Grounding Formulas with Complex Terms

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

Simon Fraser University
{aaa78,xwa33,ter,mitchell}@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. As the first-order sentences can be used to describe a combinatorial search problem, efficient grounding algorithms would help in solving such problems effectively and makes 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. In this paper, we describe the extension of this method to ground formulas of first-order logic extended with arithmetic, expansion functions and aggregate operators. Our method allows choice of particular CNF representations for complex constraints, easily.

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 propositional satisfiability (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 Satisfiability Modulo Theory (SMT) and ground constraint solvers, e.g., MINION [3], etc. 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 and $\phi(x)[x/\tilde{a}]$ denotes substituting $\tilde{a}$ for every occurrence of x in ϕ. In practice, though, effective grounding is not easy. Naive methods are too slow, and produce groundings that are too large and contain many redundant clauses.

Patterson et. al. defined a basic grounding method for function-free first-order logic (FO) in [4, 5], and a prototype implementation is described in [5]. Expressing most of interesting

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

real-world problems, e.g., Traveling Salesman problem or Knapsack problem, with function-free FO formula without having access to arithmetical operators is not an easy task. So, enriching the syntax with functions and arithmetical operators is a necessity. We describe how we have extended the existing grounding algorithm such that it can handle these constructs.

It is important to notice that the model expansion problem [5] is very different from query evaluation process. In model expansion context, there are formulas and sub-formulas which cannot be evaluated, while in query processing context, every formula can be evaluated as either true or false. First-order model expansion, when we are talking about finite domain, allows one to describe NP-complete problems while the query processing problem for FO, in finite domain context, is polynomial time. In this paper, we are interested in solving model expansion problem.

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.

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.

Our main contributions are:

1. We present an algorithm which can be used to ground specifications having different kinds of terms, e.g., aggregates, expansion/instance functions, arithmetic.
2. We enrich our language with aggregates, functions and arithmetical expression and design and develop an engine which can convert these constructs to pure SAT instances as well as to instances of SAT solvers which are able to handle more complex constraints such as cardinality constraints or Pseudo-Boolean constraints.
3. We define 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. $\mathcal{A}$ always denotes a finite σ-structure, called the instance structure, σ is the instance vocabulary, and ε the expansion vocabulary, and $\mathcal{L}$ is FO logic extended with arithmetic and aggregate operators.

Example 1. Consider the following variation of the knapsack problem:

We are given a set of items (loads), $L = \{l_1, \cdots, l_n\}$, and weight of each item is specified by an instance function W which maps items to integers ($w_i = W(l_i)$). We want to check if there is a way to put these n items into m knapsacks, $K = \{k_1, \cdots, k_m\}$ while satisfying the following constraints:

Certain items should be placed into certain knapsacks. These pairs are specified using the instance predicate "P". h of these m knapsacks have high capacity, each of them can carry a total load of H_{Cap}, while the capacity of the rest of the knapsacks is L_{Cap}. We

also do not want to put two items whose weights are very different in the same bag, i.e., the difference between the weights of the items in the same bag should be less than W_l. Each of H_{Cap}, L_{Cap} and W_l is an instance function with arity zero, i.e. a given constant.

The following formula ϕ in the first order logic is a specification for this problem:

$$\begin{array}{l}\{A_1 : \forall l \exists k : Q(l,k)\} \wedge \\ \{A_2 : \forall l \forall k_1 \forall k_2 : (Q(l,k_1) \wedge Q(l,k_2)) \supset k_1 = k_2\} \wedge \\ \{A_3 : \forall l,k : P(l,k) \supset Q(l,k)\} \wedge \\ \{A_4 : \forall k : \sum_{l:Q(l,k)} W(l) \leq H_{Cap}\} \wedge \\ \{A_5 : COUNT_k\{\sum_{l:Q(l,k)} W(l) \geq L_{Cap}\} \leq h\} \wedge \\ \{A_6 : \forall k,l_1,l_2 : (Q(l_1,k) \wedge Q(l_2,k)) \supset (W(l_1) - W(l_2) \leq W_l)\}\end{array}$$

