Grounding Formulas with Complex Terms

Amir Aavani, Xiongnan (Newman) Wu, Eugenia Ternovska, David Mitchell

Simon Fraser University
{aaa78,xwa33,ter,mitchel}@sfu.ca

Abstract. Given a finite domain, grounding is the the process of creating a variable-free first-order formula equivalent to a first-order sentence. The first-order sentences can be used to describe a combinatorial search problem. Efficient grounding algorithms would help in solving such problems effectively and make advanced solver technology (such as SAT) accessible to a wider variety of users. One promising method for grounding is based on the relational algebra from the field of Database research. We describe the extension of this method to ground formulas of first-order logic extended with arithmetic and aggregate operators. The method allows choice of particular CNF representations of complex constraints to be parameterized easily. We have implemented the methods we describe, and demonstrated that they can be effective in practice.

1 Introduction

An important direction of work in constraint-based methods is the development of declarative languages for specifying or modelling combinatorial search problems. These languages provide users with a notation in which to give a high-level specification of a problem (see e.g. ESSENCE [1]). By reducing the need for specialized constraint programming knowledge, these languages make the technology accessible to a wider variety of users. In our group, a logic-based framework for specification/modelling language was proposed [2]. We undertake a research program of both theoretical development and demonstrating practical feasibility through system development.

Our tools are based on *grounding*, which is the task of taking a problem specification, together with an instance, and producing a variable-free first-order formula representing the solutions to the instance[1]. Here, we consider grounding to propositional logic, with the aim of using SAT solvers as the problem solving engine. Note that SAT is just one possibility. A similar process can be used for grounding from a high-level language to e.g. cplex, various SMT and ground constraint solvers, e.g. MINION [3], etc., which is a subject of future work. An important advantage in solving through grounding is that the speed of ground solvers improves all the time, and we can always use the best and the latest solver available.

Grounding a first-order formula over a given finite domain A may be done simply by replacing $\forall x\ \phi(x)$ with $\wedge_{a\in A}\phi(x)[x/\tilde{a}]$, and $\exists x\ \phi(x)$ with $\vee_{a\in A}\phi(x)[x/\tilde{a}]$ where $\tilde{a}$ is a new constant symbol denoting domain element a. In practice, though, effective

[1] By instance we always understand an instance of a search problem, e.g. a graph is an instance of 3-colourability.

grounding is not easy. Naive methods are too slow, and produce groundings that are too large and contain many redundant clauses.

One advantage of using a language based on classical logic is that first-order logic (FO) is equivalent to relational algebra, and therefore we could adapt database query optimization techniques for grounding. Our group is developing grounding technology based on a generalization of the relational algebra. Since correctness and completeness are assured by algebraic properties, it is practical to employ a wider range of algorithmic paradigms that is apparent with direct grounding methods. The basic method, for function-free FO, is defined in [4, 5], and a prototype implementation is described in [5]. A variety of extensions and refinements of the method are under development, including adaptation of database query techniques to increase performance. It is important to notice that we are solving a model expansion[5] problem which is very different from query evaluation process. In model expansion context, there are formulas and subformulas for which we do not know if they are true or false, while in query processing context, every formula can be evaluated as either true or false.

While there are many advantages of choosing mathematical logic as a foundation for a modelling language, there seem to be many obstacles for developing realistic modelling languages with practical features which are not a part of classical logic, e.g. for modelling languages which have arithmetic constructs and aggregate operations (such as Count, Max, Sum, etc.). In [6], the framework [2] was extended with arithmetic and aggregates. Here, we demonstrate the practicality of the approach by extending our relational algebra based algorithms to deal with arithmetic and aggregates.

An important element in the practice of SAT solving is the choice, when designing reductions, of "good" encodings into propositional logic of complex constraints. We describe our method for grounding of formulas containing aggregate operations in terms of "gadgets" which determine the actual encoding. The choice of the particular gadget can be under user control, or even made automatically at run time based on formula and instance properties. To illustrate, we describe three gadgets for the count operator.

Even within one specification, different occurrences of the same aggregate may be grounded differently, and this may vary from instance to instance. With well designed (possibly by machine learning methods) heuristics for such choices, we may be able to produce groundings that are more effective in practice than those a human could design by hand, except through an exceedingly labour-intensive process.

Before proceeding, let us remark that the point of this work does not hinge on whether the best way to handle certain aggregates is by reduction to SAT or natively. Whenever high-level specifications are used together with high-performance solvers, there will be constraints in specifications that are not handled natively by the solvers. This work is part of a research program to develop effective techniques for this purpose.

Our main contributions are:

1. We presented an algorithm which can be used to ground specifications having different kinds of terms, e.g., aggregates, expansion/instance functions, arithmetics.
2. We enriched our language with aggregates, designed and developed an engine which can convert the these constructs to CNF. To do so, we have used some oracles which we call them gadgets.

3. We defined the notion of answer to terms and modify the previous grounding algorithm to be able to work with this new concept.

2 Background

We formalize combinatorial search problems in terms of the logical problem of *model expansion (MX)*, defined here for an arbitrary logic $\mathcal{L}$.

Definition 1 (MX). *Given an $\mathcal{L}$-sentence ϕ, over the union of disjoint vocabularies σ and ε, and a finite structure $\mathcal{A}$ for vocabulary σ, find a structure $\mathcal{B}$ that is an expansion of $\mathcal{A}$ to $\sigma \cup \varepsilon$ such that $\mathcal{B} \models \phi$.*

In this paper, ϕ is a problem specification formula, and is fixed for each search problem. $\mathcal{A}$ always denotes a finite σ-structure, called the instance structure, σ is the instance vocabulary, and ε the expansion vocabulary.

Example 1. The following formula ϕ of first order logic constitutes a specification for Graph 3-Colouring:

$$\begin{array}{l}\forall x\ [\big(R(x) \vee B(x) \vee G(x)\big)]\ \wedge \\ \forall x\ \neg[(R(x) \wedge B(x)) \vee (R(x) \wedge G(x)) \vee (B(x) \wedge G(x))]\ \wedge \\ \forall x \forall y\ [(E(x,y) \vee E(y,x)) \supset (\neg(R(x) \wedge R(y)) \wedge \neg(B(x) \wedge B(y)) \wedge \neg(G(x) \wedge G(y)))]\end{array}$$

An instance is a structure for vocabulary $\sigma = \{E\}$, i.e., a graph $\mathcal{A} = \mathcal{G} = (V; E)$. The task is to find an expansion $\mathcal{B}$ of $\mathcal{A}$ that satisfies ϕ:

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

Interpretations of the expansion vocabulary $\varepsilon = \{R, B, G\}$, for structures $\mathcal{B}$ that satisfy ϕ, are proper 3-colourings of $\mathcal{G}$.

The grounding task is to produce a ground formula $\psi = Gnd(\phi, \mathcal{A})$, such that models of ψ correspond to solutions for instance $\mathcal{A}$. Formally, to ground we bring domain elements into the syntax by expanding the vocabulary with a new constant symbol for each element of the domain. For domain A, the domain of structure $\mathcal{A}$, we denote the set of such constants by $\tilde{A}$. In practice, the ground formula should contain no occurrences of the instance vocabulary, in which case we call it reduced.

