Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Grounding with Bounds

Johan Wittocx and **Maarten Mariën** and **Marc Denecker**
Department of Computer Science, K.U. Leuven, Belgium
{johan,maartenm,marcd}@cs.kuleuven.be

Abstract

Grounding is the task of reducing a first-order theory to an equivalent propositional one. Typical grounders work on a sentence-by-sentence level, substituting variables by domain elements and simplifying where possible. In this work, we propose a method for *reasoning on the first-order theory as a whole* to optimize the grounding process. Concretely, we develop an algorithm that computes *bounds* for subformulas. Such bounds indicate for which tuples the subformulas are certainly true and for which they are certainly false. These bounds can then be used by standard grounding algorithms to substantially reduce grounding sizes, and consequently also grounding times. We have implemented the method, and demonstrate its practical applicability.

Introduction

Grounding, or propositionalisation, is an important computational task in many logic based reasoning systems. Herbrand model generators (Manthey and Bry 1988; Slaney 1994), model expansion solvers (Mitchell et al. 2006; Mariën, Wittocx, and Denecker 2007), and some theorem provers (Claessen and Sörensson 2003; Ganzinger and Korovin 2003) rely on grounding as a preprocessing step, reducing a predicate satisfiability problem in the context of a given domain to a propositional satisfiability problem. Grounding is of key importance in the context of the IDP constraint programming paradigm (Mitchell and Ternovska 2005; Mariën, Wittocx, and Denecker 2006) based on Model Expansion, and of Answer Set Programming (ASP), the constraint programming paradigm based on computing stable models of logic programs (Gelfond and Lifschitz 1988).

A basic (naive) grounding method is by instantiating variables by domain elements. Grounding in this way is polynomial in the size of the domain but exponential in the number of variables, and may easily produce propositional theories of unwieldy size. Therefore, a number of techniques have been developed for making smarter groundings. Well-known techniques for clausal First-Order logic theories are MACE-style grounding (McCune 1994) and the SEM-method (Zhang and Zhang 1996). Also in ASP, several efficient grounders have been developed (Syrjänen 1998; Leone, Perri, and Scarcello 2004).

The topic of this paper is to present an algorithm that, in combination with existing grounding techniques, can speed-up the grounding of First-Order logic theories and substantially reduce the size of groundings. To explain the intuition underlying our technique, consider the theory T_1 over the vocabulary $\{Edge, In\}$, consisting of the two sentences

$$\forall u, v\ (In(u, v) \supset Edge(u, v)) \quad (1)$$

$$\forall x, y, z\ (In(x, y) \wedge In(x, z) \supset y = z), \quad (2)$$

which express that In is a subgraph of $Edge$ with at most one outgoing edge in each vertex. Suppose we want to compute such subgraphs in the context of a given graph G with vertices D and edges E. This problem can be cast as a model expansion problem, the problem of computing all models of T_1 expanding the input structure I with domain D and $Edge^I = E$. The basic grounding algorithm would produce $|D|^2$ instantiations of (1) and $|D|^3$ of (2).

A first optimization, available in virtually all grounding systems, is by substituting truth values for atoms of known predicates (here $Edge$ and $=$), and then simplifying. In the example, this eliminates $|E|$ instantiations of (1) and $|D|$ instantiations of (2), but the result is still a theory of size $\mathcal{O}(|D|^3)$. One way to avoid such large groundings, is by adding redundant information to the formulas. For example,

$$\forall x, y, z(Edge(x, y) \wedge In(x, y) \wedge Edge(x, z) \wedge In(x, z) \supset y = z) \quad (3)$$

is equivalent to (2) given (1), but its grounding (after simplification!) is potentially a lot smaller. For example, if G itself has at most one outgoing edge per vertex, then the grounding of (3) would reduce to the empty theory.

The example illustrates how adding redundant information to formulas may dramatically reduce the size of the grounding and, consequently, lead to more efficient grounding. However, manually adding redundancy to formulas has its disadvantages: it leads to more complex and hence, less readable theories. Worse, it might introduce errors and it requires a good understanding of the used grounder, since what information is beneficial to add and where, depends on the grounder. Also, a human developer could easily miss useful information.

The above motivates a study of automated methods for deriving such redundant information and of principled ways

of adding it to formulas. We develop an algorithm that, given a theory T and a subset σ of the symbols of T, derives such redundant information, in the form of a pair of a *symbolic upper and lower bound* for each subformula of T. Each such a bound is a formula over σ. We then show how to insert these bounds in the formulas of T. For example, for T_1 and $\sigma = \{Edge, =\}$, our algorithm will compute $Edge(x, y)$ as upper bound for $In(x, y)$ and $\bot$ (false) as a lower bound; inserting this information in (2) transforms this formula into (3). The algorithm has been implemented and is a key component in the grounder GIDL. We present an evaluation of GIDL at the end of the paper.

Preliminaries

First-Order Logic (FO)

We assume the reader is familiar with FO. We introduce the conventions and notations used throughout this paper.

A vocabulary Σ consists of $\bot$, $\top$, variables, constant, predicate and function symbols. Variables and constant symbols are denoted by lowercase letters, predicate and function symbols by uppercase letters. Sets and tuples of variables are denoted by $\overline{x}, \overline{y}, \ldots$. For a formula φ, we often write $\varphi[\overline{x}]$, to indicate that $\overline{x}$ are its free variables. The formula $\varphi[x/c]$ is the formula obtained by replacing in φ all free occurrences of variable x by constant symbol c. This notation is extended to tuples of variables and constant symbols of the same length.

