1\rceil)$ is a unit. UP eliminates the second clause and turns the first clause into a binary clause. The resulting clauses would be $\{(\neg\lceil\psi\rceil \vee \lceil\psi_2\rceil), (\lceil\psi\rceil \vee \neg\lceil\psi_2\rceil)\}$, which means that $\lceil\psi\rceil$ and $\lceil\psi_2\rceil$ are logically equivalent. Exact result is obtained from $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\psi,\theta)$, by leaving out the ψ_1 branch from the grounding by either $Simpl$ or $Eval$ functions. More clearly, since $\lceil\psi_1\rceil$ is false and false is the zero element of $\vee$, the value of $\lceil\psi\rceil$ only depends on $\lceil\psi_2\rceil$, hence $\lceil\psi\rceil$ and $\lceil\psi_2\rceil$ are logically equivalent.

v. The two cases for $\lceil\psi_2\rceil$ is similar to $\lceil\psi_1\rceil$.

By the above reasoning we have shown that any clause that is eliminated by UP on the ground instance $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$ is not in $\mathrm{CNF}(Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\psi,\theta))$, and thus the proof is complete. □

One important observation from the proof of Theorem 4.3.3 is that in some cases (such as (i) and (iii)) some parts of the ground formula form autarkies and hence can be left out from the final grounding, since their value does not have any effect on the satisfiability of the finally produced CNF formula, but $\mathrm{CNF}(\mathrm{NaiveGnd}_{\mathcal{A}}(\varphi))$ is unable to identify them. This is one of the benefits of doing LUP, because during grounding over $\mathcal{LUP}(\varphi,\mathcal{A})$, the grounder has extra information, thanks to the bound structure, and can use that information to prevent itself from grounding subformulas that would lead to autarkies. autarky removal, as can be seen in Chapter 5, is an important factor in the dominance of LUP grounding over normal grounding.

4.3.1 A Datalog Program for LUP Structure Construction

To construct $\mathcal{LUP}(\varphi, \mathcal{A})$, we use a Datalog program $\Delta_{\mathcal{LUP}}(\varphi, \mathcal{A})$ obtained from φ and $\mathcal{A}$. In order to generate the Datalog program one has to traverse the formula tree and for each subformula in that tree create a Datalog rule depending on the type of the subformula (the type of the top-most operator of that subformula). In this Datalog program, we use distinct vocabulary symbols for the sets of tuples which $\hat{\mathcal{A}}$ sets to true and false for each Tseitin symbol (i.e. subformula of φ). The algorithm works based on the notion of *True (False) bounds*:

Definition 4.3.5 (Formula-Bound)**.** A *True (resp. False) bound* for a subformula $\psi(\bar{x})$ according to bound structure $\hat{\mathcal{A}}$ is the relation denoted by T_ψ (resp. F_ψ) such that:

1. $\bar{a} \in T_\psi \Leftrightarrow \lceil \psi \rceil^{\hat{\mathcal{A}}}(\bar{a}) = \top$
2. $\bar{a} \in F_\psi \Leftrightarrow \lceil \psi \rceil^{\hat{\mathcal{A}}}(\bar{a}) = \bot$

Naturally, when $\lceil \psi \rceil^{\hat{\mathcal{A}}}(\bar{a}) = \infty$, $\bar{a}$ is not contained in either T_ψ or F_ψ.

Table 4.1 provides a set of generic rules, based on which, the Datalog rules for a specific formula are created. The *type* column indicates the type of the subformula in φ, and the *rules* columns identify the Datalog rules generated for this subformula. As mentioned before, one has to traverse φ and depending on the type of the top-most operator, create the appropriate rules from Table 4.1.

