Constraint Propagation for First-Order Logic and Inductive Definitions

JOHAN WITTOCX, MARC DENECKER, and MAURICE BRUYNOOGHE, KU Leuven

In Constraint Programming, constraint propagation is a basic component of constraint satisfaction solvers. Here we study constraint propagation as a basic form of inference in the context of first-order logic (FO) and extensions with inductive definitions (FO(ID)) and aggregates (FO(AGG)). In a first, semantic approach, a theory of propagators and constraint propagation is developed for theories in the context of three-valued interpretations. We present an algorithm with polynomial-time data complexity. We show that constraint propagation in this manner can be represented by a datalog program. In a second, symbolic approach, the semantic algorithm is lifted to a constraint propagation algorithm in *symbolic structures*, symbolic representations of classes of structures. The third part of the paper is an overview of existing and potential applications of constraint propagation for model generation, grounding, interactive search problems, approximate methods for $\exists\forall$SO problems, and approximate query answering in incomplete databases.

Categories and Subject Descriptors: I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Predicate Logic*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and Constraint Programming*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Aggregates, constraint propagation, first-order logic, inductive definitions

ACM Reference Format:
Wittocx, J., Denecker, M., Bruynooghe, M. 2012. Constraint propagation for first-order logic and inductive definitions ACM Trans. Comput. Logic V, N, Article A (January YYYY), 44 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

In the context of Constraint Satisfaction Problems (CSP), constraint propagation is the process of reducing domains of variables in a solution preserving way, where solutions are assigments of values to variables that satisfy constraints [Apt 2003]. In this paper, we develop a theory of constraint propagation in the context of classical first order logic FO and extensions of FO.

Constraint propagation as defined here, operates in the context of a vocabulary Σ – its symbols corresponds to the "constraint variables" in CSP – , a theory T over Σ - the "constraints" in CSP – and a partial, 3-valued Σ-structure $\tilde{I}$ – which expresses the "domains" for symbols of Σ. Constraint solving in this setting corresponds to computing models M of T that "complete" $\tilde{I}$; M is an assignment of values to Σ that satisfies T and is approximated by $\tilde{I}$. In this context, constraint propagation is the process of refining $\tilde{I}$

This research was supported by projects G.0357.06 and G.0489.10 of Research Foundation - Flanders (FWO-Vlaanderen) and by GOA/08/008 "Probabilistic Logic Learning".
Authors address: J. Wittocx, M. Denecker (corresponding author), and M. Bruynooghe, Department of Computer Science, Katholieke Universiteit Leuven, Belgium; email: marcd@cs.kuleuven.be.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

in a model-preserving way. That is, constraint propagation derives a new three-valued structure $\tilde{J}$ that is more precise than $\tilde{I}$ and approximates each model M of T that is approximated by $\tilde{I}$. This paper presents a theoretical framework for constraint propagation in FO and extensions of FO, semantical and symbolic algorithms, complexity results and applications. The framework has found application in several state of the art systems, prototypes and applications that will be presented in Section 7.

The work on constraint propagation presented here fits into a trend of increasing convergence in computational logic areas such as constraint programming (CP), propositional satisfiability (SAT), and certain subareas of knowledge representation and reasoning (KRR) including Answer Set Programming (ASP) and model generation and expansion for FO extensions. In CP, we witness the evolution towards more expressive languages with logic features such as OPL [Van Hentenryck 1999], ESSENCE [Frisch et al. 2008] and Zinc [Marriott et al. 2008]). In these languages, constraint "variables" range over complex objects - sets, arrays. In SAT, the need for more expressive languages led to the FO-based field of SAT modulo theories (SMT) [Nieuwenhuis et al. 2006].

Coming from a different direction, languages that had been designed as expressive knowledge representation languages, gave rise to new paradigms for solving search problems. In particular, the development of effective finite model generators for the Answer Set Programming language (ASP) [Gelfond and Lifschitz 1991] such as Smodels [Niemelä et al. 2000], DLV [Leone et al. 2002] and (later) Clasp [Gebser et al. 2007] led to propose model generation for ASP as a constraint programming paradigm [Marek and Truszczyński 1999; Niemelä 1999]. As already noted in [Marek and Truszczyński 1999], model generation for solving constraint problems is applicable in all logics with a model semantics. Not long after, Herbrand model generators aspps/psgrnd [East and Truszczynski 2001] and MiDl [Mariën et al. 2005] were built for (fragments of) FO extended with inductive definitions. Later Mitchell and Ternovska [2005] proposed to generalize Herbrand model generation into *model expansion* (MX): the generation of a Σ-model M of a theory T over Σ that expands a given σ-structure, for $\sigma \subseteq \Sigma$. Their framework aims, among others, to classify logics according to the complexity of the search problems that can be expressed in them. Model expansion, as we define it here in this paper, will be the problem of finding a model M of a given theory T over Σ that completes a partial Σ-structure $\tilde{I}$. This extends the initial setting, in the sense that a two-valued σ-structure can always be viewed as a partial Σ-structure but not vice versa. Recently MX solvers for (extensions of) FO were built such as IDP [Wittocx et al. 2008c] and the Enfragmo system [Aavani et al. 2012] that compete with the best solvers of ASP as shown in the ASP system competition [Calimeri et al. 2011]. Independently, this particular form of inference was also developed in the KodKod system [Torlak and Jackson 2007], a model expander for a variant of FO and used as the main inference tool for system verification using Alloy [Jackson 2002].

Given the tight connection between model expansion and constraint solving observed in the first paragraph of this introduction, what we witness here is a surprising convergence in applications, solver techniques and languages[1]. Model expansion is not just another way of solving search problems, it is the *logical view* on constraint solving. In a deep sense, CP languages are logics and CP systems are model expanders for them.

In this paper, by studying constraint propagation for extensions of classical first-order logic (FO) we push the convergence between these areas a step further. The primary goal of constraint propagation in CSP is as a component of constraint solvers.

[1]We refer to [Mitchell and Ternovska 2008] for a logical analysis of ESSENCE.

While constraint propagation in FO as defined here could be used in model expansion systems, at present it is certainly not its primary use. As we shall see, current applications of our framework can serve very different purposes. Let us illustrate this in the context of an interactive search problem presented in [Vlaeminck et al. 2009]. Consider a software system supporting university students to compose their curriculum by selecting certain didactic modules and courses. Such a selection needs to satisfy certain rules that can be collected in a background theory T. Assume that amongst others, the following integrity constraints are imposed on the selections:

$$\forall x \forall y \, (\textbf{\textit{MutExcl}}\,(x, y) \Rightarrow \neg(\textbf{\textit{Selected}}\,(x) \wedge \textbf{\textit{Selected}}\,(y))), \tag{1}$$

$$\exists m \, (\textbf{\textit{Module}}\,(m) \wedge \textbf{\textit{Selected}}\,(m)), \tag{2}$$

$$\forall c \, (\textbf{\textit{Course}}\,(c) \wedge \exists m \, (\textbf{\textit{Module}}\,(m) \wedge \textbf{\textit{Selected}}\,(m) \wedge \textbf{\textit{In}}\,(c, m)) \Rightarrow \textbf{\textit{Selected}}\,(c)). \tag{3}$$

The first constraint states that mutually exclusive components cannot be selected both, the second one expresses that at least one module should be taken and the third one ensures that all courses of a selected module are selected. Basically, a student running this system is manually solving a constraint problem, a model expansion problem with respect to this theory. At each intermediate stage, the courses that he may have selected or deselected, form a partial, three-valued structure. E.g., he may have selected the module m_1, two mutually exclusive courses c_1 and c_2, where c_1 belongs to the module m_1. One can check that in every total selection that completes this partial structure and satisfies the constraints, c_1 will be selected, c_2 will not be selected, and no module containing c_2 will be selected. Constraint propagation for FO can be used to derive these facts. By interleaving the choices by the user with constraint propagation on the background theory and returning the results as feedback to the user, the system supports the user by preventing him to avoid choices that will lead to inconsistency. As such, constraint propagation is as useful in such interactive constraint problems as it is in full constraint solving systems.

From this and other applications demonstrated further on, constraint propagation emerges as a valuable form of inference for FO in its own right. The contributions of the paper are the following. In Section 3 we present a theoretical framework for propagation and propagators in the context of theories T and three-valued structures $\tilde{I}$. In Section 4, we develop an algorithm that computes constraint propagation for finite theories T and structures $\tilde{I}$ in time polynomial in the size of $\tilde{I}$ and exponential in the size of T. This algorithm is based on a transformation of T to what will be called INF form. It is shown how to represent this algorithm as a datalog program. Many logic-based formalisms offer computational support for computing sets defined by such rule sets. Examples are (tabled) Prolog, Datalog, Answer Set Programming , FO extended with inductive definitions (FO(ID)), and production rule systems [Forgy 1979]. As a consequence, many of the theoretical and practical results obtained for these formalisms can be applied to study properties of our method, as well as to implement it efficiently.

The aforementioned algorithm requires a concrete finite partial structure $\tilde{I}$ as input. In a second part of the paper, Section 5, we lift the method to obtain a symbolic constraint propagation method that operates on *symbolic structures*, which are symbolic descriptions of classes of three-valued structures.

Both propagation on structures and the symbolic method are first developed in the context of FO theories. However, experience in Knowledge Representation and Constraint Programming shows that (inductive) definitions and aggregates are language constructs that are often crucial to model many real-life applications. Yet in general, these concepts cannot be effectively expressed in FO. To broaden the applicability of the propagation algorithm, we extend it to FO(ID), the extension of FO with inductive

definitions [Denecker and Ternovska 2008] in Section 6 and to FO(AGG), FO extended with aggregates (Appendix A).

In the final part of this paper, Section 7, we sketch several existing and potential applications of our propagation algorithms, namely for model generation, preprocessing for grounding, interactive configuration, approximate solving of ∀SO search problems and approximate query answering in incomplete databases.

This paper is an extended and improved presentation of [Wittocx et al. 2008a]. It describes (part of) the theoretical foundation for applications presented in [Wittocx et al. 2010; Wittocx et al. 2009; Vlaeminck et al. 2010; Vlaeminck et al. 2009]. A less densely written version of this paper is part of the PhD thesis of the first author [Wittocx 2010].

2. PRELIMINARIES

We assume the reader is familiar with classical first-order logic (FO). In this section, we introduce the notations and conventions used throughout this paper and we recall definitions and results about three- and four-valued structures and constraint satisfaction problems.