A formula is in *term normal form* (TNF) when all its atomic subformulas are of the form $P(x_1, \ldots, x_n)$, $F(x_1, \ldots, x_n) = x$, $c = x$ or $x_1 = x_2$, where P, F and c are respectively a predicate, function and constant symbol, and $x_1, \ldots, x_n, x$ are variables. Each FO formula is equivalent to a formula in TNF.

The truth values *true* and *false* are denoted by respectively **t** and **f**. A Σ-*interpretation* I consists of a domain D and an assignment of appropriate values to each of its symbols, i.e.:

- **t** to $\top$ and **f** to $\bot$.
- an element $x^I \in D$ to every variable $x \in \Sigma$;
- an element $a^I \in D$ to every constant symbol $a \in \Sigma$;
- a set $P^I \subseteq D^n$ to every n-ary predicate symbol $P \in \Sigma$;
- a function $F^I : D^n \to D$ to every n-ary function symbol $F \in \Sigma$;

A Σ-*structure* is an interpretation of only the relation, constant and function symbols of Σ. The restriction of a Σ-interpretation I to a vocabulary $\sigma \subseteq \Sigma$ is denoted by $I|_\sigma$. $I[x/d]$ is the interpretation that assigns d to x and corresponds to I on all other symbols. This notation is extended to tuples of variables and domain elements of the same length.

The value t^I of a term t in an interpretation I, and the satisfaction relation $\models$ are defined as usual (see, e.g., (Enderton 1972)).

Grounding

A *grounding* of an FO theory T is an "equivalent" propositional theory T_g. The notion of equivalence depends on the application one has in mind. In this paper, we consider the application of *model expansion*.

Definition 1 (model expansion). *Given a theory T over a vocabulary Σ, a vocabulary $\sigma \subseteq \Sigma$ and a finite σ-structure I_σ, the* model expansion (MX) search problem for input $\langle \Sigma, T, \sigma, I_\sigma \rangle$ *is the problem of finding models M of T that expand I_σ, i.e., $M|_\sigma = I_\sigma$. The* MX decision problem *for the same input is the problem of deciding whether such a model exists.*

MX problems are commonly solved by creating an appropriate grounding T_g of T using I_σ and subsequently calling a propositional model generator (in the case of the search problem) or satisfiability checker (in the case of the decision problem) for input T_g. A grounding algorithm for MX was presented in (Patterson et al. 2007). For solving the MX decision problem, it suffices that T_g is satisfiable iff T has a model expanding I_σ. For solving the MX search problem, a one-to-one correspondence between the models of T_g and the models of T expanding I_σ is required. The method of this paper preserves the latter, stronger type of equivalence.

We now define grounding formally. For a domain D, let Σ^D be the vocabulary Σ extended with a new constant symbol $\mathbf{d}$ for every $d \in D$. We call these new constants *domain constants* and denote the set of all domain constants by $\mathbf{D}$. For a Σ-structure I with domain D, denote by I^D the structure expanding I to Σ^D by interpreting every $\mathbf{d} \in \mathbf{D}$ by d. A formula is in *ground normal form* (GNF) if it contains no quantifiers and all its atomic subformulas are of the form $P(\mathbf{d}_1, \ldots, \mathbf{d}_n)$, $F(\mathbf{d}_1, \ldots, \mathbf{d}_n) = \mathbf{d}$, $c = \mathbf{d}$ or $\mathbf{d}_1 = \mathbf{d}_2$, where P, F and c are respectively a predicate, function and constant symbol of Σ, and $\mathbf{d}_1, \ldots, \mathbf{d}_n, \mathbf{d}$ are domain constants. A formula in GNF is essentially propositional.

Definition 2 (grounding). *Let T be a theory over Σ, $\sigma \subseteq \Sigma$ and I_σ a σ-structure with finite domain D. A* grounding for T with respect to I_σ *is a theory T_g over Σ^D such that all sentences of T_g are in GNF and for every Σ-structure M expanding I_σ, $M \models T$ iff $M^D \models T_g$. T_g is called* reduced *if it contains no symbols of σ.*

A basic reduced grounding $Gr(T)$ of a theory T in TNF consists of all sentences $Gr(\varphi)$ where φ is a sentence of T and Gr is defined as follows. For sentences φ over σ^D, $Gr(\varphi) = \top$ if $I_\sigma^D \models \varphi$ and $Gr(\varphi) = \bot$ if $I_\sigma^D \not\models \varphi$. For all other sentences φ, we define

$$Gr(\varphi) = \begin{cases} \bigwedge_{\mathbf{d} \in \mathbf{D}} Gr(\psi[x/\mathbf{d}]) & \text{if } \varphi := \forall x\ \psi[x] \\ \bigvee_{\mathbf{d} \in \mathbf{D}} Gr(\psi[x/\mathbf{d}]) & \text{if } \varphi := \exists x\ \psi[x] \\ Gr(\psi_1) \wedge Gr(\psi_2) & \text{if } \varphi := \psi_1 \wedge \psi_2 \\ Gr(\psi_1) \vee Gr(\psi_2) & \text{if } \varphi := \psi_1 \vee \psi_2 \\ \neg Gr(\psi) & \text{if } \varphi := \neg\psi \\ \psi & \text{if } \psi \text{ is an atom} \end{cases}$$

$Gr(T)$ can be simplified by recursively replacing $\top \wedge \varphi$ by φ, $\bot \wedge \varphi$ by $\bot$, etc. The resulting ground theory is the one computed by most grounding algorithms. A smart algorithm uses different techniques from database theory in order to construct $Gr(T)$ efficiently and to interleave the construction and simplification phase (Leone, Perri, and Scarcello 2004; Syrjänen 1998).