An instance is a structure for vocabulary $\sigma = \{P, W, W_l, H_{Cap}, L_{Cap}\}$, i.e., a list of pairs, a function which maps items to integers and three constant integers. The task is to find an expansion $\mathcal{B}$ of $\mathcal{A}$ that satisfies ϕ:

$$\underbrace{(L \cup K; P^{\mathcal{A}}, W^{\mathcal{A}}, W_l^{\mathcal{A}}, H_{Cap}^{\mathcal{A}}, \overbrace{L_{Cap}^{\mathcal{A}}}^{\mathcal{A}}, Q^{\mathcal{B}})}_{\mathcal{B}} \models \phi.$$

Interpretations of the expansion vocabulary $\varepsilon = \{Q\}$, for structures $\mathcal{B}$ that satisfy ϕ, is a mapping from items to knapsacks that satisfies the problem properties.

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.*

A reduced grounding with respect to a given structure $\mathcal{A}$ can be obtained 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 [4], each tuple γ is associated with a formula ψ. For convenience, we use $\top$ and $\bot$ as propositional formulas which are always true and, false, respectively.

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 $\gamma : X \to A$, $\delta_{\mathcal{R}}(\gamma)$ 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 $P^{\mathcal{A}} = \{(1,2,3),(3,4,5)\}$. Answers to $\phi_1 \equiv P(x,y,z) \wedge E(x,y) \wedge E(y,z)$, $\phi_2 \equiv \exists z \phi_1$ and $\phi_3 \equiv \exists x \exists y \phi_2$ are demonstrated in Table 1.
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 $\phi_1[(1,1,1)]$. $E(1,2) \wedge E(2,3)$ is a reduced grounding of $\phi_2[(1,2)]$. Notice that, as ϕ_3 does not have any free variables, its corresponding answer has just a single row.

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

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

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

Table 1. Answers to ϕ_1, ϕ_2 and ϕ_3

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 [4, 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 proof for cases 1, 2 and 4 is given in [4]; 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, 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. 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. We are not going to define this concept here, but interested readers are encouraged to refer to [5].

This basic grounding approach can ground just the axioms A_1, A_2, A_3 in example 1.

2.1 FO MX with Arithmetic

In this paper, we are concerned with specifications written in FO extended with functions, arithmetic and aggregate operators. Informally, we assume that the domain of any instance structure is a subset of $\mathbb{N}$ (set of natural numbers), 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 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 We may use evaluation for formulas with expansion predicates. By evaluating a formula, which has expansion predicates, as true we mean 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 $\phi(\bar{z_1}, \bar{z_2})[\bar{z_1}/\bar{a}]$, which denotes substituting $\bar{a}$ for every occurrence of $\bar{z_1}$ in ϕ. Although our system supports grounding specification having Max, Min, Sum and Count aggregates, but for the sake of space, we just focus on Sum and Count aggregate in this paper:

- $t(\bar{y}) = Max_{\bar{x}}\{t(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y}); d_M(\bar{y})\}$, 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_M if there is none. d_M is the default value of Max aggregate which is returned whenever all conditions are evaluated as false.
- $t(\bar{y}) = Min_{\bar{x}}\{t(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y}); d_m(\bar{y})\}$ is defined dually to Max.
- $t(\bar{y}) = Sum_{\bar{x}}\{t(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y})\}$, for any instantiation $\bar{b}$ of $\bar{y}$, denotes 0 plus the sum of all values $t[\bar{a}, \bar{b}]$ for all instantiations $\bar{a}$ for $\bar{x}$ for which $\phi[\bar{a}, \bar{b}]$ is true.
- $t(\bar{y}) = Count_{\bar{x}}\{\phi(\bar{x}, \bar{y})\}$, for any instantiation $\bar{b}$ for $\bar{y}$, denotes the number of tuples $\bar{a}$ for which $\phi[\bar{a}, \bar{b}]$ is true. As we have $Count_{\bar{x}}\{\phi(\bar{x}, \bar{y})\} = Sum_{\bar{x}}\{1, \phi(\bar{x}, \bar{y})\}$, in the rest of this paper, we assume that all terms having Count aggregate are replaced with the appropriate terms in which Count is replaced with Sum aggregate and so we do not discuss the count aggregate, anymore.