2.1. First-Order Logic

A *vocabulary* Σ is a set of predicate and function symbols, each with an associated arity. We often denote a symbol S with arity n by S/n. A Σ-*structure* I consists of a domain D, an assignment of a relation $P^I \subseteq D^n$ to each predicate symbol $P/n \in \Sigma$, and an assignment of a function $F^I : D^n \to D$ to each function symbol $F/n \in \Sigma$. The domain of I is denoted dom_I. The symbols $true$ and $false$ are logical propositional symbols that are true, respectively false in each structure. The equality symbol $=$ is a logical binary predicate symbol; in each structure, its interpretation is the identity relation of the domain of I. We use $t \neq s$ as a shorthand for $\neg(t = s)$. If I is a Σ-structure and $\Sigma' \subseteq \Sigma$, we denote by $I|_{\Sigma'}$ the restriction of I to the symbols of Σ'. Vice versa, we call I an expansion of $I|_{\Sigma'}$. If Σ_1 and Σ_2 are two disjoint vocabularies, I a Σ_1-structure with domain D, and J a Σ_2-structure with the same domain, then $I + J$ denotes the unique $(\Sigma_1 \cup \Sigma_2)$-structure with domain D such that $(I + J)|_{\Sigma_1} = I$ and $(I + J)|_{\Sigma_2} = J$.

For a vocabulary Σ, $\#(\Sigma)$ is its cardinality and $MaxAr(\Sigma)$ the maximal arity of a predicate in Σ. The *size* $|\tilde{I}|$ of a structure $\tilde{I}$ is defined as the cardinality of the domain of $\tilde{I}$. This definition is precise enough for the complexity results in this paper.

Variables are denoted by lowercase letters. We use $\overline{x}$, $\overline{y}$, ..., to denote both sets and tuples of variables. A *variable assignment* with domain D is a function mapping variables to domain elements in D. If θ is a variable assignment, x a variable and d a domain element, $\theta[x/d]$ denotes the variable assignment that maps x to d and corresponds to θ on all other variables. This notation is extended to tuples of variables and domain elements of the same length.

Terms and formulas over a vocabulary Σ are defined as usual. We use $(\varphi \Rightarrow \psi)$ and $(\varphi \Leftrightarrow \psi)$ as shorthands for, respectively, the formulas $(\neg\varphi \vee \psi)$ and $((\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi))$. If $\overline{x}$ and $\overline{y}$ are, respectively, the tuples of variables $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$, then $\overline{x} \neq \overline{y}$ is a shorthand for the formula $(x_1 \neq y_1) \vee \ldots \vee (x_n \neq y_n)$. Often, we denote a formula φ by $\varphi[\overline{x}]$ to indicate that $\overline{x}$ is precisely the set of free variables of φ. That is, if $y \in \overline{x}$, then y has at least one occurrence in φ outside the scope of quantifiers $\forall y$ and $\exists y$. A formula without free variables is called a *sentence*. A theory is a set of sentences. If φ is a formula, x a variable and t a term, then $\varphi[x/t]$ denotes the result of replacing all free occurrences of x in φ by t. This notation is extended to tuples of variables and terms of the same length.

For a theory T, its *size* $|T|$ is the number of symbol occurrences in T; $\#(T)$ is the number of formulas in T; $width(T)$ is the maximal number of free variables of subfor-

mulas of T; $MaxFSize(T)$ is the size of the largest formula of T. Size and width are defined also for formulas.

The evaluation $t^{I\theta}$ of a term t in the structure I under variable assignment θ, and the satisfaction $I\theta \models \varphi$ of a formula φ in I under θ is defined by the usual induction. Since $=$ denotes identity, we have $I\theta \models t = s$ iff $t^{I\theta} = s^{I\theta}$. If $\overline{x}$ are the free variables of a formula φ, the evaluation of φ under assignment $\theta[\overline{x}/\overline{d}]$ does not depend on θ. Therefore, we occasionally omit θ in $\tilde{I}\theta[\overline{x}/\overline{d}] \models \varphi$.

A *query* is an expression $\{\overline{x} \mid \varphi[\overline{y}]\}$, where φ is a formula and $\overline{y} \subseteq \overline{x}$. Such a query corresponds to the Boolean lambda expression $\lambda\overline{x}.\varphi[\overline{y}]$. The interpretation $\{\overline{x} \mid \varphi[\overline{y}]\}^I$ of query $\{\overline{x} \mid \varphi[\overline{y}]\}$ in structure I is the set $\{\overline{d} \mid I[\overline{x}/\overline{d}] \models \varphi\}$.

Definition 2.1 (Σ-*equivalent*). Let Σ_1 and Σ_2 be two vocabularies that share a common subvocabulary Σ and let φ_1 and φ_2 be sentences over, respectively, Σ_1 and Σ_2. Then φ_1 and φ_2 are Σ-*equivalent* if for any Σ-structure I, there exists an expansion M_1 of I to Σ_1 such that $M_1 \models \varphi_1$ iff there exists an expansion M_2 of I to Σ_2 such that $M_2 \models \varphi_2$. They are *one-to-one* Σ-*equivalent* if in addition each I has at most one expansion M_1 satisfying φ_1 and at most one expansion M_2 satisfying φ_2.

The following proposition presents a method to rewrite sentences to Σ-equivalent sentences. This rewriting method is called *predicate introduction*, and is applied in, e.g., the well-known Tseitin [1968] transformation.

PROPOSITION 2.2. *Let φ be a sentence over a vocabulary Σ and let $\psi[\overline{x}]$ be a subformula of φ with n free variables. Let P/n be a new predicate symbol and denote by φ' the result of replacing $\psi[\overline{x}]$ by $P(\overline{x})$ in φ. Then $\varphi' \wedge \forall\overline{x}(P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$ is one-to-one Σ-equivalent to φ.*

PROOF. Denote the vocabulary $\Sigma \cup \{P\}$ by Σ' and let I be a Σ-structure. Any expansion of I to Σ' that satisfies the sentence $\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$ necessarily assigns $\{\overline{x} \mid \psi[\overline{x}]\}^I$ to P. Hence, such an expansion satisfies φ' iff $I \models \varphi$. □

Below, we will often facilitate the presentation by assuming that vocabularies do not contain function symbols. The following proposition sketches a method to remove function symbols from a theory. For a given vocabulary Σ, we define its graph vocabulary Σ_G as consisting of all predicate symbols of Σ and a graph predicate symbol $P_F/n+1$ for each function symbol F/n. For each Σ-interpretation I, we define $Graph(I)$ as the Σ_G-interpretation which coincides with I on the domain and the interpretation of all predicate symbols and such that $(P_F)^{Graph(I)} = F^I$ for each function symbol F (recall that an n-ary function is an $n+1$-ary relation). A standard result is the following.

PROPOSITION 2.3. *[Ebbinghaus et al. 1984] Let T be a theory over a vocabulary Σ. Then there exists a theory T' over Σ_G such that $Graph(\cdot)$ is a one-to-one correspondence between models of T and models of T'. Moreover, if T is finite, T' can be constructed in linear time in the size of T.*

Theory T' is obtained from T by adding the sentences

$$\forall\overline{x}\exists y\ P_F(\overline{x}, y),$$
$$\forall\overline{x}\forall y_1\forall y_2\ (P_F(\overline{x}, y_1) \wedge P_F(\overline{x}, y_2) \Rightarrow y_1 = y_2),$$

for each of graph symbol P_F, by removing every nested function symbol in terms, in atoms and in the righthandside of equality atoms through iterated application of the following rewrite rule:

$$P(\dots f(\overline{t}) \dots) \longrightarrow \forall x(f(\overline{t}) = x \Rightarrow P(\dots x \dots))$$

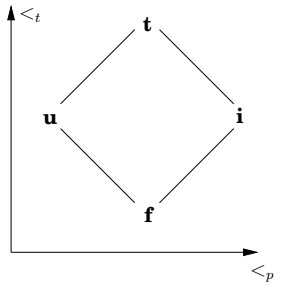

Fig. 1. The truth order $<_t$ and the precision order $<_p$. According to the truth axis, we have, e.g., $\mathbf{f} <_t \mathbf{u}$; according to the precision axis, we have, e.g., $\mathbf{u} <_p \mathbf{f}$.

where x is a fresh variable, and finally, by replacing all atoms of the form $F(\overline{x}) = y$ by $P_F(\overline{x}, y)$.

Example 2.4. Applying the sketched transformation on a theory containing the sentence *Selected* (C) (with C a constant), produces a theory containing the sentences

$$\exists y\ P_C(y);$$
$$\forall y_1 \forall y_2\ (P_C(y_1) \wedge P_C(y_2) \Rightarrow y_1 = y_2);$$
$$\forall x\ (P_C(x) \Rightarrow \textit{Selected}(x)).$$

2.2. Three- and Four-Valued Structures

In this subsection we present three- and four-valued structures. In these structures it is possible to express partial and inconsistent information. For the rest of the paper, we assume that vocabularies are function free.

2.2.1. Four-Valued Structures. Belnap [1977] introduced a four-valued logic with truth values *true*, *false*, *unknown*, and *inconsistent* which we denote by, respectively, **t**, **f**, **u** and **i**. For a truth value **v**, the *inverse value* $\mathbf{v}^{-1}$ is defined by $\mathbf{t}^{-1} = \mathbf{f}$, $\mathbf{f}^{-1} = \mathbf{t}$, $\mathbf{u}^{-1} = \mathbf{u}$ and $\mathbf{i}^{-1} = \mathbf{i}$. Belnap distinguished two orders, the *truth order* $<_t$ and the *precision order* $<_p$, also called the *knowledge order*. They are defined in Figure 1. The reflexive closure of these orders is denoted by $\leq_t$, respectively $\leq_p$.

Let Σ be a (function-free) vocabulary. A four-valued Σ-structure $\tilde{I}$ consists of a domain D and a function $P^{\tilde{I}} : D^n \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}, \mathbf{i}\}$ for every $P/n \in \Sigma$. We say that a four-valued structure $\tilde{I}$ is *three-valued* when $P^{\tilde{I}}(\overline{d}) \neq \mathbf{i}$ for any $P \in \Sigma$ and tuple of domain elements $\overline{d}$. A structure $\tilde{I}$ is *two-valued* when it is three-valued and $P^{\tilde{I}}(\overline{d}) \neq \mathbf{u}$ for every P and $\overline{d}$. We call a four-valued structure $\tilde{I}$ *strictly three-valued* if it is three-valued but not two-valued. Likewise, a structure is *strictly four-valued* if it is four-valued but not three-valued. A four-valued structure $\tilde{I}$ that is two-valued can be identified with the standard FO structure I for which for every predicate symbol P and tuple of domain elements $\overline{d}$, $\overline{d} \in P^I$ iff $P^{\tilde{I}}(\overline{d}) = \mathbf{t}$. In the rest of the paper, when we refer to a structure I (without tilde) we mean a two-valued structure, while $\tilde{I}$ means a four-valued structure (which possibly is three- or two-valued). In our examples, we will specify a four-valued function $P^{\tilde{I}}$ by writing $t : P^{\tilde{I}} = \{\text{set of tuples}\}$ and $f : P^{\tilde{I}} = \{\text{set of tuples}\}$. Tuples only in $t : P^{\tilde{I}}$ are true, tuples only in $f : P^{\tilde{I}}$ are false, tuples in both are inconsistent, and tuples in none are unknown. When P is two-valued in $\tilde{I}$, we write $P^{\tilde{I}} = \{\text{set of tuples}\}$.