These rules reflect the reasoning that UP can do. For example consider rule $(\vee_i \psi_i)$ of $\downarrow t$ for $\gamma(\bar{x}) = \psi_1(\bar{x}_1) \vee \cdots \vee \psi_N(\bar{x}_N)$, and for some $i \in \{1, \ldots, N\}$:

$$T_{\psi_i}(\bar{x}_i) \leftarrow T_\gamma(\bar{x}) \wedge \bigwedge_{j \neq i} F_{\psi_j}(\bar{x}_j).$$

This states that when a tuple $\bar{a}$ satisfies γ but falsifies all disjuncts, ψ_j, of γ except for one, namely ψ_i, then it must satisfy ψ_i.

Up to now we have created the rules of $\Delta_{\mathcal{LUP}}(\varphi, \mathcal{A})$. Observe that we only used the formula φ and the domain of the input structure $\mathcal{A}$ to do this and we have not yet incorporated the information about the relations of $\mathcal{A}$ (i.e. the instance predicates). That information is represented as the facts of the datalog program. More formally, for every $P \in \sigma$ and $\bar{a} \in A^k$, $\mathrm{Fact}(\Delta_{\mathcal{LUP}}(\varphi, \mathcal{A}))$ contains $T_P(\bar{a})$ if $\bar{a} \in P^{\mathcal{A}}$, and contains $F_P(\bar{a})$ if $\bar{a} \notin P^{\mathcal{A}}$.

We also assume that φ is $\mathcal{A}$-satisfiable, so we can also encode this information as a fact by adding T_φ to $\mathrm{Fact}(\Delta_{\mathcal{LUP}}(\varphi, \mathcal{A}))$. Note that this is only an assumption to start the process.

type	$\downarrow t$ rules		
$(\vee_i \psi_i)$	$T_{\psi_i}(\bar{x}_i)$	$\leftarrow$	$T_\gamma(\bar{x}) \wedge \bigwedge_{j \neq i} F_{\psi_j}(\bar{x}_j)$, for each i
$(\wedge_i \psi_i)$	$T_{\psi_i}(\bar{x}_i)$	$\leftarrow$	$T_\gamma(\bar{x})$, for each i
$\exists y\ \psi(\bar{x}, y)$	$T_\psi(\bar{x}, y)$	$\leftarrow$	$T_\gamma(\bar{x}) \wedge \forall y' \neq y\ \ F_\psi(\bar{x}, y')$
$\forall y\ \psi(\bar{x}, y)$	$T_\psi(\bar{x}, y)$	$\leftarrow$	$T_\gamma(\bar{x})$
$P(\bar{x})$	$T_P(\bar{x})$	$\leftarrow$	$T_\gamma(\bar{x})$
$\neg P(\bar{x})$	$F_P(\bar{x})$	$\leftarrow$	$T_\gamma(\bar{x})$

type	$\uparrow t$ rules		
$(\vee_i \psi_i)$	$T_\gamma(\bar{x})$	$\leftarrow$	$\bigvee_i T_{\psi_i}(\bar{x}_i)$, for each i
$(\wedge_i \psi_i)$	$T_\gamma(\bar{x})$	$\leftarrow$	$\bigwedge_i T_{\psi_i}(\bar{x}_i)$, for each i
$\exists y\ \psi(\bar{x}, y)$	$T_\gamma(\bar{x})$	$\leftarrow$	$\exists y\ T_\psi(\bar{x}, y)$
$\forall y\ \psi(\bar{x}, y)$	$T_\gamma(\bar{x})$	$\leftarrow$	$\forall y\ T_\psi(\bar{x}, y)$
$P(\bar{x})$	$T_\gamma(\bar{x})$	$\leftarrow$	$T_P(\bar{x})$
$\neg P(\bar{x})$	$T_\gamma(\bar{x})$	$\leftarrow$	$F_P(\bar{x})$