Grounding with Bounds

As mentioned in the introduction, we present a method of computing smaller groundings of a theory T with the help of lower and upper bounds for the subformulas of T. In this section, we formally define these bounds and give an overview of a grounding algorithm which uses them.

For the rest of this paper, let T be a theory in TNF over Σ, σ a subvocabulary of Σ and I_σ a finite σ-structure.

Bounds

We distinguish between two kinds of bounds.

Definition 3 (ct-bound, cf-bound). *A* certainly true bound (ct-bound) over σ with respect to T for a formula $\varphi[\overline{x}]$ *is a formula* $\varphi_{ct}[\overline{y}]$ *over* σ *such that* $\overline{y} \subseteq \overline{x}$ *and* $T \models \forall\overline{x}\ (\varphi_{ct}[\overline{y}] \supset \varphi[\overline{x}])$. *Vice versa, a* certainly false bound (cf-bound) over σ with respect to T for $\varphi[\overline{x}]$ *is a formula* $\varphi_{cf}[\overline{z}]$ *over* σ *such that* $\overline{z} \subseteq \overline{x}$ *and* $T \models \forall\overline{x}\ (\varphi_{cf}[\overline{z}] \supset \neg\varphi[\overline{x}])$.

When σ and T are clear from the context, we will simply refer to ct- and cf-bounds.

Intuitively, a ct-bound for φ provides a lower bound for the set of tuples for which φ is true in *every* model of T. Likewise, a cf-bound provides a lower bound on the set of tuples for which φ is false. Hence, the negation of a ct-bound (cf-bound) gives an upper bound on the set of tuples for which φ is false (true) in *some* model of T.

Example 1. Let T_1 be the theory of the introduction, $\sigma_1 = \{Edge, =\}$ and φ_1 the subformula $In(x,y) \wedge In(x,z)$ of T_1. Then $\neg Edge(x,y) \vee \neg Edge(x,z)$ is a cf-bound over σ_1 with respect to T_1 for φ_1. Indeed, one can derive from (1) that T_1 entails $\forall x,y,z\ ((\neg Edge(x,y) \vee \neg Edge(x,z)) \supset \neg\varphi_1)$.

Observe that $\top$ is a ct-bound for every *sentence* of T. Also, $\bot$ is a ct-bound as well as a cf-bound for every *formula*. We call $\bot$ the *trivial bound*. Intuitively, the trivial bound contains no information at all. According to the following definition, it is the least precise bound.

Definition 4 (precision). *Let* $\psi[\overline{y}]$ *and* $\chi[\overline{z}]$ *be two (ct- or cf-) bounds for* $\varphi[\overline{x}]$. *We say that* $\psi[\overline{y}]$ is more precise than $\chi[\overline{z}]$ *if* $\models \forall\overline{x}\ \chi[\overline{z}] \supset \psi[\overline{y}]$.

The bounds needed for grounding T are bounds for the subformulas of T. We associate a ct- and cf-bound to every subformula of T.

Definition 5 (c-map). *A* c-map $\mathcal{C}$ *over* σ *for a theory* T *is a mapping from all subformulas* φ *of* T *to tuples* $(\varphi^{\mathcal{C}}_{ct}, \varphi^{\mathcal{C}}_{cf})$, *where* $\varphi^{\mathcal{C}}_{ct}$ *and* $\varphi^{\mathcal{C}}_{cf}$ *are respectively a ct- and cf-bound over* σ *for* φ.

The notion of precision pointwise extends to c-maps. A c-map is *inconsistent* if some formula φ is both certainly true and false for some tuple, according to that c-map:

Definition 6 (inconsistent c-map). *A c-map* $\mathcal{C}$ *over* σ *for* T *is* inconsistent *(resp.* inconsistent with respect to I_σ*) if for some subformula* $\varphi[\overline{x}]$ *of* T, $\models \exists\overline{x}\varphi^{\mathcal{C}}_{ct} \wedge \varphi^{\mathcal{C}}_{cf}$ *(resp.* $I_\sigma \models \exists\overline{x}\varphi^{\mathcal{C}}_{ct} \wedge \varphi^{\mathcal{C}}_{cf}$*)*.

A c-map for T can only be inconsistent if T is unsatisfiable.

Let us now consider how to insert the bounds of $\mathcal{C}$ into the sentences of T. Observe that, for a model M of T and arbitrary $\overline{d}$, $M[\overline{x}/\overline{d}] \models \varphi[\overline{x}]$ iff $M[\overline{x}/\overline{d}] \models (\varphi[\overline{x}] \wedge \neg\varphi^{\mathcal{C}}_{cf}) \vee \varphi^{\mathcal{C}}_{ct}$.