Both truth and precision order extend to structures. E.g., if $\tilde{I}$ and $\tilde{J}$ are two Σ-structures, $\tilde{I} \leq_p \tilde{J}$ if for every predicate symbol P and tuple of domain elements $\overline{d}$, $P^{\tilde{I}}(\overline{d}) \leq_p P^{\tilde{J}}(\overline{d})$. The most precise Σ-structure with domain D is denoted by $\top^{\leq_p}_{\Sigma,D}$ and assigns $P^{\top^{\leq_p}_{\Sigma,D}}(\overline{d}) = \mathbf{i}$ to every predicate symbol $P/n \in \Sigma$ and $\overline{d} \in D^n$. Vice versa, the least precise structure $\bot^{\leq_p}_{\Sigma,D}$ assigns $P^{\bot^{\leq_p}_{\Sigma,D}}(\overline{d}) = \mathbf{u}$. We omit D and/or Σ from $\bot^{\leq_p}_{\Sigma,D}$ and $\top^{\leq_p}_{\Sigma,D}$ if they are clear from the context. If a two-valued structure I is more precise than a three-valued structure $\tilde{I}$, we say that $\tilde{I}$ *approximates* I and that I *completes* or *refines* $\tilde{I}$.

Remark. We here use three-valued structures as partial structures, structures that do not define the value of all domain atoms. Thus, **u** is viewed as "undefined". In the same spirit, when $\Sigma \subseteq \Sigma'$, we may use a Σ-interpretation $\tilde{I}$ in a context where a Σ'-structure is required, in which case $\tilde{I} + \bot^{\leq_p}_{\Sigma' \setminus \Sigma, dom_{\tilde{I}}}$ is intended.

A *domain atom* of a structure $\tilde{I}$ with domain D is a pair of an n-tuple $\overline{d}$ of domain elements and an n-ary predicate symbol. Sligtly abusing notation, we denote such a domain atom by $P(\overline{d})$. For a truth value **v** and domain atom $P(\overline{d})$, we denote by $\tilde{I}[P(\overline{d})/\mathbf{v}]$ the structure that assigns $P^{\tilde{I}}(\overline{d}) = \mathbf{v}$ and corresponds to $\tilde{I}$ on the rest of the vocabulary.

A *domain literal* is a domain atom $P(\overline{d})$ or the negation $\neg P(\overline{d})$ of a domain atom. By $\tilde{I}[\neg P(\overline{d})/\mathbf{v}]$ we denote the structure $\tilde{I}[P(\overline{d})/\mathbf{v}^{-1}]$. This notation is extended to sets of domain literals: if U is a set of domain literals, $\tilde{I}[U/\mathbf{v}]$ denotes the structure identical to I except that it assigns **v** to all domain literals of U.

The value of a formula φ in a four-valued structure $\tilde{I}$ with domain D under variable assignment θ is defined by structural induction:

— $\tilde{I}\theta(P(\overline{x})) = P^{\tilde{I}}(\theta(\overline{x}))$;
— $\tilde{I}\theta(\neg\varphi) = (\tilde{I}\theta(\varphi))^{-1}$;
— $\tilde{I}\theta(\varphi \wedge \psi) = \mathrm{glb}_{\leq_t}\{\tilde{I}\theta(\varphi), \tilde{I}\theta(\psi)\}$;
— $\tilde{I}\theta(\varphi \vee \psi) = \mathrm{lub}_{\leq_t}\{\tilde{I}\theta(\varphi), \tilde{I}\theta(\psi)\}$;
— $\tilde{I}\theta(\forall x\ \varphi) = \mathrm{glb}_{\leq_t}\{\tilde{I}\theta[x/d](\varphi) \mid d \in D\}$;
— $\tilde{I}\theta(\exists x\ \varphi) = \mathrm{lub}_{\leq_t}\{\tilde{I}\theta[x/d](\varphi) \mid d \in D\}$.

When $\tilde{I}$ is a three-valued structure, this corresponds to the standard Kleene [1952] semantics.

If $\tilde{I}$ is three-valued, then $\tilde{I}\theta(\varphi) \neq \mathbf{i}$ for every formula φ and variable assignment θ. If $\tilde{I}$ is two-valued, then $\tilde{I}\theta(\varphi) \in \{\mathbf{t}, \mathbf{f}\}$. Also, if $\tilde{I}$ is two-valued, then $\tilde{I}\theta(\varphi) = \mathbf{t}$ if $\tilde{I}\theta \models \varphi$ and **f** otherwise. We omit θ and/or $[\overline{x}/\overline{d}]$ from an expression of the form $\tilde{I}\theta[\overline{x}/\overline{d}](\varphi)$ when they are irrelevant.

If φ is a formula and $\tilde{I}$ and $\tilde{J}$ are two structures such that $\tilde{I} \leq_p \tilde{J}$, then also $\tilde{I}\theta(\varphi) \leq_p \tilde{J}\theta(\varphi)$ for every θ. If φ is a formula that does not contain negations and $\tilde{I} \leq_t \tilde{J}$, then also $\tilde{I}\theta(\varphi) \leq_t \tilde{J}\theta(\varphi)$ for every θ.

By Proposition 2.3, the restriction to a function free vocabulary is without loss of generality. The reader interested in more details about functions can consult [Wittocx 2010].

2.2.2. Encoding Four-Valued Structures by Two-Valued Structures. A standard way to encode a four-valued structure $\tilde{I}$ over a vocabulary Σ is by a two-valued structure $\mathrm{tf}(\tilde{I})$ over a vocabulary $\mathrm{tf}(\Sigma)$ containing two symbols P_{ct} and P_{cf} for each symbol $P \in \Sigma$. The inter-

pretation of P_{ct}, respectively P_{cf}, in $\mathrm{tf}(\tilde{I})$ represents what is certainly true, respectively certainly false, in $P^{\tilde{I}}$. Formally, for a vocabulary Σ and Σ-structure $\tilde{I}$, $\mathrm{tf}(\Sigma)$ denotes the vocabulary $\{P_{\mathrm{ct}}/n \mid P/n \in \Sigma\} \cup \{P_{\mathrm{cf}}/n \mid P/n \in \Sigma\}$ and the $\mathrm{tf}(\Sigma)$-structure $\mathrm{tf}(\tilde{I})$ is defined by $(P_{\mathrm{ct}})^{\mathrm{tf}(\tilde{I})} = \{\overline{d} \mid P^{\tilde{I}}(\overline{d}) \geq_p \mathbf{t}\}$ and $(P_{\mathrm{cf}})^{\mathrm{tf}(\tilde{I})} = \{\overline{d} \mid P^{\tilde{I}}(\overline{d}) \geq_p \mathbf{f}\}$ for every $P \in \Sigma$.

Observe that $\tilde{I}$ is three-valued iff $(P_{\mathrm{ct}})^{\mathrm{tf}(\tilde{I})}$ and $(P_{\mathrm{cf}})^{\mathrm{tf}(\tilde{I})}$ are disjoint for any $P \in \Sigma$; $\tilde{I}$ is two-valued iff for every $P/n \in \Sigma$, $(P_{\mathrm{ct}})^{\mathrm{tf}(\tilde{I})}$ and $(P_{\mathrm{cf}})^{\mathrm{tf}(\tilde{I})}$ are each other's complement in D^n. Also, if $\tilde{I} \leq_p \tilde{J}$, then $(P_{\mathrm{ct}})^{\mathrm{tf}(\tilde{I})} \subseteq (P_{\mathrm{ct}})^{\mathrm{tf}(\tilde{J})}$ and $(P_{\mathrm{cf}})^{\mathrm{tf}(\tilde{I})} \subseteq (P_{\mathrm{cf}})^{\mathrm{tf}(\tilde{J})}$. Therefore $\tilde{I} \leq_p \tilde{J}$ iff $\mathrm{tf}(\tilde{I}) \leq_t \mathrm{tf}(\tilde{J})$.

The value of a formula φ in a structure $\tilde{I}$ can be obtained by computing the value of two formulas over $\mathrm{tf}(\Sigma)$ in $\mathrm{tf}(\tilde{I})$. Define for a formula φ over Σ the formulas φ_{ct} and φ_{cf} over $\mathrm{tf}(\Sigma)$ by simultaneous induction:

— $(P(\overline{x}))_{\mathrm{ct}} = P_{\mathrm{ct}}(\overline{x})$ and $(P(\overline{x}))_{\mathrm{cf}} = P_{\mathrm{cf}}(\overline{x})$;
— $(\neg\varphi)_{\mathrm{ct}} = \varphi_{\mathrm{cf}}$ and $(\neg\varphi)_{\mathrm{cf}} = \varphi_{\mathrm{ct}}$;
— $(\varphi \wedge \psi)_{\mathrm{ct}} = \varphi_{\mathrm{ct}} \wedge \psi_{\mathrm{ct}}$ and $(\varphi \wedge \psi)_{\mathrm{cf}} = \varphi_{\mathrm{cf}} \vee \psi_{\mathrm{cf}}$;
— $(\varphi \vee \psi)_{\mathrm{ct}} = \varphi_{\mathrm{ct}} \vee \psi_{\mathrm{ct}}$ and $(\varphi \vee \psi)_{\mathrm{cf}} = \varphi_{\mathrm{cf}} \wedge \psi_{\mathrm{cf}}$;
— $(\forall x\ \varphi)_{\mathrm{ct}} = \forall x\ \varphi_{\mathrm{ct}}$ and $(\forall x\ \varphi)_{\mathrm{cf}} = \exists x\ \varphi_{\mathrm{cf}}$;
— $(\exists x\ \varphi)_{\mathrm{ct}} = \exists x\ \varphi_{\mathrm{ct}}$ and $(\exists x\ \varphi)_{\mathrm{cf}} = \forall x\ \varphi_{\mathrm{cf}}$.

The intuition is that φ_{ct} denotes a formula that is true iff φ is certainly true while φ_{cf} is a formula that is true iff φ is certainly false. This explains, e.g., the definition $(\neg\varphi)_{\mathrm{ct}} = \varphi_{\mathrm{cf}}$: $\neg\varphi$ is certainly true iff φ is certainly false. As another example, $(\varphi \wedge \psi)_{\mathrm{cf}} = \varphi_{\mathrm{cf}} \vee \psi_{\mathrm{cf}}$ states that $(\varphi \wedge \psi)$ is certainly false if φ or ψ is certainly false.

For a pair of formulas (φ_1, φ_2), a structure I and variable assignment θ, we denote the pair of truth values $(I\theta(\varphi_1), I\theta(\varphi_2))$ by $I\theta(\varphi_1, \varphi_2)$. We identify the pairs $(\mathbf{t}, \mathbf{f})$, $(\mathbf{f}, \mathbf{t})$, $(\mathbf{f}, \mathbf{f})$ and $(\mathbf{t}, \mathbf{t})$ with, respectively, the truth values **t**, **f**, **u** and **i**. Intuitively, the first value in the pairs states whether something is certainly true, the second value whether it is certainly false. It follows that, e.g., $(\mathbf{t}, \mathbf{f})$ corresponds to saying that something is certainly true and not certainly false and therefore identifies with **t**. Using these equalities, the next proposition expresses that the value of a formula in a four-valued structure $\tilde{I}$ can be computed by evaluating φ_{ct} and φ_{cf} in the two-valued structure $\mathrm{tf}(\tilde{I})$.