type	$\downarrow f$ rules		
$(\vee_i \psi_i)$	$F_{\psi_i}(\bar{x}_i)$	$\leftarrow$	$F_\gamma(\bar{x})$, for each i
$(\wedge_i \psi_i)$	$F_{\psi_i}(\bar{x}_i)$	$\leftarrow$	$F_\gamma(\bar{x}) \wedge \bigwedge_{j \neq i} T_{\psi_j}(\bar{x}_j)$, for each i
$\exists y\ \psi(\bar{x}, y)$	$F_\psi(\bar{x}, y)$	$\leftarrow$	$F_\gamma(\bar{x})$
$\forall y\ \psi(\bar{x}, y)$	$F_\psi(\bar{x}, y)$	$\leftarrow$	$F_\gamma(\bar{x}) \wedge \forall y' \neq y\ \ T_\psi(\bar{x}, y')$
$P(\bar{x})$	$F_P(\bar{x})$	$\leftarrow$	$F_\gamma(\bar{x})$
$\neg P(\bar{x})$	$T_P(\bar{x})$	$\leftarrow$	$F_\gamma(\bar{x})$

type	$\uparrow f$ rules		
$(\vee_i \psi_i)$	$F_\gamma(\bar{x})$	$\leftarrow$	$\bigwedge_i F_{\psi_i}(\bar{x}_i)$, for each i
$(\wedge_i \psi_i)$	$F_\gamma(\bar{x})$	$\leftarrow$	$\bigvee_i F_{\psi_i}(\bar{x}_i)$, for each i
$\exists y\ \psi(\bar{x}, y)$	$F_\gamma(\bar{x})$	$\leftarrow$	$\forall y\ F_\psi(\bar{x}, y)$
$\forall y\ \psi(\bar{x}, y)$	$F_\gamma(\bar{x})$	$\leftarrow$	$\exists y\ F_\psi(\bar{x}, y)$
$P(\bar{x})$	$F_\gamma(\bar{x})$	$\leftarrow$	$F_P(\bar{x})$
$\neg P(\bar{x})$	$F_\gamma(\bar{x})$	$\leftarrow$	$T_P(\bar{x})$

Table 4.1: Rules for Bounds Computation

Now having the facts, we may evaluate the rules, thus obtaining a set of concrete bounds for the subformulas of φ.

Example 4.3.1. Let $\varphi = \forall x \, \neg I_1(x) \vee E_1(x)$, $\sigma = \{I_1, I_2\}$, and $\mathcal{A} = \big(\{1,2,3,4\}; I_1^{\mathcal{A}} = \{1\}\big)$. The relevant rules generated based on Table (4.1) are:

$$\begin{aligned}
T_{\neg I_1(x) \vee E_1(x)}(x) &\leftarrow T_\varphi \\
T_{I_1}(x) &\leftarrow I_1(x) \\
F_{\neg I_1(x)}(x) &\leftarrow T_{I_1}(x) \\
T_{E_1(x)}(x) &\leftarrow T_{\neg I_1(x) \vee E_1(x)}(x) \wedge F_{\neg I_1(x)}(x) \\
T_{E_1}(x) &\leftarrow T_{E_1(x)}(x)
\end{aligned}$$

We find that $T_{E_1} = \{1\}$; in other words: $E_1(1)$ is true in each model of φ expanding $\mathcal{A}$.

4.3.2 An Algorithm for Construction of LUP Structures

Our method for constructing $\mathcal{LUP}(\varphi, \mathcal{A})$ is given in Algorithm 3. Several lines in the algorithm require explanation. In line 1, the $\downarrow f$ rules are omitted from the set of constructed rules. Because φ is in NNF, the $\downarrow f$ rules do not contribute any information to the set of bounds. To see this, observe that every $\downarrow f$ rule has an atom of the form $F_\gamma(\bar{x})$ in its body. Intuitively, for one of these rules to contribute a defined bound, certain information must have previously been obtained regarding bounds for its parent. It can be shown, by induction, that, in every case, the information about a bound inferred by an application of a $\downarrow f$ rule must have previously been inferred by a $\uparrow f$ rule. In line 2 of the algorithm we compute bounds using only the two sets of rules, $\downarrow t$ and $\uparrow f$. This is justified by the fact that applying $\{\uparrow t, \downarrow t, \uparrow f\}$ to a fixpoint has the same effect as applying $\{\downarrow t, \uparrow f\}$