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 special cases where terms do not contain expansion predicates/functions, and so they can be evaluated purely on 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, e.g., $\phi = P(x + y)$. As it is discussed previously, $\phi \Leftrightarrow \exists z(z = x + y \wedge P(z))$. Although we have not discussed handling of the sub-formula $z = x + y$, it is apparent that the answer to ϕ, with free variables $\{x, y\}$, is an extended $\{x, y\}$-relation R.

The extended relation R can be defined as the set of all tuples $(\langle a, b\rangle, \psi)$ 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 6 (Base Cases for Atoms with Evaluable 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. *P is an instance predicate: $\{(\gamma, \top) \mid \mathcal{A} \models P(t_1, \ldots t_n)[\gamma]\}$*
2. *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. *P is an expansion predicate: $\{(\gamma, P(a_1, \ldots a_n)) \mid \mathcal{A} \models (t_1 = a_1, \ldots t_n = a_n)[\gamma]\}$*

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

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. In example 1, this extension enables us to ground A_6.

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/functions. Thus, they cannot be evaluated out during grounding as in Section 3 and they somehow must be represented in the ground formula. We call a term which cannot be evaluated based on the instance structure a *complex term*.

In this section, we further extend the base cases of our relational algebra based grounding method to handle atomic formulas with complex terms. The key idea is to introduce the notion of 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, instance functions, expansion function, and aggregate operators involving these as well. The axioms, A_4 and A_5, in example 1, have these kinds of terms.

Let t be a term 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 a table representing all instantiations of X under which t might evaluate 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 ψ iff $(\gamma, \psi) \in \mathcal{R}$. We may also use $\delta_R(\gamma, n)$ and $\delta_{\beta_\mathcal{R}(n)}(\gamma)$ interchangeably.

Definition 7 (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 $\mathcal{A}$. Note that with this definition, α_R can be either the range of t or a superset of it.*

Example 3. (Continue of Example 1) Let $\psi(l_1, l_2) = W(l_1) - W(l_2) \leq W_l$ where the domains of both l_1 and l_2 are $L = \{0, 1, 2\}$. Let $\mathcal{A}$ be a σ-structure with $W^A = \{(0 \mapsto 7), (1 \mapsto 3), (2 \mapsto 5)\}$ and $W_l^A = 2$. Let $t = W_l$, $t_i = l_i$, $t'_i = W(l_i)$ $(i \in \{1, 2\})$, $t'' = t'_1 - t'_2$ be the terms in ψ, and $\mathcal{R}, \mathcal{R}_i, \mathcal{R}'_i$ $(i \in \{1, 2\})$, $\mathcal{R}''$ be answers to these terms, respectively. Then, $\alpha_\mathcal{R} = \{2\}$, $\alpha_{\mathcal{R}_i} = \{0, 1, 2\}$, $\alpha_{\mathcal{R}'_i} = \{3, 5, 7\}$ and $\alpha_{\mathcal{R}''} = \{0..4\}$.

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 D_X to be the set of all k-tuples of domain elements, i.e., $D_X = A^k$.

Proposition 3 (Answers to Terms). *Let $\mathcal{R}$ be the pair $(\alpha_\mathcal{R}, \beta_\mathcal{R})$, and t a term . Assume that $t_1, \ldots t_m$ are terms, and $\mathcal{R}_1, \ldots \mathcal{R}_m$ (respectively) are answers to those terms wrt $\mathcal{A}$. Also, let $\mathcal{S}$ be an answer for ϕ. 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): $\alpha_\mathcal{R} = \{n \in \mathbb{N} \mid \exists\, \overline{a} \in D_X : (t[\overline{a}] = n)\}$ and for all $n \in \alpha_\mathcal{R}$, $\beta_\mathcal{R}(n)$ is an answer to $t = n$ computed as described in Definition (6).