Definition 7 (c-transformation). *A* c-transformation *of a subformula* φ *of* T *with respect to* $\mathcal{C}$*, denoted* $\mathcal{C}(\varphi)$*, is the formula* $(\varphi' \wedge \neg\varphi^{\mathcal{C}}_{cf}) \vee \varphi^{\mathcal{C}}_{ct}$ *where* φ' *is defined by*

- $\varphi' := \neg\mathcal{C}(\psi)$ *if* $\varphi \equiv \neg\psi$.
- $\varphi' := \varphi$ *if* φ *is an atom;*
- $\varphi' := \mathcal{C}(\psi) \wedge \mathcal{C}(\chi)$ *if* $\varphi \equiv \psi \wedge \chi$*;*
- $\varphi' := \mathcal{C}(\psi) \vee \mathcal{C}(\chi)$ *if* $\varphi \equiv \psi \vee \chi$*;*
- $\varphi' := \exists x\, \mathcal{C}(\psi)$ *if* $\varphi \equiv \exists x\, \psi$*;*
- $\varphi' := \forall x\, \mathcal{C}(\psi)$ *if* $\varphi \equiv \forall x\, \psi$*;*

A c-transformation $\mathcal{C}(T)$ of T with respect to $\mathcal{C}$ *consists of a c-transformation with respect to* $\mathcal{C}$ *of every sentence of* T.

Example 1 (continued). Let $\mathcal{C}_1$ be the c-map for T_1 assigning $(\top, \bot)$ to (1), $(\bot, \neg Edge(x,y) \vee \neg Edge(x,z))$ to $In(x,y) \wedge In(x,z)$ and $(\bot, \bot)$ to every other subformula. Then – leaving out the bounds equal to $\bot$ – $\mathcal{C}_1(T_1)$ consists of the two sentences (1) $\vee \top$ and $\forall x,y,z\ (In(x,y) \wedge In(x,z) \wedge (Edge(x,y) \wedge Edge(x,z)) \supset y = z)$.

Observe that in the above example, $\mathcal{C}_1(T_1)$ is not equivalent to T_1. Indeed, the information that In must be a subset of $Edge$ in every model is lost. In the more extreme case of a c-map $\mathcal{C}_2$, assigning $(\top, \bot)$ to both (1) and (2), $\mathcal{C}_2(T_1)$ is even equivalent to $\top$, which contains no information at all. This lost information must be added to $\mathcal{C}(T)$ in order to get a theory equivalent to T.

Theorem 8. *Let* T *be an FO theory and* $\mathcal{C}$ *a c-map for* T. *Then* T *is equivalent to*

$$\mathcal{C}(T)$$
$$\cup\ \{\forall\overline{x}\ \varphi^{\mathcal{C}}_{ct} \supset \varphi[\overline{x}]\ \mid\ \varphi[\overline{x}] \text{ is a subformula of } T\ \} \quad (4)$$
$$\cup\ \{\forall\overline{x}\ \varphi^{\mathcal{C}}_{cf} \supset \neg\varphi[\overline{x}]\ \mid\ \varphi[\overline{x}] \text{ is a subformula of } T\ \} \quad (5)$$

Although after simplification, $Gr(\mathcal{C}(T))$ is typically smaller than $Gr(T)$, that is not necessarily the case for $Gr(\mathcal{C}(T) \cup (4) \cup (5))$. For certain c-maps however, (4) and (5) contain a small subset of relevant sentences, i.e. there are small sets $V_{ct} \subset$ (4) and $V_{cf} \subset$ (5) such that $V_{ct} \models$ (4) and $V_{cf} \models$ (5). All sentences in (4) $\setminus\ V_{ct}$ resp. (5) $\setminus\ V_{cf}$ are redundant and it is sufficient to compute $Gr(\mathcal{C}(T) \cup V_{ct} \cup V_{cf})$ instead of $Gr(\mathcal{C}(T) \cup (4) \cup (5))$. An *atom-based c-map* is a particular kind of c-map with this property.

Definition 9 (atom-based c-map). *Let* $\mathcal{C}$ *be a c-map for* T *and let* $V_{ct} = \{\forall\overline{x}\ \varphi^{\mathcal{C}}_{ct} \supset \varphi[\overline{x}]\ \mid\ \varphi[\overline{x}]$ *is an atomic subformula of* $T\ \}$ *and* $V_{cf} = \{\forall\overline{x}\ \varphi^{\mathcal{C}}_{cf} \supset \neg\varphi[\overline{x}]\ \mid\ \varphi[\overline{x}]$ *is an atomic subformula of* $T\ \}$. *We call* $\mathcal{C}$ atom-based *if* $V_{ct} \models$ (4) *and* $V_{cf} \models$ (5).

A c-map $\mathcal{C}$ assigning $(\bot, \bot)$ to every non-atomic subformula is atom-based. In the next section, we show how to compute a more interesting atom-based c-map. Remark that for an atom-based c-map and corresponding V_{ct} and V_{cf}, $Gr(V_{ct} \cup V_{cf})$ consists, after simplification, only of atoms.

We now have the following grounding algorithm. First compute an atom-based c-map $\mathcal{C}$ for T with corresponding sets V_{ct} and V_{cf}. If $\mathcal{C}$ is inconsistent with respect to I_σ, output $\bot$ and stop. Else, output $T_g := Gr(\mathcal{C}(T) \cup V_{ct} \cup V_{cf})$, using some existing grounding algorithm.

Computing Bounds

In this section, we develop an algorithm to compute a non-trivial c-map $\mathcal{C}$ for T. We then sketch how to derive an atom-based c-map from $\mathcal{C}$.

Refining c-maps

Constructing a non-trivial c-map can be done by starting from the least precise c-map, i.e. the one that assigns $(\bot, \bot)$ to every subformula of T, and then gradually refining it. To refine a ct-bound $\varphi^{\mathcal{C}}_{ct}$, assigned by $\mathcal{C}$ to a subformula φ, first a new ct-bound ψ^r_{ct} for φ is derived using $\mathcal{C}$ and T. We call this new bound a *refinement ct-bound*. Next, $\varphi^{\mathcal{C}}_{ct}$ is replaced by $\varphi^{\mathcal{C}}_{ct} \vee \psi^r_{ct}$ to obtain a c-map that is more precise than $\mathcal{C}$. Likewise, cf-bounds can be refined.