Definition 2 (Reduced Grounding for MX). *Formula ψ is a* reduced grounding *of formula ϕ over σ-structure $\mathcal{A} = (A; \sigma^{\mathcal{A}})$ if*
1) ψ is a ground formula over $\varepsilon \cup \tilde{A}$, and
2) for every expansion structure $\mathcal{B} = (A; \sigma^{\mathcal{A}}, \varepsilon^{\mathcal{B}})$ over $\sigma \cup \varepsilon$, $\mathcal{B} \models \phi$ iff $(\mathcal{B}, \tilde{A}^{\mathcal{B}}) \models \psi$, where $\tilde{A}^{\mathcal{B}}$ is the standard interpretation of the new constants $\tilde{A}$.

Proposition 1. *Let ψ be a reduced grounding of ϕ over σ-structure $\mathcal{A}$. Then $\mathcal{A}$ can be expanded to a model of ϕ iff ψ is satisfiable.*

Producing a reduced grounding with respect to a given structure $\mathcal{A}$ can be done by an algorithm that, for each fixed FO formula, runs in time polynomial in the size of $\mathcal{A}$. Such a grounding algorithm implements a polytime reduction to SAT for each NP search problem. Simple grounding algorithms, however, do not reliably produce groundings for large instances of interesting problems fast enough in practice.

Grounding for MX is a generalization of query answering. Given a structure (database) $\mathcal{A}$, a boolean query is a formula ϕ over the vocabulary of $\mathcal{A}$, and query answering is equivalent to evaluating whether ϕ is true, i.e., $\mathcal{A} \models \phi$. For model expansion, 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 possible expansions of $\mathcal{A}$ for which ϕ is true.

The grounding algorithms in this paper construct a grounding by a bottom-up process that parallels database query evaluation, based on an extension of the relational algebra. For each sub-formula $\phi(\bar{x})$ with free variables $\bar{x}$, we call the set of reduced groundings for ϕ under all possible ground instantiations of $\bar{x}$ *an answer to* $\phi(\bar{x})$. We represent answers with tables on which an extended algebra operates.

An X-relation is a k-ary relation associated with a k-tuple of variables X, representing a set of instantiations of the variables of X. It is a central notion in databases. In extended X-relations, introduced in [7], each tuple γ is associated with a formula ψ. For convenience, we use $\top$ and $\bot$ as propositional formulas which are always true and, respectively, false.

Definition 3 (extended X-relation; function $\delta_{\mathcal{R}}$). *Let A be a domain, and X a tuple of variables with $|X| = k$. An* extended X-relation $\mathcal{R}$ *over A is a set of pairs (γ, ψ) s.t.*
1) $\gamma : X \to A$, and
2) ψ is a formula, and
3) if $(\gamma, \psi) \in \mathcal{R}$ and $(\gamma, \psi') \in \mathcal{R}$ then $\psi = \psi'$.
The function $\delta_{\mathcal{R}}$ represented by $\mathcal{R}$ *is a mapping from k-tuples γ of elements of the domain A to formulas, defined by:*

$$\delta_{\mathcal{R}}(\gamma) = \begin{cases} \psi & \text{if } (\gamma, \psi) \in \mathcal{R}, \\ \bot & \text{if there is no pair } (\gamma, \psi) \in \mathcal{R}. \end{cases}$$

For brevity, we sometimes write $\gamma \in \mathcal{R}$ to mean that there exists ψ such that $(\gamma, \psi) \in \mathcal{R}$. We also sometimes call extended X-relations simply tables. To refer to X-relations for some concrete set X of variables, rather than in general, we write X-relation.

Definition 4 (answer to ϕ wrt $\mathcal{A}$). *Let ϕ be a formula in $\sigma \cup \varepsilon$ with free variables X, $\mathcal{A}$ a σ-structure with domain A, and $\mathcal{R}$ an extended X-relation over $\mathcal{A}$. We say $\mathcal{R}$ is* an answer to ϕ wrt $\mathcal{A}$ *if for any $\gamma : X \to A$, we have that $\delta_{\mathcal{R}}(\gamma)$ is a reduced grounding of $\phi[\gamma]$ over $\mathcal{A}$. Here, $\phi[\gamma]$ denotes the result of instantiating free variables in ϕ according to γ.*

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.

Example 2. 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 $\phi_1 \equiv P(x,y,z) \wedge E(x,y) \wedge E(y,z)$:

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 $\phi_1[(1,2,3)] = P(1,2,3) \wedge E(1,2) \wedge E(2,3)$, and $\delta_{\mathcal{R}}(1,1,1) = \bot$ is a reduced grounding of $\phi_1[(1,1,1)]$. The following extended relation is an answer to $\phi_2 \equiv \exists z \phi_1$:

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 $\phi_2[(1,2)]$. Finally, the following represents an answer to $\phi_3 \equiv \exists x \exists y \phi_2$, where the single formula is a reduced grounding of ϕ_3.

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

The relational algebra has operations corresponding to each connective and quantifier in FO, as follows: complement (negation); join (conjunction); union (disjunction), projection (existential quantification); division or quotient (universal quantification). Following [7, 5], we generalize each to extended X-relations as follows.

Definition 5 (Extended Relational Algebra). *Let $\mathcal{R}$ be an extended X-relation and $\mathcal{S}$ an extended Y-relation, both over domain A.*

1. *$\neg\mathcal{R}$ is the extended X-relation $\neg\mathcal{R} = \{(\gamma,\psi) \mid \gamma : X \to A, \delta_{\mathcal{R}}(\gamma) \neq \top,$ and $\psi = \neg\delta_{\mathcal{R}}(\gamma)\}$*
2. *$\mathcal{R} \bowtie \mathcal{S}$ is the extended $X \cup Y$-relation $\mathcal{R} \bowtie \mathcal{S} = \{(\gamma,\psi) \mid \gamma : X \cup Y \to A, \gamma|_X \in \mathcal{R}, \gamma|_Y \in \mathcal{S},$ and $\psi = \delta_{\mathcal{R}}(\gamma|_X) \wedge \delta_{\mathcal{S}}(\gamma|_Y)\}$;*
3. *$\mathcal{R} \cup \mathcal{S}$ is the extended $X \cup Y$-relation $\mathcal{R} \cup \mathcal{S} = \{(\gamma,\psi) \mid \gamma|_X \in \mathcal{R}$ or $\gamma|_Y \in \mathcal{S},$ and $\psi = \delta_{\mathcal{R}}(\gamma|_X) \vee \delta_{\mathcal{S}}(\gamma|_Y)\}$.*
4. *For $Z \subseteq X$, the Z-*projection *of $\mathcal{R}$, denoted by $\pi_Z(\mathcal{R})$, is the extended Z-relation $\{(\gamma',\psi) \mid \gamma' = \gamma|_Z$ for some $\gamma \in \mathcal{R}$ and $\psi = \bigvee_{\{\gamma \in \mathcal{R} \mid \gamma' = \gamma|_Z\}} \delta_{\mathcal{R}}(\gamma)\}$.*
5. *For $Z \subseteq X$, the Z-*quotient *of $\mathcal{R}$, denoted by $d_Z(\mathcal{R})$, is the extended Z-relation $\{(\gamma',\psi) \mid \forall\gamma(\gamma : X \to A \wedge \gamma|_Z = \gamma' \Rightarrow \gamma \in \mathcal{R}),$ and $\psi = \bigwedge_{\{\gamma \in \mathcal{R} \mid \gamma' = \gamma|_Z\}} \delta_{\mathcal{R}}(\gamma)\}$.*

To ground using this algebra, we apply the algebra inductively on the structure of the formula, just as the standard relational algebra may be applied for query evaluation. We define the answer to atomic formula $P(\bar{x})$ as follows. If P is an instance predicate, the answer to P is the set of tuples $(\bar{a}, \top)$, for $\bar{a} \in P^{\mathcal{A}}$. If P is an expansion predicate, the answer is the set of all pairs $(\bar{a}, P(\bar{a}))$, where $\bar{a}$ is a tuple of elements from the domain A. Correctness of the method then follows, by induction on the structure of the formula, from the following proposition.