PROPOSITION 2.5 ([FEFERMAN 1984]). *For every formula φ, structure $\tilde{I}$, and variable assignment θ, $\tilde{I}\theta(\varphi) = \mathrm{tf}(\tilde{I})\theta(\varphi_{\mathrm{ct}}, \varphi_{\mathrm{cf}})$.*

It follows from Proposition 2.5 that model checking and query answering algorithms for FO can be lifted to four-valued logic and that complexity results carry over. Evaluating a formula φ in a 2-valued finite structure I takes time $\mathcal{O}(|I|^{|\varphi|})$ [Grädel et al. 2007]. For finite four-valued structures $\tilde{I}$, computing $\tilde{I}\theta(\varphi)$ has the same complexity and takes time polynomial in $|\tilde{I}|$ and exponential in $|\varphi|$.

Another interesting property of the formulas φ_{ct} and φ_{cf} is stated in the following proposition.

PROPOSITION 2.6. *For every formula φ, φ_{ct} and φ_{cf} are positive formulas (i.e., do not contain $\neg$).*

PROOF. Follows directly from the definition of φ_{ct} and φ_{cf}. □

2.3. Constraint Programming

We now recall some definitions from Constraint Programming (CP). Our terminology is based on the book of Apt [2003] to which we refer for a comprehensive introduction to CP.

Let S be a sequence $(v_1, \ldots, v_n)$ of variables. A *constraint* on S is a set of n-tuples. A *constraint satisfaction problem* (CSP) is a tuple $\langle \mathcal{C}, V, \boldsymbol{DOM} \rangle$ of a set V of variables, a mapping $\boldsymbol{DOM}$ of variables in V to domains, and a set $\mathcal{C}$ of constraints on finite sequences of variables from V. A *solution* to $\langle \mathcal{C}, V, \boldsymbol{DOM} \rangle$ is a function d, mapping each variable v of V to a value $d(v) \in \boldsymbol{dom}(v)$ such that $(d(v_1), \ldots, d(v_n)) \in C$ for each constraint $C \in \mathcal{C}$ on sequence $(v_1, \ldots, v_n)$. Two CSPs sharing the same variables are called *equivalent* if they have the same solutions.

A *propagator* (also called a *constraint solver*) is a function mapping CSPs to equivalent CSPs. A propagator is called *domain reducing* if it retains the constraints of a CSP and does not increase its domains. That is, if propagator O is domain reducing and $O(\langle \mathcal{C}_1, V, \boldsymbol{DOM}_1 \rangle) = \langle \mathcal{C}_2, V, \boldsymbol{DOM}_2 \rangle$, then $\mathcal{C}_2 = \mathcal{C}_1$ and for every $v \in V$, $\boldsymbol{DOM}_2(v) \subseteq \boldsymbol{DOM}_1(v)$. In this paper, we only consider domain reducing propagators.

One straightforward way to state a constraint satisfaction problem using logic is as follows.Given a constraint satisfaction problem $\langle \mathcal{C}, V, \boldsymbol{DOM} \rangle$, we introduce a logic vocabulary Σ consisting of constants c_v and unary predicates P_v for every constraint variable v, and predicate symbol P_C/n for every n-ary constraint C. Define a $\Sigma \setminus \{c_{v_1}, \ldots, c_{v_n}\}$-structure I with domain $\boldsymbol{dom}_I = \bigcup_{v \in V} \boldsymbol{DOM}(v)$, $(P_v)^I = \boldsymbol{DOM}(v)$ and $(P_C)^I = C$, for every constraint $C \in \mathcal{C}$. Finally we define $T = \{P_v(c_v) \mid v \in V\} \cup \{P_C(c_{v_1}, \ldots, c_{v_n}) \mid \text{for each constraint } C \in \mathcal{C} \text{ on } S = (v_1, \ldots, v_n)\}$. Solutions of the constraint satisfaction problem correspond one-to-one to models of T expanding I.

Note that "variables" in CSP naturally correspond to constants in logic. To avoid confusion between variables in the context of FO and variables of a CSP, we call the latter *constraint variables* in the rest of this paper.

3. CONSTRAINT PROPAGATION FOR LOGIC THEORIES

This section starts with introducing the notion of propagation in the context of FO; it then makes the connection with CSP's, discusses refinement sequences and finishes with an analysis of complete propagators.

3.1. Model Expansion and Constraint Propagation Inference for FO

As explained in the introduction, the following definition of model expansion slightly extends that of [Mitchell and Ternovska 2005].

Definition 3.1 (*Model expansion*). For a theory T over Σ, and a (four-valued) structure $\tilde{I}$, a model expansion of $\tilde{I}$ in T is a (two-valued) model M of T such that $\tilde{I} \leq_p M$.

The topic of this paper is not model expansion but constraint propagation. The aim of constraint propagation is improve the precision of three-valued structures while preserving the approximated models.

Definition 3.2 (*Propagation*). A structure $\tilde{J}$ is a *propagation* of a theory T over Σ from a Σ-structure $\tilde{I}$ if $\tilde{I} \leq_p \tilde{J}$ and for each model M of T, if $\tilde{I} \leq_p M$ then $\tilde{J} \leq_p M$.

Definition 3.3 (*Propagator*). We call an operator O on the class of four-valued Σ-structures a *propagator for* T if for all $\tilde{I}$, $O(\tilde{I})$ is a propagation of T from $\tilde{I}$.

Propagators meet the following two conditions:

(1) O is *inflationary* with respect to $\leq_p$. That is, $\tilde{I} \leq_p O(\tilde{I})$ for every structure $\tilde{I}$.

(2) For every model M of T such that $\tilde{I} \leq_p M$, also $O(\tilde{I}) \leq_p M$.

These conditions are the translation to our setting of the requirements for a "domain reduction function" in [Apt 1999a]. The first condition states that by applying an operator no information is lost. The second condition states that no models of T approximated by $\tilde{I}$ are lost. Note that for a propagator O it follows from the definition above that $\tilde{I}$ and $O(\tilde{I})$ must have the same domain.

The composition of two propagators is a propagator. Other useful properties for a propagator are the following. A propagator O is called *monotone* if for all structures $\tilde{I}$ and $\tilde{J}$ such that $\tilde{I} \leq_p \tilde{J}$, also $O(\tilde{I}) \leq_p O(\tilde{J})$ holds. A propagator O for T is *inducing* for T if for every two-valued structure I such that $I \not\models T$, $O(I)$ is strictly four-valued, i.e., the operator recognizes that I is not a model and assigns **i** to at least one domain element. A propagator O for T is *idempotent* if for every structure $\tilde{I}, O(O(\tilde{I})) = O(\tilde{I})$.

Most propagators defined in this paper are monotone and idempotent. An example is the *inconsistency propagator* INCO which propagates inconsistency in at least one domain atom to total inconsistency. Formally,

$$\text{INCO}(\tilde{I}) = \begin{cases} \tilde{I} & \text{if } \tilde{I} \text{ is three-valued} \\ \top^{\leq_p} & \text{otherwise} \end{cases}$$

LEMMA 3.4. *If T_1 and T_2 are theories over the same vocabulary, O_1 is a propagator for T_1 and O_2 a propagator for T_2, then $O_1 \circ O_2$ is a propagator for $T_1 \cup T_2$.*

PROOF. Since O_1 and O_2 are propagators, $\tilde{I} \leq_p O_2(\tilde{I}) \leq_p O_1(O_2(\tilde{I})) = (O_1 \circ O_2)(\tilde{I})$ for every structure $\tilde{I}$. If $J \models T_1 \cup T_2$ and $\tilde{I} \leq_p J$, then $O_2(\tilde{I}) \leq_p J$ and therefore also $O_1(O_2(\tilde{I})) \leq_p J$. Hence $O_1 \circ O_2$ is a propagator. □

It is easy to check that the composition of two monotone propagators is a monotone propagator. Also, if O_1 is inducing for T_1 and O_2 is inducing for T_2, then $O_1 \circ O_2$ is inducing for $T_1 \cup T_2$.

From the definition of propagator, it follows that two logically equivalent theories have the same propagators. Let Σ and Σ' be two vocabularies such that $\Sigma \subseteq \Sigma'$ and let T and T' be Σ-equivalent theories over Σ, respectively Σ'. Recall that a Σ-structure $\tilde{I}$ can be seen as a Σ'-structure $\tilde{I} + \bot^{\leq_p}_{\Sigma' \setminus \Sigma}$, and hence any operator O on Σ'-structures can be applied to Σ-structures. We have the following property.

PROPOSITION 3.5. *Let O' be an operator on Σ'-structures and define the operator O on Σ-structures by $O(\tilde{I}) = O'(\tilde{I})|_\Sigma$ for any Σ-structure $\tilde{I}$. If O' is a propagator for T', then O is a propagator for T. If O' is monotone, then O is monotone as well.*

PROOF. Let O' be a propagator for T'. Then for every Σ-structure $\tilde{I}$, $\tilde{I} \leq_p O'(\tilde{I})|_\Sigma = O(\tilde{I})$. If J is a model of T such that $\tilde{I} \leq_p J$, then there exists an expansion J' of J to Σ' such that $J' \models T'$. Because O' is a propagator, $O'(\tilde{I}) \leq_p J'$ and therefore $O(\tilde{I}) \leq_p J$. We conclude that O is a propagator. It is straightforward to check that if O' is monotone, O is also monotone. □

3.2. Relationship to CSP

We already showed that CSP's have a natural expression as model expansion problems in logic. The inverse holds as well: model expansion problems for finite structures $\tilde{I}$ correspond to CSP's.

Let T be a logic theory over vocabulary Σ and $\tilde{I}$ a four-valued Σ-structure with domain D. If $\tilde{I}$ is finite, then the pair $\langle T, \tilde{I} \rangle$ has a corresponding CSP which is denoted by

$\langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{\tilde{I}} \rangle$ and defined as follows. The set of constraint variables $V_{\tilde{I}}$ is defined as the set of all domain atoms over Σ and D. We assume a fixed total order on $V_{\tilde{I}}$ and call the ith element in that order the ith domain atom. The domain $\boldsymbol{DOM}_{\tilde{I}}(P(\overline{d}))$ associated to domain atom $P(\overline{d})$ is defined by

$$\boldsymbol{DOM}_{\tilde{I}}(P(\overline{d})) = \begin{cases} \{\mathbf{t}, \mathbf{f}\} & \text{if } P^{\tilde{I}}(\overline{d}) = \mathbf{u}, \\ \{\mathbf{t}\} & \text{if } P^{\tilde{I}}(\overline{d}) = \mathbf{t}, \\ \{\mathbf{f}\} & \text{if } P^{\tilde{I}}(\overline{d}) = \mathbf{f}, \\ \emptyset & \text{if } P^{\tilde{I}}(\overline{d}) = \mathbf{i}. \end{cases}$$

Given a tuple $\overline{\mathbf{v}} \in \{\mathbf{t}, \mathbf{f}\}^{|V_{\tilde{I}}|}$, $I_{\overline{\mathbf{v}}}$ denotes the Σ-structure with domain D such that for every P and $\overline{d}$, $\overline{d} \in P^{I_{\overline{\mathbf{v}}}}$ iff $P(\overline{d})$ is the ith domain atom and the ith truth value in $\overline{\mathbf{v}}$ is $\mathbf{t}$. Finally, $\mathcal{C}_T$ is the singleton set containing the constraint that consists of the set of tuples $\overline{\mathbf{v}} \in \{\mathbf{t}, \mathbf{f}\}^{|V_{\tilde{I}}|}$ such that $I_{\overline{\mathbf{v}}} \models T$. It follows immediately that $\overline{\mathbf{v}}$ is a solution to $\langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{\tilde{I}} \rangle$ iff $I_{\overline{\mathbf{v}}} \models T$ and $\tilde{I} \leq_p I_{\overline{\mathbf{v}}}$.