We present six different ways to obtain refinement bounds, given T and a c-map $\mathcal{C}$. If the sentences of T are represented by their parse trees, each node corresponds to a subformula of T. *Bottom-up refinement* refines the bounds assigned to a node by considering the bounds assigned by $\mathcal{C}$ to its children. Vice versa, *top-down refinement* does so by looking at the parents and siblings of a node. *Axiom refinement* sets bounds for the roots, i.e. for the sentences of T, while *copy* and *functional refinement* refine the leaves.

Input refinement Let $\varphi[\overline{x}]$ be a formula over σ. Since $T \models \forall \overline{x}\ (\varphi[\overline{x}] \supset \varphi[\overline{x}])$ and $T \models \forall \overline{x}\ (\neg\varphi[\overline{x}] \supset \neg\varphi[\overline{x}])$, it is clear that $\varphi[\overline{x}]$ is a ct-bound and $\neg\varphi[\overline{x}]$ a cf-bound for $\varphi[\overline{x}]$. We call them *input* refinement bounds.

Axiom refinement If φ is a sentence of T, then $\top$ is an *axiom* refinement ct-bound for φ. This refinement bound states that a sentence of T is true in every model of T.

Bottom-up refinement For a compound subformula φ, depending on its structure, the following table gives the *bottom-up* refinement ct-bound φ^r_{ct} and cf-bound φ^r_{cf} for φ with respect to $\mathcal{C}$. It is rather straightforward to obtain these formulas. E.g., the formula in the bottom-right of the table indicates that if φ is the formula $\psi \vee \chi$, then φ is certainly false for those tuples for which both ψ and χ are certainly false. Or, more formally, if both $T \models \psi^{\mathcal{C}}_{cf} \supset \neg\psi$ and $T \models \chi^{\mathcal{C}}_{cf} \supset \neg\chi$, then $T \models \psi^{\mathcal{C}}_{cf} \wedge \chi^{\mathcal{C}}_{cf} \supset \neg(\psi \vee \chi)$.

φ	φ^r_{ct}	φ^r_{cf}
$\neg\psi$	$\psi^{\mathcal{C}}_{cf}$	$\psi^{\mathcal{C}}_{ct}$
$\forall x\ \psi$	$\forall x\ \psi^{\mathcal{C}}_{ct}$	$\exists x\ \psi^{\mathcal{C}}_{cf}$
$\exists x\ \psi$	$\exists x\ \psi^{\mathcal{C}}_{ct}$	$\forall x\ \psi^{\mathcal{C}}_{cf}$
$\psi \wedge \chi$	$\psi^{\mathcal{C}}_{ct} \wedge \chi^{\mathcal{C}}_{ct}$	$\psi^{\mathcal{C}}_{cf} \vee \chi^{\mathcal{C}}_{cf}$
$\psi \vee \chi$	$\psi^{\mathcal{C}}_{ct} \vee \chi^{\mathcal{C}}_{ct}$	$\psi^{\mathcal{C}}_{cf} \wedge \chi^{\mathcal{C}}_{cf}$

Top-down refinement In the case of *top-down* refinements, the bounds of a formula ψ are used to construct refinement bounds for its direct subformulas (i.e., its children in the parse tree). The top-down refinement ct-bounds φ^r_{ct} and cf-bounds φ^r_{cf} of a direct subformula φ of ψ are given by the following table. The tuple $\overline{y}$ denotes the free variables of χ not occurring in φ.

ψ	φ^r_{ct}	φ^r_{cf}
$\neg\varphi$	$\psi^{\mathcal{C}}_{cf}$	$\psi^{\mathcal{C}}_{ct}$
$\forall x\ \varphi$	$\psi^{\mathcal{C}}_{ct}$	$\bot$
$\exists x\ \varphi$	$\bot$	$\psi^{\mathcal{C}}_{cf}$
$\varphi \wedge \chi$ or $\chi \wedge \varphi$	$\exists \overline{y}\ \psi^{\mathcal{C}}_{ct}$	$\exists \overline{y}\ \psi^{\mathcal{C}}_{cf} \wedge \chi^{\mathcal{C}}_{ct}$
$\varphi \vee \chi$ or $\chi \vee \varphi$	$\exists \overline{y}\ \psi^{\mathcal{C}}_{ct} \wedge \chi^{\mathcal{C}}_{cf}$	$\exists \overline{y}\ \psi^{\mathcal{C}}_{cf}$

We illustrate two of these refinement bounds. Let ψ be the formula $\forall x\ P(x, y)$. Recall that intuitively, the ct-bound $\psi^{\mathcal{C}}_{ct}$ indicates for which $d \in D$, $\forall x\ P(x, \mathbf{d})$ is certainly true. For such a d and an arbitrary $d' \in D$, $P(\mathbf{d}', \mathbf{d})$ must be true. Hence, $\psi^{\mathcal{C}}_{ct}$ is a ct-bound for φ. Indeed, $T \models \forall x, y\ (\psi^{\mathcal{C}}_{ct} \supset P(x, y))$ follows directly from $T \models \forall y\ (\psi^{\mathcal{C}}_{ct} \supset \forall x\ P(x, y))$.