(2) t is a term in form of $t_1 + t_2$: $\alpha_\mathcal{R} = \{x + y \mid x \in \alpha_{\mathcal{R}_1}$ 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 term in form of $t_1 \{-, \times\} t_2$: similar to case (2);

(4) t is a term in form of $f(t_1, \cdots, t_m)$, where f is an instance function: $\alpha_\mathcal{R} = \{y |$ for some $x_1 \in \alpha_{\mathcal{R}_1}, \ldots, x_m \in \alpha_{\mathcal{R}_m},\ f(x_1, \ldots, x_m) = y\}$,
$\beta_\mathcal{R}(n) = \cup_{a_1 \in \alpha_{\mathcal{R}_1}, \ldots a_m \in \alpha_{\mathcal{R}_m},\ s.t. f(a_1, \ldots a_m) = n} \beta_{\mathcal{R}_1}(a_1) \bowtie \cdots \bowtie \beta_{\mathcal{R}_m}(a_m)$
Intuitively, $\beta_\mathcal{R}(n)$ is the combination of all possible ways that f can be evaluated as n.

(5) t *is a term in form of* $f(t_1, \cdots, t_m)$*, where* f *is an expansion function. We introduce an expansion predicate* $E_f(\bar{x}, y)$ *for each expansion function* $f(\bar{x})$ *where type of* y *is the same as range of* f*. Then* $\alpha_{\mathcal{R}}$ *is equal to range of* f*, and* $\beta_{\mathcal{R}}(n) = \cup_{a_1 \in \alpha_{\mathcal{R}_1}, \ldots a_m \in \alpha_{\mathcal{R}_m}}$

$$\beta_{\mathcal{R}_1}(a_1) \bowtie \cdots \bowtie \beta_{\mathcal{R}_m}(a_m) \bowtie \mathcal{T}_{a_1,\cdots,a_m,n}$$

Where $\mathcal{T}_{a_1,\cdots,a_m,n}$ *is an answer to* $\exists \wedge_i x_i = a_i \wedge y = n \wedge E_f(x_1, \cdots, x_m, y)$*.* $\beta_{\mathcal{R}}(n)$ *expresses that* $f(t_1, \cdots, t_m)$ *is equal to* n *under assignment* γ *iff* $t_i[\gamma] = a_i$ *and* $f(a_1, \cdots, a_m) = n$*.*

(6) t *is* $Sum_{\bar{x}}\{t_1(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y})\}$ *:* $\alpha_{\mathcal{R}} = \{\sum_{\bar{a} \in D_{\bar{x}}} f(\bar{a}) : \ f : \bar{x} \mapsto \{0\} \cup \alpha_{\mathcal{R}_1}\}$*. Let*

$$\delta'_{\mathcal{R}_1}(\gamma, n) = \begin{cases} \delta_{\mathcal{R}_1}(\gamma, n) & \text{if } n \neq 0 \\ \delta_{\mathcal{R}_1}(\gamma, 0) \vee \neg\delta_{\mathcal{S}}(\gamma) & \text{if } n = 0 \end{cases}$$

Then for each assignment $\bar{b} : \bar{y} \mapsto D_{(}\bar{y})$*:*

$$\delta_{\mathcal{R}}(\bar{b}, n) = \bigvee_{\substack{f:\bar{x}\mapsto\alpha_{\mathcal{R}} \\ s.t. \sum_{\bar{a}\in D_{\bar{x}}} f(\bar{a})=n}} \bigwedge_{\bar{a}\in D_{(}\bar{x})} \delta'_{\mathcal{R}_1}(\bar{a}, \bar{b}, f(\bar{a}))$$