The following proposition relates the definition of a propagator for T to the definition of propagator in the context of CP.

PROPOSITION 3.6. *Let O be a propagator for T, D a finite set, and $CSP_{T,D}$ the class of all $CSP_{\tilde{I}} = \langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{\tilde{I}} \rangle$ of all Σ-structures $\tilde{I}$ with domain D. Then the operator f on $CSP_{T,D}$ defined by $f(\langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{\tilde{I}} \rangle) = \langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{O(\tilde{I})} \rangle$, is a domain reducing propagator.*

PROOF. Any two $CSP_{\tilde{I}}, CSP_{\tilde{J}} \in CSP_{T,D}$ are based on structures with the same domain atoms, hence $V_{\tilde{I}} = V_{\tilde{J}}$. Also, $\tilde{I} \leq_p \tilde{J}$ iff $\boldsymbol{DOM}_{\tilde{I}}(P(\overline{d})) \supseteq \boldsymbol{DOM}_{\tilde{J}}(P(\overline{d}))$ for every domain atom $P(\overline{d})$. Therefore, f is domain reducing iff $O(\tilde{I}) \geq_p \tilde{I}$ for every structure $\tilde{I}$.

Function f is a propagator iff $\langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{\tilde{I}} \rangle$ and $\langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{O(\tilde{I})} \rangle$ have the same solutions. Because of the correspondence between models of T approximated by $\tilde{I}$, respectively $O(\tilde{I})$, and solutions of $\langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{\tilde{I}} \rangle$, respectively $\langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{O(\tilde{I})} \rangle$, it follows that f is a propagator iff the models of T approximated by $\tilde{I}$ are precisely the models of T approximated by $O(\tilde{I})$.

We conclude that O is a propagator for T iff f is a domain reducing propagator for CSPs of the form $\langle \mathcal{C}_T, V_{\tilde{I}}, \boldsymbol{DOM}_{\tilde{I}} \rangle$. □

3.3. Refinement Sequences

If V is a set of propagators for a theory T, Lemma 3.4 ensures that constraint propagation for T can be performed by starting from $\tilde{I}$ and successively applying propagators from V. We then get a sequence of increasingly precise four-valued structures. If such a sequence is strictly increasing in precision, we call it a *V-refinement sequence from $\tilde{I}$*.

Definition 3.7 (*Refinement sequence*). Let V be a set of propagators for T. With α an ordinal, we call a (possibly transfinite) sequence $\langle \tilde{J}_\xi \rangle_{0 \leq \xi \leq \alpha}$ of four-valued structures a *V-refinement sequence from $\tilde{I}$* if

— $\tilde{J}_0 = \tilde{I}$,
— for every successor ordinal $\xi + 1 \leq \alpha$: $\tilde{J}_{\xi+1} = O(\tilde{J}_\xi)$ for some $O \in V$,
— for every successor ordinal $\xi + 1 \leq \alpha$: $\tilde{J}_\xi <_p \tilde{J}_{\xi+1}$, and,
— for every limit ordinal $0 < \lambda \leq \alpha$: $\tilde{J}_\lambda = \mathrm{lub}_{\leq_p}(\{\tilde{J}_\xi \mid \xi < \lambda\})$.

In the CP literature, refinement sequences are sometimes called *derivations*, and constructing a derivation is called *constraint propagation*. Since refinement sequences

are strictly increasing in precision, it follows that every refinement sequence from a finite structure $\tilde{I}$ is finite. Moreover, the maximum length is a function of the number of predicate symbols ($\#(\Sigma)$), the maximum arity of a predicate symbol ($MaxAr(\Sigma)$) and the number of elements in the domain of the structure ($|\tilde{I}|$).

PROPOSITION 3.8. *For any set of propagators V, the length of a V-refinement sequence from a finite structure $\tilde{I}$ is less than* $2 \cdot |\tilde{I}|^{MaxAr(\Sigma)} \cdot \#(\Sigma)$.

PROOF. Recall that we defined $|\tilde{I}|$ as the cardinality of the domain of $\tilde{I}$. In any strictly increasing sequence of partial interpretations, at least one domain atom changes value with each next element. Each domain atom $P(\bar{d})$ can change value at most two times (from **u** to **t** to **i**, or from **u** to **f** to **i**). There are less than $|\tilde{I}|^{MaxAr(\Sigma)} \cdot \#(\Sigma)$ domain atoms. Thus, the maximal length is $2 \cdot |\tilde{I}|^{MaxAr(\Sigma)} \cdot \#(\Sigma)$. □

A refinement sequence is *stabilizing* if it cannot be extended anymore. The last structure in a stabilizing refinement sequence is called the *limit* of the sequence. A well-known result (see, e.g., Lemma 7.8 in [Apt 2003]) states:

PROPOSITION 3.9. *Let V be a set of monotone propagators for T and let $\tilde{I}$ be a structure. Then every stabilizing V-refinement sequence from $\tilde{I}$ has the same limit.*

PROOF. Let $\langle \tilde{J}_\xi \rangle_{0 \leq \xi \leq \alpha}$ and $\langle \tilde{K}_\xi \rangle_{0 \leq \xi \leq \beta}$ be two stabilizing V-refinement sequences from $\tilde{I}$. Let $\langle \tilde{L}_\xi \rangle_{0 \leq \xi \leq \beta}$ be the sequence of structures defined by

— $\tilde{L}_0 = \tilde{J}_\alpha$,
— For every successor ordinal $\xi + 1 \leq \beta$: $\tilde{L}_{\xi+1} = O(\tilde{L}_\xi)$ where O is a propagator from V such that $\tilde{K}_{\xi+1} = O(\tilde{K}_\xi)$,
— For every limit ordinal $0 < \lambda \leq \beta$: $\tilde{L}_\lambda = \mathrm{lub}_{\leq_p}(\{\tilde{L}_\xi \mid \xi < \lambda\})$.

Because $\langle \tilde{J}_\xi \rangle_{0 \leq \xi \leq \alpha}$ is stabilizing, it follows that $\tilde{L}_\beta = \tilde{J}_\alpha$. Since $\tilde{I} \leq_p \tilde{J}_\alpha$, we have that $\tilde{K}_\beta \leq_p \tilde{L}_\beta$. Hence, we obtain that $\tilde{K}_\beta \leq_p \tilde{J}_\alpha$. Similarly, we can derive that $\tilde{J}_\alpha \leq_p \tilde{K}_\beta$. Hence $\tilde{J}_\alpha = \tilde{K}_\beta$. It follows that every stabilizing V-refinement sequence from $\tilde{I}$ has the same limit, namely $\tilde{J}_\alpha$. □

If V only contains monotone propagators, we denote by $\lim_V$ the operator that maps every structure $\tilde{I}$ to the unique limit of any stabilizing V-refinement sequence from $\tilde{I}$. From Lemma 3.4 it follows that $\lim_V$ is a propagator.

Besides monotonicity, other properties of propagators, e.g., idempotence, may be taken into account by algorithms to efficiently construct refinement sequences. Apt [1999a] provides a general overview of such properties and algorithms.

3.4. Complete Propagators

The *complete propagator* for a theory T is the propagator that yields the most precise structures. This propagator is denoted by $\mathscr{O}^T$ and defined by

$$\mathscr{O}^T(\tilde{I}) = \mathrm{glb}_{\leq_p}\left(\{M \mid \tilde{I} \leq_p M \text{ and } M \models T\}\right).$$

Observe that if T has no models approximated by $\tilde{I}$, then $\mathscr{O}^T(\tilde{I}) = \top^{\leq_p}$. This is the case if $\tilde{I}$ is strictly four-valued. Otherwise, $\mathscr{O}^T(\tilde{I})$ is the partial structure that extends $\tilde{I}$ with truth values agreed by all models $M \geq_p \tilde{I}$.

The following properties hold for $\mathscr{O}^T$:

PROPOSITION 3.10. *For every theory T, $\mathscr{O}^T$ is a monotone, inducing and idempotent propagator.*

PROOF. Monotoniticy follows from the fact that $\{M \mid \tilde{I} \leq_p M \text{ and } M \models T\}$ is a superset of $\{M \mid \tilde{J} \leq_p M \text{ and } M \models T\}$ if $\tilde{I} \leq_p \tilde{J}$. The other two properties are straightforward as well. □

PROPOSITION 3.11. *Let T be a theory and $\tilde{I}$ a structure. Then, for any propagator O of T: $O(\tilde{I}) \leq_p \mathscr{O}^T(\tilde{I})$. That is, $\mathscr{O}^T$ is the most precise propagator.*

PROOF. To prove the proposition, we show that $P^{O(\tilde{I})}(\overline{d}) \leq_p P^{\mathscr{O}^T(\tilde{I})}(\overline{d})$ for any domain atom $P(\overline{d})$. If $P^{O(\tilde{I})}(\overline{d}) = \mathbf{i}$, it follows from the fact that O is a propagator that there is no model of T approximated by $\tilde{I}$. From the definition of $\mathscr{O}^T$, we conclude that also $P^{\mathscr{O}^T(\tilde{I})}(\overline{d}) = \mathbf{i}$. If on the other hand $P^{O(\tilde{I})}(\overline{d}) = \mathbf{t}$ or $P^{O(\tilde{I})}(\overline{d}) = \mathbf{f}$, then $P(\overline{d})$ is true, respectively false, in every model of T approximated by $\tilde{I}$. Therefore $P^{\mathscr{O}^T(\tilde{I})}(\overline{d}) \geq_p \mathbf{t}$, respectively $P^{\mathscr{O}^T(\tilde{I})}(\overline{d}) \geq_p \mathbf{f}$, in this case. It follows that $P^{O(\tilde{I})}(\overline{d}) \leq_p \mathscr{O}^T(\tilde{I})(\overline{d})$ for every domain atom of the form $P(\overline{d})$. □