Proposition 2. *Suppose that $\mathcal{R}$ is an answer to ϕ_1 and $\mathcal{S}$ is an answer to ϕ_2, both with respect to (wrt) structure $\mathcal{A}$. Then*

1. *$\neg\mathcal{R}$ is an answer to $\neg\phi_1$ wrt $\mathcal{A}$;*
2. *$\mathcal{R} \bowtie \mathcal{S}$ is an answer to $\phi_1 \wedge \phi_2$ wrt $\mathcal{A}$;*
3. *$\mathcal{R} \cup \mathcal{S}$ is an answer to $\phi_1 \vee \phi_2$ wrt $\mathcal{A}$;*
4. *If Y is the set of free variables of $\exists\bar{z}\phi_1$, then $\pi_Y(\mathcal{R})$ is an answer to $\exists\bar{z}\phi_1$ wrt $\mathcal{A}$.*
5. *If Y is the set of free variables of $\forall\bar{z}\phi_1$, then $d_Y(\mathcal{R})$ is an answer to $\forall\bar{z}\phi_1$ wrt $\mathcal{A}$.*

The straightforward proof for cases 1, 2 and 4 is given in [7]; the other cases follow easily.

The answer to an atomic formula $P(\bar{x})$, where P is from the expansion vocabulary, is formally a universal table, but in practice we may represent this table implicitly and avoid explicitly enumerating the tuples. As operations are applied, some subset of columns remain universal, while others do not. Again, those columns which are universal may be represented implicitly. This could be treated as an implementation detail, but the use of such implicit representations dramatically affects the cost of operations, and so it is useful to further generalize our extended X-relations. Following [5], we call the variables which are implicitly universal "hidden" variables, as they are not represented explicitly in the tuples, and the other variables "explicit" variables.

Definition 6 (Extended Hidden X-Relation $\mathcal{R}_Y$; $\delta_{\mathcal{R}_Y}$). *Let X, Y be tuples of variables, with $Y \subseteq X$ (when viewed as sets), and $|X| = k$. An extended hidden X-relation $\mathcal{R}_Y$ is a set of tuples (γ, ψ) s.t.*
1) $\gamma : X\backslash Y \to A$, and
2) ψ is a formula, and
3) if $(\gamma, \psi) \in \mathcal{R}_Y$ and $(\gamma, \psi') \in \mathcal{R}_Y$, then $\psi = \psi'$.
The function $\delta_{\mathcal{R}_Y}$ represented by $\mathcal{R}_Y$ is a mapping from k-tuples γ' of elements of the domain A to formulas, defined by

$$\delta_{\mathcal{R}_Y}(\gamma') = \begin{cases} \psi & \text{if } (\gamma' \mid_{X\backslash Y}, \psi) \in \mathcal{R}, \\ \bot & \text{if there is no pair } (\gamma' \mid_{X\backslash Y}, \psi) \in \mathcal{R}. \end{cases}$$

So, an extended hidden X-relation $\mathcal{R}_Y$ is a compact representation of an extended X-relation by an extended $X\backslash Y$-relation, which may be used whenever the columns for variables of Y are universal. If $X = Y$, we have a compact representation of a universal relation; if $Y = \emptyset$, we have a normal extended X-relation.

All the operations of the algebra generalize easily. The hidden variables technique does not alter the semantics of the operations. Henceforth, the term table may denote either an extended X-relation or a hidden extended X-relation.

2.1 FO MX with Arithmetic

In this paper, we are concerned with specifications written in FO extended with arithmetic and aggregate operators. Informally, we assume that the domain of any instance structure is $\mathbb{N}$, and that arithmetic operators have their standard meanings. Details of aggregate operators need to be specified, but these also behave according to our normal

intuitions. Quantified variables and the range of instance functions must be restricted to finite subsets of the integers, and possible interpretations of expansion predicates and expansion functions must be restricted to a finite domain of $\mathbb{N}$, as well. This can be done by employing a multi-sorted logic in which all sorts are required to be finite subsets of $\mathbb{N}$, or by requiring specification formulas to be written in a certain "guarded" form.

In [6], these intuitions are formalized using the concepts of embedded structures and double-guarded logics. There, it is shown that MX specifications in FO extended with arithmetic and aggregates, as used in the present paper, can express exactly the problems in NP restricted to "small cost" structures, in which numbers are restricted to not be extremely large. Exact details are unimportant for this paper. In the rest of this paper, we assume that all variables are ranging over the finite domain[2] $T \subset \mathbb{N}$ and $\phi(t_1(\bar{x}), \cdots, t_k(\bar{x}))$ is a short-hand for

$$\exists y_1, \cdots, y_k : y_1 = t_1(\bar{x}) \wedge \cdots y_k = t_k(\bar{x}) \wedge \phi(y_1, \cdots, y_k)$$

Under these assumptions, we do not need to worry about the interpretation of predicates and functions outside T.

Syntax and Semantics of Aggregate Operators In this section, we may use evaluation for formulas with expansion predicates. What we mean by evaluating a formula, which has expansion predicates, as true is that there is a solution for the whole specification which satisfies the given formula, too. Also, for sake of representation, we may use $\phi[\bar{a}, \bar{z_2}]$ as a short-hand for $\phi(\bar{z_1}, \bar{z_2})[\bar{z_1}/\bar{a}]$. We defined the following aggregates terms in this paper:

- $Count_{\bar{x}}\{\phi(\bar{x}, \bar{y})\}$, for any instantiation $\bar{b}$ for $\bar{y}$, denotes the number of tuples $\bar{a}$ for which $\phi[\bar{a}, \bar{b}]$ is true;
- $Max_{\bar{x}}\{t(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y}); d_M\}$, for any instantiation $\bar{b}$ for $\bar{y}$, denotes the maximum value obtained by $t[\bar{a}, \bar{b}]$ over all instantiations $\bar{a}$ for $\bar{x}$ for which $\phi[\bar{a}, \bar{b}]$ is true, or d if there are none. In fact, d_M is the default value of Max aggregate which is returned whenever all conditions are evaluated as false. We have implemented a more general form of Max aggregate in which the default value can be a general term, but for sake of explanation, we present a special case where the default value is a constant.
- $Min_{\bar{x}}\{t(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y}); d_m\}$ is defined dually to Max.
- $Sum_{\bar{x}}\{t(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y})\}$, for any instantiation $\bar{b}$ of $\bar{y}$, denotes 0 plus the sum of all values $t[\bar{a}, \bar{b}]$ across all instantiations $\bar{a}$ for $\bar{x}$ for which $\phi[\bar{a}, \bar{b}]$ is true.