Algorithm 3 Computation of $\mathcal{LUP}(\varphi, \mathcal{A})$

1: Construct the rules $\{\uparrow t, \downarrow t, \uparrow f\}$
2: Compute bounds by evaluating the inductive definition $\{\downarrow t, \uparrow f\}$
3: **if** Bounds are inconsistent **then**
4:    **return** "$\mathcal{A}$ has no solution"
5:    {This corresponds to the case were Units are unsatisfiable in Definition 4.3.1 and hence no LUP structure exists.}
6: **end if**
7: Throw away $T_\psi(\bar{x})$ for all non-atomic subformulas $\psi(\bar{x})$
8: Compute new bounds by evaluating the inductive definition $\{\uparrow t\}$
9: **return** LUP structure constructed from the computed bounds, according to Definition 4.3.5 .

---

to a fixpoint and then applying the $\uparrow t$ rules afterwards. So we postpone the execution of the $\uparrow t$ rules to line 7.

Line 3 checks for the case that the definition has no model, which is to say that the rules allow us to derive that some atom is both in the true bound and the false bound for some subformula. This happens exactly when UP applied to the naive grounding would detect inconsistency.

Finally, in lines 7 and 8 we throw away the true bounds for all non-atomic subformulas, and then compute new bounds by evaluating the $\uparrow t$ rules, taking already computed bounds (with true bounds for non-atoms set to empty) as the initial bounds in the computation. To see why, observe that the true bounds computed in line 2 are based on the assumption that φ is $\mathcal{A}$-satisfiable. So $\lceil\varphi\rceil$ is set to true which stops the top-down bounded grounding algorithm of Section 4.2 from producing a grounding for φ. That is because the $Simpl$ function, considering the true bound for the φ, simply returns $\top$ instead of calling $Gnd_{\hat{\mathcal{A}}}(.,.)$ on subformulas of the φ. This also holds for all the formulas with true-bounds, calculated this way, except for the atomic formulas. So, we delete these true bounds based on the initial unjustified assumption, and then construct the correct true bounds by application of the $\uparrow t$ rules, in line 7. This is the main reason for postponing the execution of $\uparrow t$ rules.

Finally, the algorithm returns the LUP structure. The following proposition states this more formally:

Proposition 4.3.1. *Let φ be a formula over $\sigma \cup \epsilon$ and $\mathcal{A}$ a σ-structure. Algorithm 3 outputs $\mathcal{LUP}(\varphi, \mathcal{A})$ if such structure exists for φ and $\mathcal{A}$, and returns* "$\mathcal{A}$ has no solution" *otherwise.*

Implementation

The bottleneck of Algorithm 3 is the fixpoint computation of line 2. This computation can be regarded as solving a system of relational equations: Each true/false bound is a relation and every rule is an equation:

$$R_i = E_i(R_1, \ldots, R_n) \qquad \text{(for } i = 1 \ldots n). \tag{4.4}$$

The result of each R_i is determined by doing a relational algebraic operation on other relations, based on the equation E_i that defines R_i. In Datalog community [9] the Gauss-Seidel algorithm is considered as a relational adaptation of the fixpoint characterization of $\text{cons}(\Delta)$ operator of Section 2.2.2 in solving Datalog programs. The algorithm starts by initializing the R_is. Then computation of equations is iterated until the values of R_is do not change in two consecutive iterations, i.e. when the system reaches a fixpoint.