Example 3.12. Let $\Sigma = \{\textit{Module}/1, \textit{Course}/1, \textit{Selected}/1, \textit{In}/2, \textit{MutExcl}/2\}$ and let $\tilde{I}_0$ be the Σ-structure with domain $\{m_1, m_2, c_1, c_2, c_3, c_4\}$ that is two-valued on all symbols except *Selected*, and that is given by (remind, we only show the true tuples for two-valued predicates):

$$\textit{Module}^{\tilde{I}_0} = \{m_1, m_2\}, \qquad \textit{MutExcl}^{\tilde{I}_0} = \{(c_1, c_2)\},$$
$$t:\textit{Selected}^{\tilde{I}_0} = \{c_1\}, \qquad f:\textit{Selected}^{\tilde{I}_0} = \emptyset,$$
$$\textit{In}^{\tilde{I}_0} = \{(c_1, m_1), (c_3, m_1), (c_2, m_2)\}, \qquad \textit{Course}^{\tilde{I}_0} = \{c_1, c_2, c_3, c_4\}.$$

This structure expresses that course c_1 is certainly selected, while it is unknown whether other modules or courses are selected. Let T_1 be the theory that consists of the sentences (1)–(3) from the introduction. Let $\tilde{J}$ be the structure $\mathscr{O}^{T_1}(\tilde{I}_0)$. We have $t : Selected^{\tilde{J}} = \{m_1, c_1, c_3\}$ and $f : Selected^{\tilde{J}} = \{m_2, c_2\}$. Indeed, because c_1 is selected according to $\tilde{I}_0$, we can derive from (1) that c_2 cannot be selected. Next, (3) implies that module m_2 cannot be selected. It then follows from (2) that m_1 must be selected. This implies in turn that c_3 must be selected. No information about c_4 can be derived since both $\tilde{J}[\textit{Selected}\,(c_4)/\mathbf{t}]$ and $\tilde{J}[\textit{Selected}\,(c_4)/\mathbf{f}]$ are models of T_1.

It is unlikely that tractable methods for computing $\mathscr{O}^T(\tilde{I})$ exist. That can be seen from the fact that the problem of deciding whether a given domain atom is inconsistent in $\mathscr{O}^T(\tilde{I})$ is at least as hard as deciding whether T has a model approximated by $\tilde{I}$. For some FO theories T, the latter problem is **NP**-complete [Fagin 1974; Mitchell and Ternovska 2005]. The naive way of computing $\mathscr{O}^T(\tilde{I})$ for finite T and $\tilde{I}$ by constructing all models $M \geq_p \tilde{I}$ and taking their $\mathrm{glb}_{\leq_p}$ is exponential in the size of $\tilde{I}$. Alternatively, the truth value of each domain atom $P(\overline{d})$ in $\mathscr{O}^T(\tilde{I})$ can be computed by solving two model expansion problems for T in $\tilde{I}[P(\overline{d})/\mathbf{t}]$ and in $\tilde{I}[P(\overline{d})/\mathbf{f}]$: the truth value is $\mathbf{i}$ if none has a solution, $\mathbf{t}$ if the first has a solution and the second has not, etc. For fixed theory T, this method relies on solving a polynomial number of problems in NP and that is NP-hard for some theories T.

Similarly as for theories, we associate to each sentence φ the monotone propagator $\mathscr{O}^{\varphi}$:

$$\mathscr{O}^{\varphi}(\tilde{I}) = \mathrm{glb}_{\leq_p}\left(\{M \mid \tilde{I} \leq_p M \text{ and } M \models \varphi\}\right).$$

Observe that for any sentence φ implied by T, $\mathscr{O}^{T}(\tilde{I})$ is more precise than $\mathscr{O}^{\varphi}(\tilde{I})$, since

$$\{J \mid \tilde{I} \leq_p J \text{ and } J \models T\} \subseteq \{J \mid \tilde{I} \leq_p J \text{ and } J \models \varphi\}$$

As such, we obtain the following proposition.

PROPOSITION 3.13. *If $T \models \varphi$, then $\mathscr{O}^{\varphi}$ is a monotone propagator for T.*

In particular, if $\varphi \in T$, then $\mathscr{O}^{\varphi}$ is a monotone propagator for T.

From Proposition 3.9 and Proposition 3.13 it follows that every stabilizing $\{\mathscr{O}^{\varphi} \mid \varphi \in T\}$-refinement sequence from finite structure $\tilde{I}$ has the same limit. We denote the propagator $\lim_{\{\mathscr{O}^{\varphi} \mid \varphi \in T\}}$ by $\mathscr{L}_T$. We call a $\{\mathscr{O}^{\varphi} \mid \varphi \in T\}$-refinement sequence also a *T-refinement sequence*.

Example 3.14. Let $\tilde{I}_0$ and T_1 be as in Example 3.12. Let $\langle \tilde{I}_i \rangle_{0 \leq i \leq 4}$ be the T_1-refinement sequence from $\tilde{I}_0$ obtained by applying (in this order) the propagators $\mathscr{O}^{(1)}$, $\mathscr{O}^{(3)}$, $\mathscr{O}^{(2)}$ and $\mathscr{O}^{(3)}$. A reasoning similar to the one we made in Example 3.12 shows that *Selected* $^{\tilde{I}_1}(c_2) = \mathbf{f}$, *Selected* $^{\tilde{I}_2}(m_2) = \mathbf{f}$, *Selected* $^{\tilde{I}_3}(m_1) = \mathbf{t}$, and *Selected* $^{\tilde{I}_4}(c_3) = \mathbf{t}$.

Hence, $\tilde{I}_4 = \mathscr{O}^{T_1}(\tilde{I}_0)$, the refinement sequence is stabilizing and $\mathscr{L}_{T_1}(\tilde{I}_0) = \mathscr{O}^{T_1}(\tilde{I}_0)$.

Example 3.15. Let T be the theory $\{P \Leftrightarrow Q, P \Leftrightarrow \neg Q\}$. Then $\mathscr{O}^{T}(\bot^{\leq_p}) = \top^{\leq_p}$ and $\mathscr{L}_T(\bot^{\leq_p}) = \bot^{\leq_p}$.

As Example 3.15 shows, it is not necessarily the case that $\mathscr{L}_T(\tilde{I}) = \mathscr{O}^{T}(\tilde{I})$. In general, only $\mathscr{L}_T(\tilde{I}) \leq_p \mathscr{O}^{T}(\tilde{I})$ holds. Note that $\mathscr{L}_T(\tilde{I}) = \mathscr{O}^{T}(\tilde{I})$ holds if T contains precisely one sentence.

4. FEASIBLE PROPAGATION FOR FIRST-ORDER LOGIC

In this section, we introduce a constraint propagation method that, for finite FO theories T and in finite partial structures, has polynomial-time data complexity and hence, is computationally less expensive than $\mathscr{O}^{T}$ or $\mathscr{L}_T$. The method we propose is based on *implicational normal form propagators* (INF propagators). These propagators have several interesting properties. First, they are monotone, ensuring that stabilizing refinement sequences constructed using only INF propagators have a unique limit. Secondly, for finite theories and partial structures INF propagators have polynomial-time data complexity and therefore stabilizing refinement sequences using only INF propagators can be computed in polynomial time. Thirdly, such a refinement sequence can be represented by a set of positive, i.e., negation-free, rules, which makes it possible to use, e.g., logic programming systems to compute the result of propagation. Finally, INF propagators can be applied on symbolic structures, i.e., independent of a four-valued input structure, in which case its propagations hold in all finite or infinite structures that are represented by the symbolic structure.

Our method is inspired by standard rules for Boolean constraint propagation as studied, e.g., by McAllester [1990] and Apt [1999b]. The intuition is as follows. Consider the parse trees of all formulas of T. With each node, we may associate a set of propagators. For example, with $\varphi = \psi \wedge \phi$, we associate the following propagators: to derive the falsity of φ if ψ or ϕ is false, to derive the truth of φ if both ψ, ϕ are true, to derive that ψ, ϕ are true if φ is true, and finally, to derive that a conjunct (one of ψ and ϕ) is false if φ is false and the other conjunct is true. All these propagators correspond

to INF formulas. These are obtained by introducing for each non-leaf node φ a separate predicate A_φ of arity equal to the number of free variables of φ. In case of the above conjunction (assuming ψ and ϕ are non-leafs), these formulas are:

$$\forall \overline{x}(\neg A_\psi(\overline{x}) \Rightarrow \neg A_\varphi(\overline{x})), \quad \forall \overline{x}(\neg A_\phi(\overline{x}) \Rightarrow \neg A_\varphi(\overline{x}))$$
$$\forall \overline{x}(A_\psi(\overline{x}) \wedge A_\phi(\overline{x}) \Rightarrow A_\varphi(\overline{x}))$$
$$\forall \overline{x}(A_\varphi(\overline{x}) \Rightarrow A_\psi(\overline{x})), \quad \forall \overline{x}(A_\varphi(\overline{x}) \Rightarrow A_\phi(\overline{x}))$$
$$\forall \overline{x}(A_\psi(\overline{x}) \wedge \neg A_\varphi(\overline{x}) \Rightarrow \neg A_\phi(\overline{x})), \quad \forall \overline{x}(A_\phi(\overline{x}) \wedge \neg A_\varphi(\overline{x}) \Rightarrow \neg A_\psi(\overline{x}))$$

Each propagator implements "forward reasoning" on such an implication: it derives the consequent if the condition holds. Each implements a propagation either upward in the parse tree (truth or falsity of φ) or downward (truth or falsity of either ψ or ϕ). The set of all these propagators give us the new propagation method for FO theories.

4.1. Implicational Normal Form Propagators

INF propagators are associated to FO sentences in implicational normal form.

Definition 4.1 (*INF*). An FO sentence is in *implicational normal form (INF)* if it is of the form $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$, where ψ is an arbitrary formula with free variables among $\overline{x}$ and L a literal with free variables $\overline{x}$.

For an INF sentence $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$, the associated INF propagator computes the value of ψ in the given structure. If this value is **t** or **i**, the literal $L[\overline{x}]$ is made true (or inconsistent if it was false or inconsistent in the given structure). This is formalized in the following definition.