Now let ψ be the formula $P(x) \vee Q(x, y)$. If we know that $P(\mathbf{d}_1) \vee Q(\mathbf{d}_1, \mathbf{d}_2)$ is certainly false, but $Q(\mathbf{d}_1, \mathbf{d}_2)$ is certainly true, then $P(\mathbf{d}_1)$ must be certainly false. Hence, $\exists y\ \psi^{\mathcal{C}}_{cf} \wedge \chi^{\mathcal{C}}_{ct}$ is a cf-bound for $P(x)$.

Functional refinement If $\varphi[\overline{x}, y]$ is the formula $F(\overline{x}) = y$, functional refinement bounds for φ take into account that F is a function. The functional refinement ct-bound φ^r_{ct} and cf-bound φ^r_{cf} are given by:

$$\varphi^r_{ct} := \forall y'\ (y' \neq y \supset \varphi^{\mathcal{C}}_{cf}[y/y'])$$

$$\varphi^r_{cf} := \exists y'\ (\varphi^{\mathcal{C}}_{ct}[y/y'] \wedge y \neq y')$$

where y' is a new variable. Informally, the first of these formulas indicates that $F(\overline{x})$ is certainly equal to y if for every $y' \neq y$, $F(\overline{x})$ is certainly not equal to y'. The second one says that $F(\overline{x})$ is certainly not equal to y if $F(\overline{x})$ is certainly y' for some $y' \neq y$.

Copy refinement Copy refinement copies a bound of a formula to a similar formula. We present copy refinement for one specific type of formulas, but it can easily be generalized. Let φ be the formula $P(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are n different variables and let $P(y_1, \ldots, y_n)$ be another subformula of T, where the y_i are n different variables. Then the formula obtained by replacing in $\varphi^{\mathcal{C}}_{ct}$ all bound variables by new variables and renaming all free variables x_i by y_i, is a *copy refinement* ct-bound for $P(y_1, \ldots, y_n)$. E.g., if φ is the formula $P(x_1, x_2)$ and $\varphi^{\mathcal{C}}_{ct}$ is the bound $\forall z\ Q(z, x_2)$, then the copy refinement ct-bound for $P(y_1, y_2)$ is given by $\forall z'\ Q(z', y_2)$, where z' is a new variable. A copy refinement cf-bound is defined similarly.

Definition 10. *We call φ^r_{ct} (φ^r_{cf}) a* refinement ct-bound (cf-bound) for φ *if it is an input, axiom, bottom-up, top-down, functional or copy refinement ct-bound (cf-bound) for φ.*

Proposition 11. *If φ^r_{ct} is a refinement ct-bound for φ, then $\varphi^{\mathcal{C}}_{ct} \vee \varphi^r_{ct}$ is a ct-bound for φ. Moreover, it is a more precise ct-bound than $\varphi^{\mathcal{C}}_{ct}$. The same property holds for cf-bounds.*

Refinement Algorithm Based on Proposition 11, Algorithm 1 presents an indeterministic any-time algorithm to compute ct- and cf-bounds for any subformula of a theory T.

Input: A theory T over Σ and a vocabulary $\sigma \subseteq \Sigma$
1 Set $\varphi^{\mathcal{C}}_{ct} = \bot$ and $\varphi^{\mathcal{C}}_{pf} = \bot$ for every subformula φ of T;
2 **while** *the stop criterion is not reached* **do**
3 Choose a subformula φ of T;
4 Let $\varphi^{r}_{ct} =$ `ct_refinement(`φ`)`;
5 Let $\varphi^{r}_{cf} =$ `cf_refinement(`φ`)`;
6 $\varphi^{\mathcal{C}}_{ct} =$ `simplify(`$\varphi^{\mathcal{C}}_{ct} \vee \varphi^{r}_{ct}$`)`;
7 $\varphi^{\mathcal{C}}_{cf} =$ `simplify(`$\varphi^{\mathcal{C}}_{cf} \vee \varphi^{r}_{cf}$`)`;

Algorithm 1: Computing bounds

The functions `ct_refinement` and `cf_refinement` return some refinement ct-bound, respectively cf-bound, constructed using the current c-map $\mathcal{C}$ for T. The function `simplify` takes as argument an FO formula $\varphi[\overline{x}]$ and returns a logically equivalent FO formula $\varphi'[\overline{x}]$, such that φ' is smaller than φ. Remark that if `simplify` is the identity function, the size of the bounds will increase every time they are refined, which may lead to an exponential blow-up.

As presented here, the choice of φ in line 3 is arbitrary, as is the choice of the kind of refinement bound φ^{r}_{ct} and φ^{r}_{cf} in line 4 and 5. A reasonable choice heuristic consists of first applying all possible axiom and input refinements and then only choosing those φ which children, siblings or parent in the parse tree have a recently changed bound. E.g., if $\varphi \wedge \chi$ is a subformula of T and $\chi^{\mathcal{C}}_{cf}$ is recently changed by the algorithm, then adapt $\varphi^{\mathcal{C}}_{ct}$ by using top-down refinement.

Algorithm 1 reaches a fixpoint when it cannot refine the current c-map $\mathcal{C}$ to a more precise one. Such a fixpoint cannot always be reached, as we will illustrate below. Hence some stop criterion for the while-loop is needed, e.g., a bound on the number of iterations of the loop. Also, it is undecidable whether a fixpoint is reached. However, if the simplification algorithm is strong enough, it is sometimes possible that none of the refinements leads to a syntactically different c-map. It is then certain that a fixpoint is reached.