We tried to define the semantics for aggregates very similar to what we have in SQL, but SQL has a special value, *NULL*, which cannot be modelled in propositional logic. The default value in Min and Max aggregates is used to make all terms/functions total.

Throughout the paper, we use σ, ε and ν, to denote instance, expansion vocabularies, and the vocabulary of the "background structure" of the natural numbers, respectively. Here, we assume the background structure consists of $\{=, <, +, -, \times, Count, Max, Min, Sum\}$ and a constant symbol for each natural number.

[2] A more general version, where each variable may have its own domain, is implemented, but is more complex to explain.

3 Evaluating Out Arithmetic and Instance Functions

The relational algebra-based grounding algorithm, described in Section 2, is designed for the relational (function-free) case. Below, we extend it to the case where arguments to atomic formulas may be complex terms. In this section, we present a simple method for the special case where functions do not contain expansion predicates, so can be evaluated purely in terms of the instance structure.

Recall that an answer to a sub-formula $\phi(X)$ of a specification is an extended X-relation R. If $|X| = k$, then the tuples of R have arity k. Now, consider an atomic formula whose arguments are terms containing instance functions and arithmetic operations, for example $P(x + y)$. This formula is equivalent to the formula $\phi = \exists z(z = x+y \wedge P(z))$. Although we have not discussed handling of the sub-formula $z = x+y$, it is apparent that the answer to ϕ, which has free variables $\{x, y\}$, is an extended $\{x, y\}$-relation R.

One possibility for R is the set of all pairs $(\langle a, b\rangle, \psi)$ such that $a + b$ is in the interpretation of P. To modify the grounding algorithm of previous sub-section, we revise the base cases of definition as follows.

Definition 7 (Base Cases for Atoms with Evaluated Terms). *For an atomic formula $\phi = P(t_1, \cdots, t_n)$ with terms $t_1 \ldots t_n$ and free variables X, use the following extended X-relation (which is an answer to ϕ wrt $\mathcal{A}$):*

1. *if P is an instance predicate, $\{(\gamma, \top) \mid \mathcal{A} \models P(t_1, \ldots t_n)[\gamma]\}$*
2. *if P is $t_1(\bar{x}) \odot t_2(\bar{x})$, where $\odot \in \{=, <\}$, $\{(\gamma, \top) \mid \mathcal{A} \models t_1 \odot t_2[\gamma]\}$*
3. *if P is an expansion predicate, $\{(\gamma, P(a_1, \ldots a_n)) \mid \mathcal{A} \models (t_1 = a_1, \ldots t_n = a_n)[\gamma]\}$*

Terms involving aggregate operators, provided the formula argument to that operator contains only instance predicates and functions with a given interpretation, can also be evaluated out in this way. The base case from the definition above is implemented and can be used in the absence of aggregates which contain expansion predicates.

4 Answers to Terms

Terms involving expansion functions or predicates, including aggregate terms involving expansion predicates, can only be evaluated with respect to a particular interpretation of those expansion predicates. Thus, they cannot be evaluated out during grounding as in Section 3 and must be represented in the ground formula.

In this section, we further extend the base cases of our relational algebra based grounding method to handle atomic formulas with terms that cannot be evaluated out. The key idea is to introduce a notion of an answer to a term. The new base cases then construct an answer to the atom from the answers to the terms which are its arguments. The terms we allow here include arithmetic expressions, expansion functions, expansion predicates, and aggregate operators involving these as well. As an example for aggregates, in N-Queen problem, the condition $\text{Max}_r\{\text{Count}_c\{Q(r, c)\} : \top; 0\} \leq 1$ expresses that there should be at most one queen in each row.

It will be convenient to use the following representation of extended relations. Let t be a term over $\sigma \cup \varepsilon \cup \nu$ with free variables X, and $\mathcal{A}$ a σ-structure. Let $\mathcal{R}$ be a pair $(\alpha_{\mathcal{R}}, \beta_{\mathcal{R}})$ such that $\alpha_{\mathcal{R}}$ is a finite subset of $\mathbb{N}$, and $\beta_{\mathcal{R}}$ is a function mapping each element $a \in \alpha_{\mathcal{R}}$ to an extended X-relation $\beta_{\mathcal{R}}(a)$.

Intuitively, $\alpha_{\mathcal{R}}$ is the set of all possible values that a given term $t(\bar{X})$ may take, $\beta_{\mathcal{R}}(a)$ is table representing all instantiations of t under which it evaluates to a. We sometimes use $\mathcal{R}_a$ as a shorthand to $\beta_{\mathcal{R}}(a)$. We define $\beta_{\mathcal{R}}(a) = \emptyset$ for $a \notin \alpha_{\mathcal{R}}$. Recall that we defined $\delta_{\mathcal{R}}(\gamma)$ to be a iff $(\gamma, a) \in \mathcal{R}$. We may also use $\delta_R(\gamma, n)$ and $\delta_{\beta_{\mathcal{R}}(n)}(\gamma)$ interchangeably.

Definition 8 (Answer to term t wrt $\mathcal{A}$). *We say that $\mathcal{R} = (\alpha_{\mathcal{R}}, \beta_{\mathcal{R}})$ is an answer to term t wrt $\mathcal{A}$ if, for every $a \in \alpha_R$, the extended X-relation $\beta_{\mathcal{R}}(a)$ is an answer to the formula $(t = a)$ wrt $\mathcal{A}$, and for every $a \notin \alpha_R$, the formula $(t = a)$ is not satisfiable wrt $\mathcal{A}$.*

Example 3. Let $\psi(x, y) = P(x + f(y))$ where domain of both x and y are $\{0, .., 2\}$, $\sigma = \{P, f\}$. Let $\mathcal{A}$ be a σ-structure with $P^A = \{1\}$ and $f^A = \{(0 \mapsto 1), (1 \mapsto 0), (2 \mapsto 1)\}$. Let $t_1 = x$, $t_2 = y$, $t_3 = f(t_2)$ and $t_4 = t_1 + t_3$ be the terms in ψ, and $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$ and $\mathcal{R}_4$ be answers to these terms, respectively. Then, it is easy to verify that $\alpha_{\mathcal{R}_1} = \alpha_{\mathcal{R}_2} = \{0..2\}$, $\alpha_{\mathcal{R}_3} = \{0..1\}$ and $\alpha_{\mathcal{R}_4} = \{0..3\}$. The relations $\beta_{\mathcal{R}_i}$ can be computed using Proposition 3, as demonstrated in example 4.

We now give properties that are sufficient for particular extended X-relations to constitute answers to particular terms. For a tuple X of variables of arity k, define $dom(X)$ to be the set of all k-tuples of domain elements, i.e., $dom(X) = A^k$.

Proposition 3 (Answers to Terms). *Let $\mathcal{R}$ be the pair $(\alpha_{\mathcal{R}}, \beta_{\mathcal{R}})$, and t a term over $\sigma \cup \epsilon \cup \nu$. Assume that $t_1, \ldots t_m$ are terms, and $\mathcal{R}_1, \ldots \mathcal{R}_m$ (respectively) are answers to those terms wrt $\mathcal{A}$. Then $\mathcal{R}$ is an answer to t wrt $\mathcal{A}$ if:*

(1) t is a simple term (i.e., involves only variables, instance functions, and arithmetic operators) and $\alpha_{\mathcal{R}} = \{n \in \mathbb{N} \mid \exists\, \overline{a} \in dom(X) : (t[\overline{a}] = n)\}$ and for all $n \in \alpha_{\mathcal{R}}$, $\beta_{\mathcal{R}}(n)$ is an answer to $t = n$ computed as described in Definition (7).