Definition 4.2 ($\mathscr{I}^\varphi$). The operator $\mathscr{I}^\varphi$ associated to the sentence $\varphi := \forall \overline{x}\ (\psi \Rightarrow P(\overline{x}))$ is defined by

$$\begin{array}{rcl} P^{\mathscr{I}^\varphi(\tilde{I})}(\overline{d}) & = & \text{if } \tilde{I}[\overline{x}/\overline{d}](\psi) \geq_p \mathbf{t} \text{ then } \mathrm{lub}_{\leq_p}\{\mathbf{t}, P^{\tilde{I}}(\overline{d})\} \text{ else } P^{\tilde{I}}(\overline{d}) \\ Q^{\mathscr{I}^\varphi(\tilde{I})}(\overline{d}) & = & Q^{\tilde{I}}(\overline{d}) \text{ if } Q \neq P \end{array}$$

The operator $\mathscr{I}^\varphi$ associated to the sentence $\varphi := \forall \overline{x}\ (\psi \Rightarrow \neg P(\overline{x}))$ is defined by

$$\begin{array}{rcl} P^{\mathscr{I}^\varphi(\tilde{I})}(\overline{d}) & = & \text{if } \tilde{I}[\overline{x}/\overline{d}](\psi) \geq_p \mathbf{t} \text{ then } \mathrm{lub}_{\leq_p}\{\mathbf{f}, P^{\tilde{I}}(\overline{d})\} \text{ else } P^{\tilde{I}}(\overline{d}) \\ Q^{\mathscr{I}^\varphi(\tilde{I})}(\overline{d}) & = & Q^{\tilde{I}}(\overline{d}) \text{ if } Q \neq P \end{array}$$

Example 4.3. Sentence (3) of the introduction is an INF sentence. Let $\tilde{I}$ be a structure such that $\mathit{Course}^{\tilde{I}}(c_1)$, $\mathit{Module}^{\tilde{I}}(m_1)$, $\mathit{Selected}^{\tilde{I}}(m_1)$, and $\mathit{In}^{\tilde{I}}(c_1, m_1)$ are true. Then according to the definition of $\mathscr{I}^{(3)}$, $\mathit{Selected}^{\mathscr{I}^{(3)}(\tilde{I})}(c_1) \geq_p \mathbf{t}$. That is, if module m_1 is certainly selected and course c_1 certainly belongs to m_1, then the operator $\mathscr{I}^{(3)}$ associated to sentence (3) derives that c_1 is certainly selected. Note that this operator does not perform contrapositive propagation. For instance, if m_2 is a module, c_2 a course, $\mathit{In}^{\tilde{I}}(c_2, m_2) = \mathbf{t}$ and $\mathit{Selected}^{\tilde{I}}(c_2) = \mathbf{f}$, the operator does not derive that m_2 is certainly not selected. However, $\neg \mathit{Selected}(m_2)$ can be derived by the propagator associated with the equivalent INF formula

$$\forall c \forall m(\mathit{Course}(c) \wedge \mathit{Module}(m) \wedge \neg \mathit{Selected}(c) \wedge \mathit{In}(c, m) \Rightarrow \neg \mathit{Selected}(m)).$$

PROPOSITION 4.4. *For every INF sentence φ, $\mathscr{I}^\varphi$ is a monotone propagator.*

PROOF. Since φ is an INF sentence, it is of the form $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$. Let P be the predicate in $L[\overline{x}]$, i.e., $L[\overline{x}]$ is either the positive literal $P(\overline{x})$ or the negative literal $\neg P(\overline{x})$.

It follows directly from the definition of $\mathscr{I}^\varphi$ that $\tilde{I} \leq_p \mathscr{I}^\varphi(\tilde{I})$ for every structure $\tilde{I}$. Now let J be a structure such that $\tilde{I} \leq_p J$ and $J \models \varphi$. To show that $\mathscr{I}^\varphi$ is a propagator, we have to prove that $P^{\mathscr{I}^\varphi(\tilde{I})}(\overline{d}) \leq_p P^J(\overline{d})$ for every tuple $\overline{d}$ of domain elements. If $\tilde{I}[\overline{x}/\overline{d}](\psi) \leq_p \mathbf{f}$ then $P^{\mathscr{I}^\varphi(\tilde{I})}(\overline{d}) = P^{\tilde{I}}(\overline{d}) \leq_p P^J(\overline{d})$. Otherwise, given that $\tilde{I} \leq_p J$, we have that $\tilde{I}$ is consistent, hence, $\tilde{I}[\overline{x}/\overline{d}](\psi) = \mathbf{t}$. It follows that $J[\overline{x}/\overline{d}](\psi) = \mathbf{t}$ and therefore $J[\overline{x}/\overline{d}](L[\overline{x}]) = \mathbf{t}$. It follows that $\tilde{I}[\overline{x}/\overline{d}](L[\overline{x}]) \leq_p \mathbf{t}$ and hence $\mathscr{I}^\varphi(\tilde{I})[\overline{x}/\overline{d}](L[\overline{x}]) = \mathbf{t}$. We conclude that $P^{\mathscr{I}^\varphi(\tilde{I})}(\overline{d}) \leq_p P^J(\overline{d})$.

The monotonicity of $\mathscr{I}^\varphi$ follows from the fact that $\tilde{I} \leq_p \tilde{J}$ implies $\tilde{I}\theta(\psi) \leq_p \tilde{J}\theta(\psi)$ for any two structures $\tilde{I}$ and $\tilde{J}$ and variable assignment θ. □

As mentioned in Section 2.2.2, evaluating a formula in a finite four-valued structure $\tilde{I}$ takes time polynomial in $|\tilde{I}|$, the cardinality of the domain. It follows that for a fixed INF sentence φ and finite structure $\tilde{I}$, computing $\mathscr{I}^\varphi(\tilde{I})$ takes time polynomial in $|\tilde{I}|$. If we combine this result with Proposition 3.8, we obtain the following theorem.

THEOREM 4.5. *For finite sets V of INF sentences and finite $\tilde{I}$, $\lim_{\{\mathscr{I}^\varphi|\varphi\in V\}}(\tilde{I})$ is computable in time polynomial in $|\tilde{I}|$ and exponential in $|V|$.*

PROOF.
Let $\varphi_1, \ldots, \varphi_n$ be all sentences in V, with $n = \#(V)$. Let $\langle \tilde{J}_i \rangle_{0 \leq i \leq m}$ be the longest sequence of structures such that $\tilde{J}_0 = \tilde{I}$ and $\tilde{J}_{i+1} = \mathscr{I}^{\varphi_k}(\tilde{J}_i)$, where k is the lowest number between 1 and n such that $\tilde{J}_i \neq \mathscr{I}^{\varphi_k}(\tilde{J}_i)$. Clearly, $\langle \tilde{J}_i \rangle_{0 \leq i \leq m}$ is a stabilizing $\{\mathscr{I}^\varphi \mid \varphi \in V\}$-refinement sequence from $\tilde{I}$. Proposition 3.8 implies that the length m of this sequence is bound by $2 \cdot |\tilde{I}|^{MaxAr(\Sigma)} \cdot \#(\Sigma)$. To perform one step, we search amongst the $\#(V)$ operators for one that infers at least one domain literal. Applying one operator involves querying the body of the corresponding rule for at most $|I|^{MaxAr(\Sigma)}$ instances. Let $MaxBod(V)$ be the size of the largest body of a rule of V. Each query can be solved in $O(|\tilde{I}|^{MaxBod(V)})$ [Grädel et al. 2007].

Combining all costs, we obtain that an algorithm exists that runs in time:

$$2 \cdot \#(\Sigma) \cdot \#(V) \cdot |\tilde{I}|^{2 \cdot MaxAr(\Sigma)} \cdot O(|\tilde{I}|^{MaxBod(V)})$$

Given that $MaxAr(\Sigma) \leq |V|$ and $MaxBod(V) \leq |V|$, the theorem follows. □

4.2. Representing INF Refinement Sequences by a Positive Rule Set

For the rest of this section, let V be a set of INF sentences over Σ and denote by $\mathscr{I}(V)$ the set $\{\mathscr{I}^\varphi \mid \varphi \in V\}$. We now show how to represent the propagator $\lim_{\mathscr{I}(V)}$ by a rule set Δ consisting of material implications of the form:

$$\forall \overline{x}\ (P(\overline{x}) \Leftarrow \varphi[\overline{y}]),$$

where $P(\overline{x})$ is an atom and φ a positive formula over $\mathrm{tf}(\Sigma)$, and $\overline{y} \subseteq \overline{x}$. In particular, for every structure $\tilde{I}$, $\lim_{\mathscr{I}(V)}(\tilde{I})$ corresponds to a model of Δ satisfying an appropriate minimality condition. The benefit of this is that such models can be computed using a range of technologies developed in many logic- and rule-based formalisms.

A well-known property of rule sets Δ of this form is that for an arbitrary structure I, the set $\{M | M \models \Delta \wedge M \geq_t I\}$ contains a least element that can be computed by iterating the *inflationary consequence operator* of Δ from I [Abiteboul and Vianu 1991].

To each finite set of INF sentences over Σ, we associate the following positive rule set over $\mathrm{tf}(\Sigma)$.

Definition 4.6 (*Rule set*). Let V be a set of INF sentences and $\tilde{I}$ a structure. The *rule set associated to* V is denoted by Δ_V and defined by

$$\Delta_V = \{\forall \overline{x}\ ((L[\overline{x}])_{\mathrm{ct}} \Leftarrow \psi_{\mathrm{ct}}) \mid \forall \overline{x}\ (\psi \Rightarrow L[\overline{x}]) \in V\}\,.$$

Observe that because of Proposition 2.6, bodies in Δ_V are positive. The following proposition explains that Δ_V can be seen as a description of $\lim_{\mathscr{I}(V)}$.

PROPOSITION 4.7. *For every set V of INF sentences over Σ and Σ-structure $\tilde{I}$,* $\mathrm{tf}(\lim_{\mathscr{I}(V)}(\tilde{I})) = Min_{\leq_t}(\{M \mid M \models \Delta_V$ *and* $M \geq_t \mathrm{tf}(\tilde{I})\})$.

PROOF. In the rest of this proof, let $\tilde{J}$ be the structure $\lim_{\mathscr{I}(V)}(\tilde{I})$. Since $\tilde{I} \leq_p \tilde{J}$, it holds that $\mathrm{tf}(\tilde{I}) \leq_t \mathrm{tf}(\tilde{J})$. It now suffices to show that $\mathrm{tf}(\tilde{J}) \models \Delta_V$ and that $\mathrm{tf}(\tilde{J}) \leq_t M$ holds for every model M of Δ_V such that $\mathrm{tf}(\tilde{I}) \leq_t M$.

We first show that $\mathrm{tf}(\tilde{J})$ is a model of Δ_V. Let $\forall \overline{x}(\psi \Rightarrow L[\overline{x}]) \in V$. Because $\tilde{J}$ is the limit of a $\mathscr{I}(V)$-refinement sequence, $\tilde{J}\theta(\psi) \leq_p \tilde{J}\theta(L[\overline{x}])$ for every variable assignment θ. Hence, if $\mathrm{tf}(\tilde{J})\theta(\psi_{\mathrm{ct}}) = \mathbf{t}$, then $\mathrm{tf}(\tilde{J})\theta((L[\overline{x}])_{\mathrm{ct}}) = \mathbf{t}$. It follows that $\mathrm{tf}(\tilde{J}) \models \forall \overline{x}\ ((L[\overline{x}])_{\mathrm{ct}} \leftarrow \psi_{\mathrm{ct}})$. Hence, $\mathrm{tf}(\tilde{J})$ is a model of $\tilde{J}$.