For a fixed instantiation of $\bar{y}$ *(*$\bar{b}$*), each instantiation of* $\bar{x}$ *(*$\bar{a}$*), might or might not contribute to the output value of the aggregate when* $\bar{y}$ *is equal to* $\bar{b}$*.* $\bar{a}$ *contributes to the output of aggregate if* $\mathcal{B} \models \phi(\bar{a}, \bar{b})$*.* $\delta'_{\mathcal{R}_1}(\gamma, \alpha, f(\bar{a}))$ *describes the condition under which* $t_1(\bar{a})$ *contributes* $f(\bar{a})$ *to the output. So, for a given mapping from* $D_{\bar{x}}$ *to* $\alpha_{\mathcal{R}_1}$*, we need to conjunct the conditions obtained from* $\delta'_{\mathcal{R}_1}$ *to find the necessary and sufficient condition for one of the cases where the output sum is exactly* n*. And the outside disjunction, finds the complete condition.*

Although what is described in (7) can be used directly to find an answer for the Sum aggregates, in practice many of the entries in $\mathcal{R}$ will be eliminated during grounding as they are joined with false formula or unioned with a true formula. So, to reduce the grounding time, we use a place holder, SUM place holder, in the form $SUM(\mathcal{R}_1, \mathcal{S}, n, \gamma)$, as the formula corresponding to $\delta_{\mathcal{R}}(\gamma, n)$. The sum gadget is stored and propagated during grounding as a formula. After the end of the grounding phase, the engine enters the CNF generation phase in which a SAT instance is created from the obtained reduced grounding. In the CNF generation phase, the ground formula is traversed and by using normal Tseitin transformation [6] a corresponding CNF is generated[3].

While the engine traverses the formula tree, produced in the grounding phase it might encounter a SUM place holder. If this happens, the engine passes the SUM place holder to the SUM gadget which in turn converts the SUM place holder to CNF. This design has another benefit, too. If we decided to use a SAT solver which is capable of handling Pseudo-Boolean constraints natively, the SUM gadget can easily be changed to generate the Pseudo-Boolean constraints form the SUM place holder. One can find an implementation for Sum gadget in appendix A.

Example 4. (Continue From Example 3) $\beta_{\mathcal{R}_i}$ corresponds to the answer to variables l_i ($i \in \{1, 2\}$). So, $\mathcal{R}_1(\mathcal{R}_2)$ should have one free variable, namely $l_1(l_2)$. Having an answer to t_i, an answer to t'_i, $(\alpha_{\mathcal{R}'_i}, \beta_{\mathcal{R}'_i})$, can be computed. By proposition 3, we have $\beta_{\mathcal{R}''}(2) = \beta_{\mathcal{R}'_1}(7) \bowtie \beta_{\mathcal{R}'_2}(5) \cup \beta_{\mathcal{R}'_1}(5) \bowtie \beta_{\mathcal{R}'_2}(3)$. In other word, the answer to t'' is 2 if either $t'_1 = 7 \wedge t'_2 = 5$ or $t'_1 = 5 \wedge t'_2 = 3$.

[3] It is not the purpose of this paper to discuss the techniques we have used in this phase.

(a)

ψ
True

(b)

l_1	ψ
0	True

(c)

l_1	ψ
1	True

(d)

l_1	ψ
2	True

(e)

l_2	ψ
0	True

(f)

l_1	l_1	ψ
0	1	False
2	1	True

Table 2. Tables for Example 4: a) Answer to $W_l = 2$, i.e. $\beta_{\mathcal{R}}(2)$, b) Answer to $l_1 = 0$, i.e. $\beta_{\mathcal{R}_1}(0)$, c) Answer to $l_1 = 1$, i.e. $\beta_{\mathcal{R}_1}(1)$, e) Answer to $W(l_1) = 5$, i.e. $\beta_{\mathcal{R}'_1}(5)$, f) Answer to $W(l_2) = 7$, i.e. $\beta_{\mathcal{R}'_2}(7)$, g) Answer to $W(l_1) - W(l_2) = 2$, i.e. $\beta_{\mathcal{R}''}(2)$,