(2) t is a complex term in form of $t_1 + t_2$ and $\alpha_{\mathcal{R}} = \{x + y \mid x \in \alpha_{\mathcal{R}_1}$ and $y \in \alpha_{\mathcal{R}_2}\}$

$$\beta_{\mathcal{R}}(n) = \cup_{(j \in \alpha_{\mathcal{R}_1},\ k \in \alpha_{\mathcal{R}_2},\ n = j+k)} \beta_{\mathcal{R}_1}(j) \bowtie \beta_{\mathcal{R}_2}(k)$$

(3) t is a complex term in form of $t_1 - t_2$ — similar to case (2);

(4) t is a complex term in form of $t_1 \times t_2$ — similar to case (2);

(5) t is a complex term in form of $f(t_1, \cdots, t_m)$, where f is an instance function, and $\alpha_{\mathcal{R}} = \{y |$ for some $x_1 \in \alpha_{\mathcal{R}_1}, \ldots, x_m \in \alpha_{\mathcal{R}_m},\ f(x_1, \ldots, x_m) = y\}$,

$$\beta_{\mathcal{R}}(n) = \cup_{a_1 \in \alpha_{\mathcal{R}_1}, \ldots a_m \in \alpha_{\mathcal{R}_m},\ s.t. f(a_1, \ldots a_m) = n} \beta_{\mathcal{R}_1}(a_1) \bowtie \cdots \bowtie \beta_{\mathcal{R}_m}(a_m)$$

(6) t is a complex term in form of $f(t_1, \cdots, t_m)$, where f is an expansion function, and $\alpha_{\mathcal{R}}$ is equal to range of (f),

$$\beta_{\mathcal{R}}(n) = \cup_{a_1 \in \alpha_{\mathcal{R}_1}, \ldots a_m \in \alpha_{\mathcal{R}_m}} \beta_{\mathcal{R}_1}(a_1) \bowtie \cdots \bowtie \beta_{\mathcal{R}_m}(a_m) \bowtie T(a_1, \cdots, a_m)$$

Where $T(a_1, \cdots, a_m)$ is an answer to $\exists x_1, \cdots, x_m : (x_1 = a_1) \wedge \cdots \wedge (x_m = a_m) \wedge f(x_1, \cdots, x_m) = n$.

(7) t *is* $\textit{Count}_{\bar{x}}\{\phi(\bar{x},\bar{y})\}$ *and* $\alpha_{\mathcal{R}} = \{n \mid 0 \leq n \leq |dom(\bar{x})|\}$, $\beta_{\mathcal{R}}(n)$ *is an answer to*

$$\bigvee_{S\subseteq dom(\bar{x}),\ |S|=n} \Big(\bigwedge_{\bar{a}\in S} \phi(\bar{x},\bar{y})[\bar{x}/\bar{a}] \wedge \bigwedge_{\bar{a}\in dom(\bar{x})\setminus S} \neg\phi(\bar{x},\bar{y})[\bar{x}/\bar{a}]\Big)$$

Intuitively, the above formula asserts that there should be exactly n *different instantiations for* $\bar{x}$ *such that* $\phi(\bar{x},\bar{y})$ *is evaluated as true.*

(8) t *is* $\textit{Max}_{\bar{x}}\{t_1(\bar{x},\bar{y}) : \phi(\bar{x},\bar{y}); d_M)\}$ *and* $\alpha_{\mathcal{R}} = \alpha_{\mathcal{R}_1} \cup \{d_M\}$, $\beta_{\mathcal{R}}(n)$ *for every* $n \in \alpha_{\mathcal{R}}$ *is an answer to*

$$\Big(\big(\exists\bar{x}\, t_1(\bar{x},\bar{y}) = n \wedge \phi(\bar{x},\bar{y})\big) \wedge \forall\bar{x}(\phi(\bar{x},\bar{y}) \Rightarrow t_1(\bar{x},\bar{y}) \leq n)\Big) \vee \Big(n = d_M \wedge \forall\bar{x}\, \neg\phi(\bar{x},\bar{y})\Big)$$

Intuitively, the value of Max aggregate, for a fixed instantiation of $\bar{y}$*, is* n *iff either there is at least one instantiation for* $\bar{x}$ *such that* $\phi(\bar{x},\bar{y})$ *is evaluated as true and among those* $\bar{x}$*, the maximum value of* $t_1(\bar{x},\bar{y})$ *is equal to* n *or there is no* $\bar{x}$ *such that* $\phi(\bar{x},\bar{y})$ *is evaluated as true and the aggregate's default value,* d_M*, is equal to* n*.*

(9) t *is* $\textit{Min}_{\bar{x}}\{t_1(\bar{x},\bar{y}) : \phi(\bar{x},\bar{y}); d_m)\}$ *— similar to case (8)*

(10) t *is* $\textit{Sum}_{\bar{x}}\{t_1(\bar{x},\bar{y}) : \phi(\bar{x},\bar{y})\}$ *and* $\alpha_{\mathcal{R}} = \{\sum_{x\in T} x \mid T \subseteq \alpha_{\mathcal{R}_1}\}$ *and* $\beta_{\mathcal{R}}(n)$ *for every* $n \in \alpha_{\mathcal{R}}$ *is an answer to:*

$$\bigvee_{\substack{T\subseteq dom(\bar{x}),\\ m\mid\ T\to\alpha_{\mathcal{R}},\\ \sum_{\bar{a}\in T} m(\bar{a})=n}} \Big(\bigwedge_{\bar{a}\in T} \big(t_1(\bar{a},\bar{y}) = m(\bar{a}) \wedge \phi(\bar{x},\bar{y})[\bar{x}/\bar{a}]\big) \wedge \bigwedge_{\bar{a}\in dom(\bar{x})\setminus T} \neg\phi(\bar{x},\bar{y})[\bar{x}/\bar{a}]\Big)$$

Remark 1. Although properties of answers for aggregate terms given in Proposition (3) can be seen as constructions too, they do not demonstrate the actual constructions. We presented them to define the properties of the relation corresponding to $\beta_R(.)$, the actual constructions are described in section 5.

Example 4. (Continue From Example 3) $\beta_{\mathcal{R}_1}$ and $\beta_{\mathcal{R}_2}$ correspond to the answer to variables x and y. So, each of them should have one free variable. As these two relations are very similar, we only demonstrate $\beta_{\mathcal{R}_1}$. Having an answer to t_2, an answer to t_3, $(\alpha_{\mathcal{R}_3}, \beta_{\mathcal{R}_3})$, can be computed. By proposition 3, we have $\beta_{\mathcal{R}_4}(1) = \beta_{\mathcal{R}_1}(0) \bowtie \beta_{\mathcal{R}_3}(1) \cup \beta_{\mathcal{R}_1}(1) \bowtie \beta_{\mathcal{R}_3}(0)$. In other word, the answer to t_4 is 1 if either $t_1 = 0 \wedge t_3 = 1$ or $t_1 = 1 \wedge t_3 = 0$.