Example 1 (continued). For $T = T_1$ and $\sigma = \sigma_1$, Algorithm 1 can reach a fixpoint. The final c-map assigns $\bot$ as ct-bound and $(y \neq z) \vee \neg Edge(x, y) \vee \neg Edge(x, z)$ as cf-bound to φ_1.

Example 2. Consider the theory T_2, taken from some planning domain, consisting of the sentence $\forall a_0, a_p, t_0\ \ Prec(a_p, a_0) \wedge Do(a_0, t_0) \supset (\exists t_p\ \ t_p < t_0 \wedge Do(a_p, t_p))$. This sentence describes that an action a_0 with precondition a_p can only be performed at timepoint t_0 if a_p is performed at an earlier timepoint t_p. For every $i > 0$, let $\chi_i[a_0, t_0]$ be a formula over $\sigma_2 = \{Prec, <\}$, expressing that t_0 is smaller than the ith timepoint and there exists a chain of actions $a_1, \ldots, a_i$ such that $Prec(a_1, a_0) \wedge \ldots \wedge Prec(a_i, a_{i-1})$ holds. For $T = T_2$ and $\sigma = \sigma_2$, one can check that for any $n > 0$, given enough iterations of the while-loop, Algorithm 1 can derive cf-bound $\psi_n := \chi_1 \vee \ldots \vee \chi_n$ for $Do(a_0, t_0)$. Clearly, for $n_1 \neq n_2$, ψ_{n_1} is not equivalent to ψ_{n_2}. This indicates that a fixpoint cannot be reached for T_2.

Deriving an atom-based c-map

A method to derive an atom-based c-map from an arbitrary c-map is based on the following observation. Let $\mathcal{C}$ be a c-map for T and let φ be the subformula $\chi \wedge \psi$ of T. If $\varphi^{\mathcal{C}}_{ct} \equiv \chi^{\mathcal{C}}_{ct} \wedge \psi^{\mathcal{C}}_{ct}$, i.e. it is the bottom-up refinement ct-bound for φ according to $\mathcal{C}$, then $T \models \forall\, (\varphi^{\mathcal{C}}_{ct} \supset \varphi)$ is implied by $T \models \forall\, (\chi^{\mathcal{C}}_{ct} \supset \chi)$ and $T \models \forall\, (\psi^{\mathcal{C}}_{ct} \supset \psi)$. A similar observation holds for all other bottom-up refinement bounds.

We call a c-map $\mathcal{C}$ for T a *bottom-up c-map* if for every non-atomic subformula φ of T, $\varphi^{\mathcal{C}}_{ct}$ is the bottom-up refinement ct-bound and $\varphi^{\mathcal{C}}_{cf}$ is the bottom-up refinement cf-bound for φ with respect to $\mathcal{C}$. Observe that a bottom-up c-map $\mathcal{C}$ for T is completely determined by the bounds it assigns to the atomic subformulas of T. Hence, given a c-map, one can derive a bottom-up c-map from it by retaining the bounds for the atomic subformulas and then computing the corresponding bottom-up c-map. Because of the following proposition, this derived c-map is also atom-based.

Proposition 12. *A bottom-up c-map $\mathcal{C}$ for T is an atom-based c-map for T.*

Example 1 (continued). For $T = T_1$ and $\sigma = \sigma_1$, and let $\mathcal{C}_3$ be the fixpoint computed by Algorithm 1. The bottom-up c-map $\mathcal{C}_4$ derived from $\mathcal{C}_3$ assigns $(\top, \bot)$ to (1) and $(\bot, \neg Edge(x, y) \vee \neg Edge(x, z))$ to $In(x, y) \wedge In(x, z)$.

For $\mathcal{C}_4$ as in the example above, $\mathcal{C}_4(In(x, y) \wedge In(x, z)) = (In(x, y) \wedge Edge(x, y)) \wedge (In(x, z) \wedge Edge(x, z))) \wedge (Edge(x, y) \wedge Edge(x, z))$. This formula contains repeated constraints on the variables x, y and z. In general applying bottom-up c-maps produces many such repetitions. These could easily be eliminated, but it depends on the used grounder which ones are best deleted. No deletions are done in the experiments of the next section.

Results

In this section, we report on an implementation of the presented grounding algorithm for FO, called GIDL [1]. It is used as a front-end for the model generator MIDL (Mariën, Wittocx, and Denecker 2007). The GIDL system incorporates a lot more than the above sketched algorithm. Most importantly, it can ground FO(ID), an extension of FO with inductive definitions (Denecker 2000)[2]. Other features are support for a sorted vocabulary, for partial functions, arithmetic, aggregates, ... It is beyond the scope of this paper to discuss all of these in detail.

We implemented the basic grounding algorithm using the techniques of (Leone, Perri, and Scarcello 2004). The bounds are represented by binary decision diagrams (BDDs) for FO. We used the simplification algorithm for BDDs of (Goubault 1995). The stop criterion used for Algorithm 1 is simply a bound on the number of iterations of the while loop. To avoid a blow-up of the ct- and cf-bounds, each refinement step that would create a bound greater than a given size is not executed.

[1] GIDL binaries are downloadable from `http://www.cs.kuleuven.be/~dtai/krr/software/idp.html`.

[2] The algorithm of (Patterson et al. 2007) also serves to ground FO(ID), but until now, only a fragment of it has been implemented.