A modified version of this algorithm is used in the implementation of LUP, namely the *semi-naive* method. Let $D_i^k = R_i^k - R_i^{k-1}$, which is the set of new tuples that are added to relation R_i in iteration k. The semi-naive method uses the body of the Gauss-Seidel algorithm but instead of using the whole relation, which can be quit large, at each stage, like $k+1$, it only uses the differential of that relation at stage k and stage $k-1$, namely D_k. This way, because the differentials are much smaller compared to the whole relations, computation of the equations would be faster and thus the overall performance of the fixpoint computation.

4.3.3 Bottom Up Grounding with LUP Bounds

We can modify the bottom-up reduced grounding algorithm of Section 3.3 to ground using $\mathcal{LUP}(\varphi, \mathcal{A})$. We only need to change the base case for expansion predicates. To be precise, we set the answer to $P(\bar{x})$ to the set of pairs $(\bar{a}, \rho)$ such that:

$$\rho = \begin{cases} P(\tilde{\bar{a}}) & \text{if } \ P^{\mathcal{LUP}(\varphi,\mathcal{A})}(\bar{a}) = \infty \\ \top & \text{if } \ P^{\mathcal{LUP}(\varphi,\mathcal{A})}(\bar{a}) = \top \\ \bot & \text{if } \ P^{\mathcal{LUP}(\varphi,\mathcal{A})}(\bar{a}) = \bot. \end{cases} \tag{4.5}$$

Note that the above initialization method works even if P is an instance predicate, because $\mathcal{LUP}(\varphi, \mathcal{A})$ is a strengthening of instance structure $\mathcal{A}$.

Proposition 4.3.2. *Let $(\langle\rangle, \psi)$ be the answer to sentence φ after initialization (4.5) of atomic formulas according to $\mathcal{LUP}(\varphi, \mathcal{A})$, then:*

$$Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\varphi, \emptyset) \equiv \psi$$

where $Gnd_{\mathcal{LUP}(\varphi,\mathcal{A})}(\varphi, \emptyset)$ is the result of top-down grounding Algorithm 2 of φ over LUP structure $\mathcal{LUP}(\varphi, \mathcal{A})$.

This bottom-up method uses only the reduct of $\mathcal{LUP}(\varphi, \mathcal{A})$ defined by $\sigma \cup \varepsilon \cup \tilde{A}$, not the entire LUP structure. Also observe that the above bottom-up grounding algorithm mimics the second phase of Algorithm 3, i.e., a bottom-up truth propagation, except that it also propagates the falses. So, for bottom up grounding, we can omit line 8 from Algorithm 3.

Chapter 5

Experiments

In this chapter we present an empirical study of the effect of LUP on grounding size and on grounding and solving times. We also compare LUP with GWB in terms of these same measures. The implementation of LUP is within our bottom-up grounder Enfragmo, as described in this paper, and the implementation of GWB is in the top-down grounder GIDL , which is described in [34, 36]. GIDL has several parameters to control the precision of the bounds computation. In our experiments we use the default settings. We used MINISAT as the ground solver for Enfragmo. GIDL produces an output specifically for the ground solver MINISAT(ID), and together they form the IDP system [35].

We report data for instances of three problems: Latin Square Completion, Bounded Spanning Tree and Sudoku. The instances are latin_square.17068* instances of Normal Latin Square Completion, the 104_rand_45_250_* and 104_rand_35_250_* instances of BST, and the ASP contest 2009 instances of Sudoku from the Asparagus repository[1]. All experiments were run on a Dell Precision T3400 computer with a quad-core 2.66GHz Intel Core 2 processor having 4MB cache and 8GB of RAM, running CentOS 5.5 with Linux kernel 2.6.18.

In Tables 5.1 and 5.4, columns headed "Literals" or "Clauses" give the number of literals or clauses in the CNF formula produced by Enfragmo without LUP (our baseline), or these values for other grounding methods expressed as a percentage of the baseline value. In Tables 5.2 and 5.3, all values are times seconds. All the given values are the means for the entire collection of instances. Variances are not given, because they are very small. We split the instances of BST into two sets based on the number of nodes (35 or 45), because these two groups exhibit somewhat different

[1] http://asparagus.cs.uni-potsdam.de

behaviour, but within the groups variances are also small. In all tables, the minimum (best) values for each row are in bold face type, to highlight the conditions which gave best performance.