4.1 Base Case for Unevaluated Terms

To extend our grounding algorithm to handle terms which cannot be evaluated out as in Section 3, we add the following base cases to the algorithm.

Definition 9 (Base Case for Atoms with Unevaluated Terms). *Let* $t_1, \cdots, t_m$ *be terms, and assume that* $\mathcal{R}_1, \ldots \mathcal{R}_m$ *(respectively) are answers to those terms wrt structure* $\mathcal{A}$*. Then* $\mathcal{R}$ *is an answer to* $P(t_1, \ldots t_m)$ *wrt* $\mathcal{A}$ *if*

1. $P(.,.)$ *is* $t_1 = t_2$ *and* $\mathcal{R} = \cup_{(i\in\alpha_{\mathcal{R}_1}\cap\alpha_{\mathcal{R}_2})} \beta_{\mathcal{R}_1}(i) \bowtie \beta_{\mathcal{R}_2}(i)$
2. $P(.,.)$ *is* $t_1 \leq t_2$ *and* $\mathcal{R} = \cup_{(i\in\alpha_{\mathcal{R}_1},\ j\in\alpha_{\mathcal{R}_2},\ i\leq j)} \beta_{\mathcal{R}_1}(i) \bowtie \beta_{\mathcal{R}_2}(j)$

(a)

x	ϕ
0	True
1	False
2	False

(b)

x	ϕ
0	False
1	True
2	False

(c)

x	ϕ
0	False
1	False
2	True

(d)

y	ϕ
0	False
1	True
2	False

(e)

y	ϕ
0	True
1	False
2	True

(f)

x	y	ϕ
0	0	True
0	1	False
0	2	False
1	0	False
1	1	True
1	2	False
2	⋮	False

Table 1. Tables for Example 4: a) Answer to $x = 0$,i.e, $\beta_{\mathcal{R}_1}(0)$, b) Answer to $x = 1$,i.e, $\beta_{\mathcal{R}_1}(1)$, c) Answer to $x = 2$,i.e, $\beta_{\mathcal{R}_1}(2)$, d) Answer to $f(y) = 0$,i.e, $\beta_{\mathcal{R}_3}(0)$, e) Answer to $f(y) = 1$,i.e, $\beta_{\mathcal{R}_3}(1)$, e) Answer to $x + f(y) = 1$,i.e, $\beta_{\mathcal{R}_4}(1)$,

3. *P is an instance predicate and*

$$\mathcal{R} = \cup_{(a_1,\cdots,a_m)\in P^{\mathcal{A}},\ a_1\in\alpha_{\mathcal{R}_1},\ldots a_m\in\alpha_{\mathcal{R}_m})}\beta_{\mathcal{R}_1}(a_1)\bowtie\cdots\bowtie\beta_{\mathcal{R}_m}(a_m)$$

4. *P is an expansion predicate and $\mathcal{R}$ is an answer to*

$$\exists x_1\ldots x_m\ (x_1 = t_1\wedge\cdots\wedge x_m = t_m\wedge P(x_1,\ldots,x_m))$$

Example 5. (Continue From Example 3) Till now, we have computed an answer to t_4, $(\alpha_{\mathcal{R}_4}, \beta_{\mathcal{R}_4})$. To compute an answer to $\psi(x, y) = P(x + f(y))$, one needs to the union of $\beta_{\mathcal{R}_4}(n)$ for $n \in P^{\mathcal{A}} \cap \alpha_{\mathcal{R}_4}$. In this simple example, $P^{\mathcal{A}} \cap \alpha_{\mathcal{R}_4} = \{1\}$ and therefore $\beta_{\mathcal{R}_4}(1)$ is the answer to ψ, too.

5 Constructing Answers to Aggregates

We could have an implementation which constructs answers to complex terms by taking literally the conditions described in Proposition 3. However, we would expect this implementation to result in a system with poor performance. In the grounding algorithm, the function which generates ψ for tuple (γ, ψ) may produce any formula logically equivalent to ϕ. We may think of the function as "gadgets", in the sense this term is

used in reductions to prove NP-completeness. Choice of these gadgets is important for some constraints. For example, choosing CNF representations of aggregates is an active area of study in the SAT community (e.g., see [8]). Our method allows these choices to be made at run time, either by the user or automatically, without modification to the main grounding machinery.

As it described in previous sections, to compute an answer to an aggregate one needs to find a set $\alpha_{\mathcal{R}} \subset N$ and a function $\beta_{\mathcal{R}}$ which maps every integer to a ground formula. In section 4, we have showed what the set $\alpha_{\mathcal{R}}$ is for each term and also described the properties of the output of $\beta_{\mathcal{R}}$ function. Here, we are going to present some gadgets which can be used as $\beta_{\mathcal{R}}$ for Count, Min, Max and Sum aggregates.

5.1 Gadgets for Count

A gadget for Count aggregate, denoted by $C(S, n)$, takes a set $S = \{f_1, \cdots, f_k\}$ of formulas and a non-negative integer, n, and returns a formula f, such that in every model satisfying f, exactly n of the formulas in S are evaluated as true.

We have implemented gadgets for count aggregate based on several representations that seem useful in practice, and describe two of these here. In [9], we have described six gadgets we have used to translate the Count aggregate into CNF.

BDD Gadget for Count One can use binary decision diagrams (BDD) to construct CNF clauses for cardinality constraints. We adopt the algorithm described in [10]. Each BDD node is an if-then-else gate. So, the constraint $(x_1, \cdots, x_k) = n$ can be described using $(k - n + 1) \times n$ such nodes.

We inductively calculate F_r^c's where F_r^c is a formula satisfying the properties of output of $C(\{x_1 \cdots x_r\}, c)$. The base cases are:

$$F_r^c = \begin{cases} \top & \text{if } r \text{ and } c \text{ are both zero} \\ \bot & r = 0 \text{ and } c > 0 \end{cases}$$

The inductive case of F_{r+1}^{c+1} can be described as follows:

$$F_{r+1}^{c+1} = (F_r^c \wedge x_{r+1}) \vee (F_r^{c+1} \wedge \neg x_{r+1})$$

And the other inductive case is $F_{r+1}^0 = F_r^0 \wedge \neg x_{r+1}$

Proposition 4. *If a structure satisfies F_k^n, it satisfies exactly n formulas out of $\{x_1, \cdots, x_k\}$. So, F_k^n can be used as the output of $C(\{x_1, \cdots, x_k\}, n)$.*

Example 6. Let $x_1 = E_1(1, 2)$, $x_2 = \neg E_1(2, 3)$ and $x_3 = E_2(1)$, an answer to $C(\{x_1, \cdots, x_3\}, 1)$ can be produced using this method:

$$F_3^1 = (x_3 \wedge (\neg x_2 \wedge \neg x_1)) \vee (\neg x_3 \wedge ((x_2 \wedge \neg x_1) \vee (\neg x_2 \wedge x_1)))$$

$$F_3^1 = (E_2(1) \wedge (E_1(2,3) \wedge \neg E_1(1,2))) \vee (\neg E_2(1) \wedge ((\neg E_1(2,3) \wedge \neg E_1(1,2)) \vee (E_1(2,3) \wedge E_1(1,2))))$$

Sorting Network Gadget for Count A sorting network is a circuit with k input wires and k output wires consisting of a set of comparators with two input wires and two output wires. Each output of a comparator is used as an input to another comparator except those used as output wires of the sorting network itself.