4.1 Base Case for Complex Terms

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

Definition 8 (Base Case for Atoms with Complex 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 $t_1 = t_2$: $\mathcal{R} = \cup_{(i \in \alpha_{\mathcal{R}_1} \cap \alpha_{\mathcal{R}_2})} \beta_{\mathcal{R}_1}(i) \bowtie \beta_{\mathcal{R}_2}(i)$*
2. *$P(...)$ is $t_1 \leq t_2$: $\mathcal{R} = \cup_{(i \in \alpha_{\mathcal{R}_1},\ j \in \alpha_{\mathcal{R}_2},\ i \leq j)} \beta_{\mathcal{R}_1}(i) \bowtie \beta_{\mathcal{R}_2}(j)$*
3. *P is an instance predicate: $\mathcal{R} = \cup_{(a_1, \cdots, a_m) \in P^{\mathcal{A}},\ a_i \in \alpha_{\mathcal{R}_i}} \beta_{\mathcal{R}_1}(a_1) \bowtie \cdots \bowtie \beta_{\mathcal{R}_m}(a_m)$*
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 Example 3) Although, ψ does not have any complex term, to demonstrate how the base cases can be handled, the process of computing an answer for ψ is described here. We have computed an answer to t'', t. To compute an answer to $\psi(l_1, l_2) = t''(l_1, l_2) \leq W_l$, one needs to find the union of $\beta_{\mathcal{R}''}(n) \bowtie \beta_{\mathcal{R}}(m)$ for $m \in \alpha_t = \{2\}$ and $n \leq 2 \in \{0..2\}$. In this example, $\{(0, 2, \top), (2, 1, \top)\}$ is an answer to ψ.

5 Experimental Evaluation

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

Thus far, we presented our approach to grounding aggregates and arithmetic. As a motivating example, we show how haplotype inference problem[7] 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.

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 [7] in the assumption that the gadget used for count is a simplified adder circuit [7].

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. $\forall i \forall j \left(g(i,j) = 2 \supset \exists k \left((h(k,j) \vee \neg ga(i,j) \vee \neg S^a(k,i)) \wedge (\neg h(k,j) \vee ga(i,j) \vee \neg S^a(k,i)) \wedge (h(k,j) \vee \neg gb(i,j) \vee \neg S^b(k,i)) \wedge (\neg h(k,j) \vee gb(i,j) \vee \neg S^b(k,i)) \right) \right)$

5,6. $\forall i \ (Count_k(S^a(k,i)) = 1) \wedge \forall i \ (Count_k(S^b(k,i)) = 1)$

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 (3) shows the detailed information about running time of haplotype inference instances produced by the ms program[7]. The axiomatization given above corresponds to the row labeled with "Optimized Encoding". The other row labeled with "Basic Encoding" also comes from the same paper [7] but as noted there and shown here produces CNF's that take more time to solve.

Table 3. 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

Thus, using our system, 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 achieve higher grounding speed although, as table (3) 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.

6 Related Work

The ultimate goal of all systems like ours is to have a high-level syntax which ease the task for problem description for both naive and expert users. To achieve this goal, these systems should be extended to handle complex terms. As different systems use different grounding approaches, each of them should have its very specific way of handling complex terms.

Essence, [8], is a declarative modelling language to specify search problems. In Essence, we do not have expansion predicates. Users need to describe their problems by expansion functions (which are variables, array of variables, matrix of variables and so on), instance predicates and mathematical operators. Then, the problem description is transformed to a Constraint Satisfying Problem, CSP, instance by an engine called *Tailor*. As there is no standard input format for CSP solvers, Tailor has to be developed separately for each CSP solver. Unlike SAT solvers which are just capable of handling Boolean variables, CSP solvers can work with instances in which the variables' domains are arbitrary. In [8], a method called *Flattening* has been described which resembles Tseitin transformation. Flattening process describes a complex term by introducing some auxiliary variables and decomposing the complex term into simpler terms. The flattening method is also used in Zinc system [9].

IDP, [10], is system for model expansion whose input is an extension of first-order logic with inductive definitions. Essentially, the syntax of IDP is very similar to that our system

but their approach to ground a given specification is different. A ground formula is created using a top-down procedure. The formula is written in Term Normal Form, TNF, in which all arguments to predicates are variables and the complex terms can only appear in the atomic formula in the form $x\{\leq, <, =, >, \geq\}t(\bar{y})$. And then, the atomic formulas which have complex terms are rewritten as disjunction or conjunction of atomic formulas in form $x < t(\bar{y})$ and $x > t(\bar{y})$ [11]. The ground solver used by IDP system is an extension of regular SAT solvers which is capable of handling aggregates internally. This enables them to translate specifications and instances into their ground solver input.