Table 5.1 compares the sizes of CNF formulas produced by Enfragmo without LUP (the base line) with the formulas obtained by running UP on the baseline formulas and by running Enfragmo with LUP. Clearly LUP reduces the size at least as much as UP, and usually reduces the size much more, due to the removal of autarkies.

	Enfragmo		Enfragmo+UP (%)		Enfragmo+LUP (%)	
Problem	Literals	Clauses	Literals	Clauses	Literals	Clauses
Latin Square	7452400	2514100	**0.07**	**0.07**	**0.07**	**0.07**
BST 45	22924989	9061818	0.96	0.96	**0.24**	**0.24**
BST 35	8662215	3415697	0.95	0.96	**0.37**	**0.37**
Sudoku	2875122	981668	0.17	0.18	**0.07**	**0.08**

Table 5.1: Impact of LUP on the size of the grounding. The first two columns give the numbers of literals and clauses in groundings produced by Enfragmo without LUP (the baseline). The other columns give these measures for formulas produced by executing UP on the baseline groundings (Enfragmo+UP), and for groundings produced by Enfragmo with LUP (Enfragmo+LUP), expressed as a fraction baseline values.

	Enfragmo			Enfragmo with LUP			Speed Up Factor		
Problem	Gnd	Solving	Total	Gnd	Solving	Total	Gnd	Solving	Total
Latin Square	**0.89**	1.39	**2.28**	3.27	**0.34**	3.61	-2.38	1.05	-1.33
BST 45	6.08	7.56	13.64	**2**	**1.74**	**3.74**	4.07	5.82	9.9
BST 35	2.13	2.14	4.27	**1.07**	**0.46**	**1.53**	1.06	1.68	2.74
Sudoku	**0.46**	1.12	**1.59**	2.08	**0.26**	2.34	-1.62	0.86	-0.76

Table 5.2: Impact of LUP on reduction in both grounding and (SAT) solving time. Grounding time here includes LUP computations and CNF generation.

Total time for solving a problem instance is composed of grounding time and SAT solving time. Table 5.2 compares the grounding and SAT solving time with and without LUP bounds. It is evident that the SAT solving time is always reduced with LUP. This reduction is due to the elimination of the unit clauses and autark subformulas from the grounding. Autark subformula elimination also affects the time required to convert the ground formula to CNF which reduces the grounding time, but in some cases the overhead imposed by LUP computation may not be made up for by this reduction.

As the table shows, when LUP outperforms the normal grounding we get factor of 3 speed-ups, whereas when it loses to normal grounding the slowdown is by a factor of 1.5.

Table 5.3 compares the size reductions obtained by LUP and by GWB in GIDL . The output of GIDL contains clauses and rules. The rules are transformed to clauses in (MINISAT(ID)). The measures reported here are after that transformation. LUP reduces the size much more than GWB, in most of the cases. This stems from the fact that GIDL 's bound computation does not aim for completeness wrt unit propagation. This also affects the solving time because the CNF formulas are much smaller with LUP as shown in Table 5.4. Table 5.4 shows that Enfragmo with LUP and MINISAT is always faster than GIDL with MINISAT(ID) with or without bounds, and it is in some cases faster than Enfragmo without LUP.

	Enfragmo (no LUP)		GIDL (no bounds)		Enfragmo with LUP		GIDL with bounds	
Problem	Literals	Clauses	Literals	Clauses	Literals	Clauses	Literals	Clauses
Latin Square	7452400	2514100	0.74	0.84	**0.07**	**0.07**	0.59	0.61
BST 45	22924989	9061818	0.99	1.02	**0.24**	**0.24**	0.25	**0.24**
BST 35	8662215	3415697	1.01	1.04	**0.37**	**0.37**	0.39	0.39
Sudoku	2875122	981668	0.56	0.6	**0.07**	**0.08**	0.38	0.39

Table 5.3: Comparison between the effectiveness of LUP and GIDL Bounds on reduction in grounding size. The columns under Enfragmo show the actual grounding size whereas the other columns show the ratio of the grounding size relative to that of Enfragmo (without LUP).