For theory T_1 of Example 1, our implementation of Algorithm 1 detects the fixpoint. For theory T_2 of Example 2, GIDL is able to compute, almost instantaneously, the mentioned bounds ψ_n for $Do(a_0, t_0)$ up till $n = 3$. However, for $n \geq 4$, it cannot compute ψ_n in a reasonable time, because the internal representation of the bounds becomes too large.

It is difficult to present objective results about the impact of the bounds on grounding size and time, as this strongly varies with the input theory T and structure I_σ. Indeed, when the most important bounds are already manually added to T, one cannot expect much improvement anymore. Also, for some theories the grounding size can drop orders of magnitude, depending on the input structure. E.g., for Example 1, the size of the grounding is strongly related to the density of the input graph. To nevertheless give an idea about the impact of Algorithm 1, we tested GIDL on 20 standard benchmark problems[3]. We allowed at most $(2 \times$ # subformulas in $T)$ iterations of the while-loop, and a maximum of 8 internal nodes in each BDD. The time for computing the bounds was always neglectable as it never exceeded 0.01 seconds. The table below shows the ratios of the size (time) of the resulting grounding to the size (time) when no bounds were used.

#	size	time	#	size	time	#	size	time
1	0.83	0.74	8	0.03	0.03	15	1.00	1.84
2	0.02	0.16	9	1.00	0.91	16	1.00	0.27
3	0.97	1.09	10	0.00	0.00	17	1.00	0.92
4	0.90	1.00	11	0.87	0.78	18	0.99	1.03
5	0.29	0.50	12	0.95	0.53	19	0.80	0.11
6	1.00	1.01	13	0.94	1.08	20	0.99	0.90
7	0.30	0.25	14	1.00	0.98			

The algorithm reduced the size of the grounding with at least 10% in nine of the twenty cases and led to a drastic reduction in five cases. More suprisingly, the use of the bounds led in only one case to a significant increase of the grounding time, produced at least 10% time reduction in half of the cases and more than 70% in 6 cases. Interestingly, we observed several cases where the grounding time decreased a lot while there was no significant size reduction. The explanation is that the algorithm spreads the information in the bounds over the whole theory, allowing the grounder to simplify at an earlier stage, and therefore leading to more efficient grounding.

Conclusions

We presented a method to create smaller groundings in the context of model expansion for FO. It consists of computing, independent of the input structure, upper and lower bounds for each subformula of the input theory. These bounds are then used to transform the theory such that its grounding is compacter and often more efficiently computable. Our implementation shows that the method is applicable in practice. We are currently exploring the use of the presented approximate reasoning techniques in other applications such as incomplete databases (under integrity constraints).

[3]Most problems were taken from `http://asparagus.cs.uni-potsdam.de`

Acknowledgments

Johan Wittocx is research assistant of the *Fonds voor Wetenschappelijk Onderzoek - Vlaanderen* (FWO Vlaanderen)

References

Claessen, K., and Sörensson, N. 2003. New techniques that improve MACE-style model finding. In *Proc. of Workshop on Model Computation (MODEL)*.

Denecker, M. 2000. Extending classical logic with inductive definitions. In Lloyd et al., J., ed., *CL'2000*, volume 1861 of *LNAI*, 703–717. Springer.

Enderton, H. B. 1972. *A Mathematical Introduction To Logic*. Academic Press.

Ganzinger, H., and Korovin, K. 2003. New directions in instantiation-based theorem proving. In Kolaitis, P., ed., *LICS-03*, 55–64.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *International Joint Conference and Symposium on Logic Programming (JICSLP'88)*, 1070–1080. MIT Press.

Goubault, J. 1995. A bdd-based simplification and skolemization procedure. *Logic Journal of IGPL* 3(6):827–855.

Leone, N.; Perri, S.; and Scarcello, F. 2004. Backjumping techniques for rules instantiation in the DLV system. In *NMR*, 258–266.

Manthey, R., and Bry, F. 1988. Satchmo: a theorem prover implemented in prolog. In *Proc. of the Conference on Automated Deduction*. Springer-Verlag.

Mariën, M.; Wittocx, J.; and Denecker, M. 2006. The IDP framework for declarative problem solving. In *Search and Logic: Answer Set Programming and SAT*, 19–34.

Mariën, M.; Wittocx, J.; and Denecker, M. 2007. Integrating inductive definitions in sat. In *LPAR*, 378–392.

McCune, W. 1994. A Davis-Putnam program and its application to finite first-order model search: quasigroup existence problems. Technical Report ANL/MCS-TM-194, Argonne National Laboratory.

Mitchell, D., and Ternovska, E. 2005. A framework for representing and solving NP search problems. In *AAAI'05*, 430–435. AAAI Press/MIT Press.

Mitchell, D.; Ternovska, E.; Hach, F.; and Mohebali, R. 2006. Model expansion as a framework for modelling and solving search problems. Tech. Rep. TR2006-24, SFU.

Patterson, M.; Liu, Y.; Ternovska, E.; Gupta, A. 2006. Grounding for Model Expansion in k-Guarded Formulas with Inductive Definitions. In *IJCAI'07*, 161–166.

Slaney, J. K. 1994. Finder: Finite domain enumerator - system description. In Bundy, A., ed., *CADE*, volume 814 of *LNCS*, 798–801. Springer.

Syrjänen, T. 1998. Implementation of local grounding for logic programs with stable model semantics. Tech. Rep. B18, Digital Systems Laboratory, HUT.

Zhang, J., and Zhang, H. 1996. System description: Generating models by sem. In *CADE-13*, 308–312. London, UK: Springer-Verlag.