If the input to a sorting network contains Z zeros and O ones, the first O outputs of the sorting network are guaranteed to contain one and the rest output wires contain zeros. Therefore, it can be concluded that there are exactly i ones in the input iff the value of i-th output wire is one while the $i+1$-th output wire is zero.

A single comparator element with inputs f_1 and f_2 is defined to be $f_1 \wedge f_2$, for smaller value, and $f_1 \vee f_2$, for larger value.

Proposition 5. *An implementation of counting gadget using BDD's needs $O(nk)$ clauses and $O(nk)$ literals. Such an implementation using bitonic sorting networks needs $O(n \log^2 n)$ clauses and $O(n \log^2 n)$ literals.*

5.2 Gadgets for Max and Min

A gadget for Max aggregate, denoted by $M(T, d_M, n)$, takes a set $T = \{(n_1, f_1), \cdots, (n_k, f_k)\}$ of pairs of integer and formulas and an integer, n, and returns a formula f, such that in every structure satisfying f, $\beta_{\mathcal{R}}(n)$ is satisfied and vice versa. A gadget for Max can be implemented using the following approach. A similar approach is also used for Min.

Let $T = \{(n_1, f_1), \cdots, (n_k, f_k)\}$ and d_M be an integer. Without loss of generality, we can assume that $n_{i+1} > n_i$. Let $F_r = f_r \wedge \bigwedge_{1 \leq i < r} \neg f_i$. Then,

$$M(T, d_M, n) = \begin{cases} F_i & \exists i : (n_i = n) \wedge (n \neq d_M) \\ f_j^{'} \wedge \bigwedge_{1 \leq i \leq k} \neg f_i & \nexists i : (n_i = n) \wedge (n = d_M) \\ F_i \vee \bigwedge_{1 \leq i \leq k} \neg f_i & \exists i : (n_i = n) \wedge (n = d_M) \\ \bot & n \notin \alpha_{\mathcal{R}} \end{cases}$$

Proposition 6. *The implementation for the gadget $M(T, d_M, n)$ given above describes a correct gadget for Max. The CNF produced by this gadget contains, at most, $k+1$ clauses and $\theta(k^2)$ literals.*

A correct gadget for Min can be obtained dually.

5.3 Gadgets for Sum

A gadget for Sum aggregate, denoted by $S(T, n)$, takes a set T of pairs of integer and formula and an integer, n, and returns a formula f, such that in every structure satisfying f, $\beta_{\mathcal{R}}(n)$ is satisfied and vice versa. A gadget for Sum can be implemented using the following approach.

Let $T = \{(n_1, f_1), \cdots, (n_r, f_k)\}$. Define $S(T, n)$ to be F_k^n where F_r^s's are inductively constructed based on the following definitions:

$$F_r^s = \begin{cases} \top & \text{if } r \text{ and } s \text{ are both zero} \\ \bot & r = 0 \text{ and } s \neq 0 \end{cases}$$

$$F_{r+1}^s = (F_r^s \wedge \neg f_{r+1}) \vee (F_r^{s-n_{r+1}} \wedge f_{r+1})$$

Proposition 7. *The implementation for the gadget $S(T, s)$ given above describes a correct gadget for sum aggregate. The CNF generated using this gadget has $O(nk)$ clauses and $O(nk)$ literals.*

6 Complexity of Grounding

For any fixed FO ϕ, grounding with respect to finite structure $\mathcal{A}$ can easily be done in time polynomial in $|A|$. Either direct generation of all instantiations of variables or the algorithm of Section 2 will suffice to do so. (It does not follow that producing a good grounding quickly in practice is easy. Indeed, it is not.) Proving this for the relational algebra based method is a simple structural induction involving the observation that the relational algebra operations (including the versions adapted for extended X relations), are polytime.

Answers to terms are also constructed inductively using operations of the relational algebra. After verifying that the tables for aggregates can be constructed in polynomial time, a similar induction can be used to extend the proof of polytime grounding for relational algebra based grounding to the method described in this paper for formulas with complex terms, for all terms that do not involve the Sum aggregate.

For the sum aggregate, there are cases where any answer constructed according to our definition of an answer to a term must be of exponential size. To see this, consider the following knapsack constraint

$$Sum_o\{w(o) : InKnapsack(o)\} < k,$$

Which states a bound on the sum of the weights of the objects in the knapsack. Assume that InKnapsack is an expansion predicate. There are exponentially many possible choices for the contents of the knapsack (each object o may be in or out), and thus exponentially many possible sums, each of which may be distinct. Since our answer contains a tuple for each possible sum, the answer must be of exponential size. (Notice that, although every relation over A is polysize, the answer to aggregate operations may contain values which are not in A.)

It is worth mentioning that, although in the worst case a grounding algorithm based on our current definition of an answer to a term can be exponential, the method we implement is pseudo-polynomial (much like pseudo-polynomial algorithms used for knapsack and subset sum), and many real world examples can be grounded quickly.

To obtain polysize groundings for a term in which Sum is applied to an expansion predicate, we must encode numbers in binary. To do this within our grounding algorithm, we must modify our definition of an answer to a term, as follows.

(Binary Answers to Terms). Let $t(\overline{x})$ be a term and $\mathcal{A}$ an arithmetical structure. We say $\mathcal{R}$ is a binary answer to t wrt $\mathcal{A}$ iff $\mathcal{R}$ is a collection of extended X-relations $\mathcal{R}_0, \cdots, \mathcal{R}_{n-1}$ where for any of these extended X-relations like $\mathcal{R}_k$ we have that $\mathcal{R}_k$ is an answer to formula

$$\left(t(\overline{x}) \bmod 2^{k+1}\right) \geq 2^k$$

Intuitively, a binary answer to a term is a set of n relations where the k-th relation tells us under what conditions the k-th bit of the result of a term becomes one. With this

approach, we can implement a relational algebra based grounder which is polytime for terms that contain Sum, as well as all other operations mentioned in this paper. Details are left for a future report.

7 Experimental Evaluation

In this section we report some empirical observations on the performance of an implementation of the methods we have described.

We will start by giving the specification of one of our examples and discussing how our empirical results for this specific example can be interpreted.

Example 7. The first specification is as follows:

$$\forall\, x : Max_y(Count_z(E(y,z)); E(x,y); Min_z(z; I(z))) = x$$

where E is an expansion predicate and I an instance predicate. For this example, three different gadgets are used for count-aggregate. The BDD based method, the sorting network method and a merged method which chooses either the output of BDD's or the output of sorting network gadgets. Table (2) shows the number of Tseiten variables generated by our grounder, the number of clauses in the output CNF file, and running time of grounder and SAT solver. The entries in Table (2) are for the case when domain of all variables are from 1 to 100.

Table 2. Statistics of the First Specification

Gadget	# of CNF Variables	# of Clauses	Gnd. Time	SAT Solving Time
BDD	1.5M	5M	24s	3s
Sorting Networks	280K	1.3M	5s	0.6s
Merged	840K	2.9M	13s	1.5s

The results on the rest of examples are given in Table (3). As one might expect, the merged version of counting has a performance in between the two others. This version of counting gadget can then be seen as one which can be used as a default option when users do not have in depth knowledge about their problem.