	Enfragmo			IDP			Enfragmo+LUP			IDP (Bounds)		
Problem	Gnd	Solving	Total	Gnd	Solving	Total	Gnd	Solving	Total	Gnd	Solving	Total
Latin Square	**0.89**	1.39	**2.28**	3	4.63	7.63	3.27	**0.34**	3.61	2.4	3.81	6.21
BST 45	6.08	7.56	13.64	7.25	20.84	28.09	2	**1.74**	**3.74**	**1.14**	4.45	5.59
BST 35	2.13	2.14	4.27	2.63	6.31	8.94	1.07	**0.46**	**1.53**	**0.67**	2.73	3.4
Sudoku	**0.46**	1.12	**1.59**	1.81	1.3	3.11	2.08	**0.26**	2.34	2.85	0.51	2.37

Table 5.4: Comparison of solving time for Enfragmo and IDP, with and without LUP/GIDL bounds.

Chapter 6

Discussion

In the context of grounding-based problem solving, we have described a method we call lifted unit propagation (LUP) for carrying out a process essentially equivalent to unit propagation before and during grounding. Our experiments indicate that the method can substantially reduce grounding size – even more than unit propagation itself, and sometimes reduce total solving time as well.

Our work was motivated by the results of [34, 36], which presented the method we have referred to as GWB. In GWB, bounds on sub-formulas of the specification formula are computed without reference to an instance structure, and represented with FO formulas. The grounding algorithm evaluates instantiations of these bound formulas on the instance structure to determine that certain parts of the naive grounding may be left out. If the bound formulas exactly represent the information unit propagation can derive, then LUP and GWB are equivalent (though implemented differently). However, generally the GWB bounds are weaker than the LUP bounds, for two reasons. First, no FO formula can define the bounds obtainable, with respect to an arbitrary instance structure. Second, to make the implementation in GIDL efficient, the computation of the bounds is heuristically truncated. This led us to ask how much additional reduction in formula size might be obtained by the complete LUP method, and whether the LUP computation could be done fast enough for this extra reduction to be useful in practice.

Our experiments with the Enfragmo and GIDL grounders show that, at least for some kinds of problems and instances, using LUP can produce much smaller groundings than the GWB implementation in GIDL . In our experiments, the total solving times for Enfragmo with ground solver MINISAT were always less than those of GIDL with ground solver MINISAT(ID). However, LUP reduced total solving time of Enfragmo with MINISAT significantly in some cases, and increased it — albeit less significantly — in others. Since there are many possible improvements of the LUP

implementation, the question of whether LUP can be implemented efficiently enough to be used all the time remains unanswered.

Investigating more efficient ways to do LUP, such as by using better data structures, is a subject for future work, as is consideration of other approximate methods such, as placing a heuristic time-out on the LUP structure computation, or dovetailing of the LUP computation with grounding. We also observed that the much of the reduction in grounding size obtained by LUP is due to identification of Autark sub-formulas. These cannot be eliminated from the naive grounding by unit propagation. Further investigation of the importance of these in practice is another direction we are pursuing. One more direction we are pursuing is the study of methods for deriving even stronger information than that represented by the LUP structure, to further reduce ground formula size, and possibly grounding time as well.

Bibliography

[1] The Asparagus Library of Examples for ASP Programs, http://asparagus.cs.uni-potsdam.de/.

[2] *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[3] *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *LNCS*. Springer, 2007.

[4] *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *LNCS*. Springer, 2010.