7 Conclusion

In 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 this approach, systems with a high-level language helps users a lot and reduces the amount of expertise a user need to have, and thus provides a way of solving computationally hard AI problems to a wider variety of users.

In this paper, we described how we extended our engine to handle complex terms. Having access to aggregates and arithmetic operators eases the task of describing problems for our system and, enables more users to work with our system for solving their theoretical and real-world problems. We also extended our grounder in such a way that it is able to convert the new construct to CNF and further showed that our grounder can reproduce the same CNF files from the high level language as the one obtains through direct reductions.

Acknowledgements

The authors are grateful to the Natural Sciences and Engineering Research Council of Canada (NSERC), MITACS and D-Wave Systems for their financial support. In addition, the anonymous reviewers' comments helped us in improving this paper and clarifying the presentation.

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) 161–166
5. Mohebali, R.: A method for solving NP search based on model expansion and grounding. Master's thesis, Simon Fraser University (2006)
6. Tseitin, G.: On the complexity of derivation in propositional calculus. Studies in constructive mathematics and mathematical logic **2**(115-125) (1968) 10–13
7. Lynce, I., Marques-Silva, J.: Efficient haplotype inference with boolean satisfiability. In: AAAI, AAAI Press (2006)
8. Rendl, A.: Effective compilation of constraint models. (2010)
9. Nethercote, N., Stuckey, P., Becket, R., Brand, S., Duck, G., Tack, G.: Minizinc: Towards a standard CP modelling language. Principles and Practice of Constraint Programming–CP 2007 (2007) 529–543
10. Wittocx, J., Mariën, M., De Pooter, S.: The idp system. Obtainable via www. cs. kuleuven. be/dtai/krr/software. html (2008)
11. Wittocx, J.: Finite domain and symbolic inference methods for extensions of first-order logic. AI Communications (2010) Accepted.

12. 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)
13. Eén, N.: SAT Based Model Checking. PhD thesis (2005)

A SUM Gadget

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 [12]). Our method allows these choices to be made at run time, either by the user or automatically.

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 present one method to construct a SUM gadget which can be used during the CNF Generation phase.

A gadget for Sum aggregate, denoted by $S(\mathcal{R}_1, \mathcal{S}, n)$, takes an answer to a term, $\mathcal{R}_1$, and an answer to a formula $\mathcal{S}$, and an integer and returns a CNF formula f.

Let's assume the original Sum aggregate is $t(\bar{y}) = \text{Sum}_{\bar{x}}\{t_1(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y})\}$, where $\mathcal{R}_1$ is an answer to t_1 and $\mathcal{S}$ is an answer to ϕ. Let $\mathcal{T}_i = \beta_{\mathcal{R}_1}(i) \bowtie \mathcal{S}$ for all $i \in \alpha_{\mathcal{R}_1}$. So, $\psi = \delta_{\mathcal{T}_i}(\gamma)$ is the necessary and sufficient condition for $t_1(\gamma)$ to be equal to i.

Remember that the SUM gadget is called during the CNF generation and it returns a Tseitin variable which is true iff $t(...) = n$. The gadget generates/retrieves a Tseitin variable, v_ψ for $\psi(\gamma)$ if $\psi = \delta_{\mathcal{T}_i}(\gamma) \neq \bot$ and stores the pair (i, v_ψ). After fetching all these pairs, $(n_1, v_1) \cdots (n_k, v_k)$, SUM gadget starts generating a CNF for $t(\gamma) = n$. In fact, our gadget can be any valid encoding for Pseudo-Boolean constraints such as Binary Decision Diagrams (BDD) based encoding, Sorting network based encoding and etc, [13]. Here, we describe the BDD based encoding:

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

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

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