Examples 2, 3 and 4 in Table (3) have instance domain sizes of 100, 1000 and 100 respectively.

7.1 A Real World Example

As far, we presented our approach to grounding aggregates and arithmetic. As a motivating example, we show how haplotype inference problem[11] can be axiomatized in our grounder. To argue that the CNF generated through our grounder is efficient, we will use a well-known and optimized encoding for haplotype inference problem and show that the same CNF will be obtained without much hardship.

Table 3. Statistics on the Rest of Examples

Example #	Gadget	# of CNF Variables	# of Clauses	Gnd. Time	SAT Time
2	BDD	520K	850K	3.3s	0.7s
2	Sorting Networks	180K	540K	2.1s	0.2s
2	Merged	210K	530K	2s	0.25s
3	BDD	5K	7K	0.8s	0.0s
3	Sorting Network	42K	120K	0.31s	1.1s
3	Merged	5K	7K	0.9s	0s
4	BDD	1.5M	2.5M	1.0s	8.2s
4	Sorting Network	290K	790K	3.1s	32.7s
4	Merged	840K	1.7M	6.9s	35.3s

In haplotype inference problem, we are given an integer r and a set, G, consisting of n strings in $\{0,1,2\}^m$, for a fixed m. We are asked if there exists a set of r strings, H, in $\{0,1\}^m$ such that for every $g \in G$ there are two strings in H which explain g. We say two strings h_1 and h_2 explain an string g iff for every position $1 \leq i \leq m$ either $g[i] = h_1[i] = h_2[i]$ or $g[i] = 2$ and $h_1[i] \neq h_2[i]$.

The following axiomatization is intentionally produced in a way to generate the same CNF encoding as presented in [11] in the assumption that the gadget used for count is a simplified adder circuit [11].

1. $\forall i \forall j\ (g(i,j) = 0 \supset \exists k\ ((\neg h(k,j) \vee \neg S^a(k,i)) \wedge (\neg h(k,j) \vee \neg S^b(k,i))))$
2. $\forall i \forall j\ (g(i,j) = 1 \supset \exists k\ ((h(k,j) \vee \neg S^a(k,i)) \wedge (h(k,j) \vee \neg S^b(k,i))))$
3. $\forall i \forall j\ (ga(i,j) \neq gb(i,j))$
4. $\forall i \forall j \left(g(i,j) = 2 \supset \exists k \left((h(k,j) \vee \neg ga(i,j) \vee \neg S^a(k,i)) \wedge (\neg h(k,j) \vee ga(i,j) \vee \neg S^a(k,i)) \wedge (h(k,j) \vee \neg gb(i,j) \vee \neg S^b(k,i)) \wedge (\neg h(k,j) \vee gb(i,j) \vee \neg S^b(k,i))\right)\right)$
5. $\forall i\ (Count_k(S^a(k,i)) = 1)$
6. $\forall i\ (Count_k(S^b(k,i)) = 1)$

In the above axiomatization, $g(i,j)$ is an instance function which gives the character at position j of i-th string in G. The expansion predicate $h(k,i)$ is true iff the i-th position of the k-th string in H is one. The expansion predicate $S^a(k,i)$ is true iff k-th string in H is one of the explanations for i-th string in G. S^b has a similar meaning. $ga(i,j)$ and $gb(i,j)$ are some peripheral variables which are used in axiom (4).

Table (4) shows the detailed information about running time of haplotype inference instances produced by the ms program[11]. The axiomatization given above corresponds to the row labelled with "Optimized Encoding". The other row labelled with "Basic Encoding" also comes from the same paper [11] but as noted there and shown here produces CNF's that take more time to solve.

So, using Enfragmo as grounder, we have been able to describe the problem in a high level language and yet reproduce the same CNF files that have been obtained through direct reductions. Thus, Enfragmo enables us to try different reductions faster. Of course, once a good reduction is found, one can always use direct reductions to

Table 4. Haplotyping Problem Statistics

	Grounding	SAT Solving	CNF Size
Basic Encoding	2.2 s	12.3 s	18.9 MB
Optimized Encoding	1.9 s	0.95 s	13.3 MB

achieve higher grounding speed although, as table (4) shows, Enfragmo also has a moderate grounding time when compared to the solving time.

Another noteworthy point is that different gadgets show different performances under different combinations of problems and instances. So, using different gadgets also enables a knowledgeable user to choose the gadget that serves them best. The process of choosing a gadget can also be automatized through some heuristics in the grounder.

8 Conclusion

n model-based problem solving, users need to answer the questions in form of "What is the problem?" or "How can the problem be described?". In fact they do not need to know how the problem can be solved. In this approach, the users only need to describe their problems in a high-level language. Then, the system takes the specification, together with a problem instance, and produces the solution to the problem. This considerably reduces the amount of expertise a user need to have, and makes advanced solver technology accessible to a wider variety of users.

The untimate goal of having a high-level language is to enable naive users to encode their problems. For this purpose, it is necessary to enrich the language with useful constructs such as aggregates. Although, adding these constructs to the syntax of language does not increase the expressive power of the language, it makes the process of describing problems much easier. E.g. it is not easy to express the cardinality constraints in pure first-order logic. We extended our grounder to be able to convert the new structures to CNF and further showed that our grounder can reproduce the same CNF files from the high level language as the one obtained through direct reductions.

References

1. Frisch, A.M., Grum, M., Jefferson, C., Hernandez, B.M., Miguel, I.: The design of ESSENCE: a constraint language for specifying combinatorial problems. In: Proc. IJCAI'07. (2007)
2. Mitchell, D., Ternovska, E.: A framework for representing and solving NP search problems. In: Proc. AAAI'05. (2005)
3. Gent, I., Jefferson, C., Miguel, I.: Minion: A fast, scalable, constraint solver. In: ECAI 2006: 17th European Conference on Artificial Intelligence, August 29-September 1, 2006, Riva del Garda, Italy: including Prestigious Applications of Intelligent Systems (PAIS 2006): proceedings, Ios Pr Inc (2006) 98
4. Patterson, M., Liu, Y., Ternovska, E., Gupta, A.: Grounding for model expansion in k-guarded formulas with inductive definitions. In: Proc. IJCAI'07). (2007)
5. Mohebali, R.: A method for solving np search based on model expansion and grounding. Master's thesis, Simon Fraser University (2006)

6. Ternovska, E., Mitchell, D.G.: Declarative programming of search problems with built-in arithmetic. Technical Report TR-2007-28, Simon Fraser University (2008)
7. Patterson, M., Liu, Y., Ternovska, E., Gupta, A.: Grounding for model expansion in k-guarded formulas with inductive definitions. In: Proc. IJCAI'07. (2007) 161–166
8. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality Networks and Their Applications. In: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing, Springer-Verlag (2009)
9. Aavani, A., Wu, X., Mitchell, D., Ternovska, E.: Grounding Cardinality Constraints. (2010)
10. Eén, N.: SAT Based Model Checking. PhD thesis (2005)
11. Lynce, I., Marques-Silva, J.: Efficient haplotype inference with boolean satisfiability. In: AAAI, AAAI Press (2006)