[5] *Advances in Artificial Intelligence - 24th Canadian Conference on Artificial Intelligence, Canadian AI 2011, St. John's, Canada, May 25-27, 2011. Proceedings*, volume 6657 of *LNCS*. Springer, 2011.

[6] Amir Aavani, Shahab Tasharrofi, Gulay Ünel, Eugenia Ternovska, and David G. Mitchell. Speed-up techniques for negation in grounding. In *LPAR (Dakar)* [4], pages 13–26.

[7] Amir Aavani, Xiongnan (Newman) Wu, Eugenia Ternovska, and David G. Mitchell. Grounding formulas with complex terms. In *Canadian Conference on AI* [5], pages 13–25.

[8] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[9] S Ceri, G Gottlob, and L Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.

[10] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[11] D. East and M. Truszczynski. Predicate-calculus based logics for modeling and solving search problems. *ACM Trans. Comput. Logic (TOCL)*, 7(1):38 – 83, 2006.

[12] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation, SIAM-AMC proceedings*, 7:43–73, 1974.

[13] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. A user's guide to gringo, clasp, clingo, and iclingo. Unpublished draft, 2008.

[14] M. Gebser, T. Schaub, and S. Thiele. Gringo: A new grounder for answer set programming. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)* [2], pages 266–271.

[15] E. Graedel. *Finite Model Theory and Descriptive Complexity*, pages 125–230. Springer, 2007.

[16] E. Graedel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M.Y. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Applications*. Springer, 2007.

[17] N. Immerman. *Descriptive complexity*. Springer Verlag, New York, 1999.

[18] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[19] S.C. Kleene. *Introduction to Metamathematics*. Bibliotheca Mathematica. North-Holland, 1952.

[20] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[21] L. Libkin. *Elements of Finite Model Theory*. Springer Verlag, 2004.

[22] David Mitchell and Eugenia Ternovska. Knowledge representation, search problems and model expansion. In *Knowing, Reasoning and Acting: Essays in Honour of Hector J. Levesque*, pages 347–362, 2011.

[23] David Mitchell, Eugenia Ternovska, Faraz Hach, and Raheleh Mohebali. Model expansion as a framework for modelling and solving search problems. Technical Report TR 2006-24, December 2006.

[24] David G. Mitchell and Eugenia Ternovska. A framework for representing and solving NP search problems. In Veloso and Kambhampati [32], pages 430–435.

[25] Raheleh Mohebali. A method for solving NP search based on model expansion and grounding. Master's thesis, Simon Fraser University, 2004.

[26] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2n steps. *Discrete Appl. Math.*, 10:287–295, March 1985.

[27] M. Patterson, Y. Liu, E. Ternovska, and A. Gupta. Grounding for model expansion in k-guarded formulas with inductive definitions. In *Proc. IJCAI'07)*, 2007.

[28] Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In *TACAS* [3], pages 632–647.

[29] Luca Trevisan. Lecture notes on computational complexity. 2004.

[30] G.S. Tseitin. *On the complexity of derivation in propositional calculus*, pages 115 – 125. 1968.

[31] Pashootan Vaezipoor, David Mitchell, and Maarten Mariën. Lifted unit propagation for effective grounding. *CoRR*, abs/1109.1317, 2011.

[32] Manuela M. Veloso and Subbarao Kambhampati, editors. *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. AAAI Press / The MIT Press, 2005.

[33] Johan Wittocx, Maarten Mariën, and Marc Denecker. GidL: A grounder for FO+. In *In Proc., Twelfth International Workshop on NMR,*, pages 189–198, September 2008.

[34] Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding with bounds. In *AAAI*, pages 572–577, 2008.

[35] Johan Wittocx, Maarten Mariën, and Marc Denecker. The IDP system: a model expansion system for an extension of classical logic. In Marc Denecker, editor, *LaSh*, pages 153–165, 2008.

[36] Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO(ID) with bounds. *J. Artif. Intell. Res. (JAIR)*, 38:223–269, 2010.