Second, assume that $M \models \Delta_V$ and $M \geq_t \mathrm{tf}(\tilde{I})$. To show that $\mathrm{tf}(\tilde{J}) \leq_t M$, let $\langle \tilde{K}_\xi \rangle_{0 \leq \xi \leq \alpha}$ be a stabilizing $\mathscr{I}(V)$-refinement sequence from $\tilde{I}$. Then $\tilde{K}_\alpha = \tilde{J}$. We prove by induction that for each $0 \leq \xi \leq \alpha$, $\mathrm{tf}(\tilde{K}_\xi) \leq_t M$. The induction hypothesis is trivially satisfied for $\xi = 0$. If it holds for all ξ less than limit ordinal λ, it is clearly satisfied for λ as well. So, suppose it holds for ordinal ξ. Assume that $\tilde{K}_{\xi+1} = \mathscr{I}^{\varphi}(\tilde{K}_\xi)$, for some $\varphi = \forall \overline{x}\ (\psi \Rightarrow P(\overline{x})) \in V$. The partial structure $\tilde{K}_{\xi+1}$ is identical to $\tilde{K}_\xi$ except that if for some θ, $\tilde{K}_\xi\theta(\psi) \geq_p \mathbf{t}$, then $\tilde{K}_{\xi+1}\theta(P(\overline{x})) = \mathrm{lub}_{\leq_p}\{\mathbf{t}, P^{\tilde{K}_\xi}(\theta(\overline{x}))\}$. In such a case, the body ψ_{ct} of the rule in Δ_V for φ holds in $\mathrm{tf}(\tilde{K}_\xi)\theta$. Since $\mathrm{tf}(\tilde{K}_\xi) \leq_t M$ and ψ_{ct} is a positive formula, we have $M\theta \models \psi_{\mathrm{ct}}$. Given that $M \models \Delta_V$, it follows that $M\theta \models P_{\mathrm{ct}}(\overline{x})$. Consequently, we have $\mathrm{tf}(K)_{\xi+1} \leq_t M$. The case for formulas of the form $\varphi = \forall \overline{x}\ (\psi \Rightarrow \neg P(\overline{x}))$ is analogous. □

The least model M of Δ_V for which $I \leq_t M$ holds, corresponds to the least Herbrand model of the positive rule set derived from Δ_V by introducing a fresh constant symbol C_d for every domain element d in $\tilde{I}$ and adding to Δ_V the atomic formulas $P_{\mathrm{ct}}(C_{d_1}, \ldots, C_{d_n})$, respectively $P_{\mathrm{cf}}(C_{d_1}, \ldots, C_{d_n})$, for every domain atom $P(d_1, \ldots, d_n)$ that is true, respectively false, in $\tilde{I}$.

There are several benefits of using Δ_V as a description of $\lim_{\mathscr{I}(V)}$. From a practical point of view, Proposition 4.7 states that we can use any existing algorithm that computes the least Herbrand model of positive rule sets to implement $\lim_{\mathscr{I}(V)}$. Several such algorithms have been developed. For example, in the area of production rule systems, RETE [Forgy 1982] and LEAPS [Miranker et al. 1990] are two well-known algorithms. Other examples are the algorithms implemented in Prolog systems with tabling such as XSB [Swift 2009] and YAP [Faustino da Silva and Santos Costa 2006]. In the context of databases, a frequently used algorithm is semi-naive evaluation [Ullman 1988]. Rule sets are also supported by the model generator IDP for the logic FO(ID) [Wittocx et al. 2008c] and by ASP systems such as clasp [Gebser et al. 2007] and DLV [Perri et al. 2007]. A recent comparision of various algorithms for evaluating rule sets is in [Van Weert 2010]. It follows that the large amount of research on optimization techniques and execution strategies for these algorithms can be used to obtain efficient implementations of $\lim_{\mathscr{I}(V)}$ for a set V of INF propagators.

Most of the algorithms and systems mentioned above expect that all rules are of the form $\forall \overline{x}\ (P(\overline{x}) \leftarrow \exists \overline{y}\ (Q_1(\overline{z}_1) \wedge \ldots \wedge Q_n(\overline{z}_n)))$, i.e., each body is the existential quantifi-

cation of a conjunction of atoms. Some of the algorithms, e.g., semi-naive evaluation, can easily be extended to more general rule sets. Instead of extending the algorithms, one can as well rewrite rule sets into the desired format by applying predicate introduction [Vennekens et al. 2007], provided that only structures with finite domains are considered.

Other potential benefits of representing $\lim_{\mathscr{I}(V)}$ by Δ_V stem from the area of logic program analysis. For instance, *abstract interpretation* of logic programs [Bruynooghe 1991] could be used to derive interesting properties of $\lim_{\mathscr{I}(V)}$, *program specialization* [Leuschel 1997] to tailor Δ_V to a specific class of structures $\tilde{I}$, *folding* [Pettorossi and Proietti 1998] to combine the application of several propagators, etc.

4.3. From FO to INF

Having introduced INF propagators, we now describe a conversion from FO theories to INF sentences which is the basis of our propagation method. The method consists of transforming a theory T into an equivalent set of INF sentences. It does this in linear time in the total size of the parse trees of the sentences in T and hence, in linear time in $|T|$, the number of symbol occurrences in T. Then propagation on T can be performed by applying the INF propagators on the obtained INF sentences. We will show that this method has polynomial-time data complexity. The price for this improved efficiency is loss in precision.

The next subsection describes the transformation. Note that the algorithm is non-optimal and that often a much more compact set of INF sentences can be generated (our implementation does). However, as we have no claims of optimality and our sole aim is to state polynomial-time data complexity, we present the most straightforward transformation.

4.3.1. From FO to Equivalence Normal Form. The transformation of finite theories to INF sentences works in two steps. First, a theory T is transformed into a Σ-equivalent set of sentences in *equivalence normal form* (ENF). Next, each ENF sentence is replaced by a set of INF sentences. We show that both steps can be executed in linear time. Also, we prove a theorem stating that under mild conditions, no precision is lost in the second step. That is, for each ENF sentence φ that satisfies these conditions, there exists a set of INF sentences V such that $\mathscr{O}^{\varphi} = \lim_{\mathscr{I}(V)}$.

Definition 4.8 (***ENF***). An FO sentence φ is in *equivalence normal form (ENF)* if it is of the form $\forall \overline{x}\ (L[\overline{x}] \Leftrightarrow \psi[\overline{x}])$, where ψ is of the form $(L_1 \wedge \ldots \wedge L_n)$, $(L_1 \vee \ldots \vee L_n)$, $(\forall \overline{y}\ L')$, or $(\exists \overline{y}\ L')$, and L, L', L_1, $\ldots$, L_n are literals. An FO theory is in ENF if all its sentences are.

Recall that we denote by $\psi[\overline{x}]$ that $\overline{x}$ are precisely the free variables of ψ. Thus, the definition of ENF implicitly states that in every ENF sentence $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$, the free variables of ψ are the free variables of $P(\overline{x})$.

We now show that every FO theory T over a vocabulary Σ can be transformed into a Σ-equivalent ENF theory T'. The transformation is akin to the Tseitin transformation for propositional logic [Tseitin 1968].

Definition 4.9 (***Transformation to ENF***).

Let $NNF(T)$ be the negation normal form of T that is obtained by eliminating implication, pushing negation inside (eliminating double negation) and bringing nested conjunctions and disjunctions in right-associative form, e.g., $((P \wedge Q) \wedge R)$ is replaced by $P \wedge (Q \wedge R)$.

Let T' consist of all formulas $\varphi \in T$ of the form $\forall \overline{x}\ (L[\overline{x}] \Leftrightarrow \psi[\overline{x}])$ with L a literal, and of formulas $\mathit{true} \Leftrightarrow \varphi$ for all remaining formulas $\varphi \in T$.

$ENF(T)$ is obtained by iteratively rewriting T' using the following rewrite rule: for some non-literal subformula $\psi[\overline{x}]$ that occurs inside a conjunction, disjunction or quantified formula φ on the righthandside of an equivalence, introduce a new predicate A_ψ of the same arity as $\psi[\overline{x}]$, substitute $A_\psi(\overline{x})$ for $\psi[\overline{x}]$ in φ, and add the equivalence

$$\forall \overline{x}(A_\psi(\overline{x}) \Leftrightarrow \psi[\overline{x}])$$

This rule is applied till all formulas are in ENF, i.e., all formulas are equivalences with conjunctions, disjunctions, or quantified formulas of literals at the righthand side.

Note that the auxiliary predicate symbols correspond to nodes in the parse tree of formulas, as explained in the introduction of this section.

Clearly, $ENF(T)$ is a well-defined theory in ENF. If T is finite, $ENF(T)$ can be constructed in time linear in $|T|$ and its size is linear in $|T|$. The arity of the auxiliary predicates is bound by $width(T)$, the maximal number of free variables in a subformula of T.

Example 4.10. The result of applying the ENF transformation on the theory T_1 from Example 3.12 is the theory

$$\begin{aligned}
&true \Leftrightarrow \forall x \forall y\ A_1(x,y),\\
&\forall x \forall y\ (A_1(x,y) \Leftrightarrow \neg \textbf{\textit{MutExcl}}\,(x,y) \vee \neg \textbf{\textit{Selected}}\,(x) \vee \neg \textbf{\textit{Selected}}\,(y)),\\
&true \Leftrightarrow \exists m\ A_2(m),\\
&\forall m\ (A_2(m) \Leftrightarrow \textbf{\textit{Module}}\,(m) \wedge \textbf{\textit{Selected}}\,(m)),\\
&true \Leftrightarrow \forall c\ A_3(c),\\
&\forall c\ (A_3(c) \Leftrightarrow \neg \textbf{\textit{Course}}\,(c) \vee A_4(c) \vee \textbf{\textit{Selected}}\,(c)),\\
&\forall c\ (A_4(c) \Leftrightarrow \forall m\ A_5(m,c)),\\
&\forall c \forall m\ (A_5(m,c) \Leftrightarrow \neg \textbf{\textit{Module}}\,(m) \vee \neg \textbf{\textit{Selected}}\,(m) \vee \neg \textbf{\textit{In}}\,(c,m)).
\end{aligned}$$

As the transsformations to NNF and of formulas φ to $true \Leftrightarrow \varphi$ in Definition 4.9 trivially preserve logical equivalence and the rewrite rule preserves Σ-equivalence according to Proposition 2.2, the following proposition holds:

PROPOSITION 4.11. *For each theory T over Σ, T and $ENF(T)$ are Σ-equivalent.*

The combination of Proposition 3.5 and Proposition 4.11 ensures propagators for T' can be used to implement propagators for T.

4.3.2. From ENF to INF. As shown in the previous section, every theory over Σ can be transformed into a Σ-equivalent ENF theory. Now we show that any ENF theory T can be transformed into a logically equivalent theory INF(T) containing only INF sentences. As a result, we obtain a propagator for T with polynomial-time data complexity.

Definition 4.12 (*From ENF to INF*). For an ENF sentence φ, the set of INF sentences INF(φ) is defined in Table I. For an ENF theory T, INF(T) denotes the set of INF sentences $\bigcup_{\varphi \in T}$ INF(φ).

The set of all INF sentences associated to an ENF sentence φ contains for each predicate P that occurs in φ a sentence of the form $\forall \overline{x}\ (\psi \Rightarrow P(\overline{x}))$ and a sentence of the form $\forall \overline{x}\ (\psi \Rightarrow \neg P(\overline{x}))$. As such, the corresponding propagators are, in principle, able to derive that a domain atom $P(\overline{d})$ is true, respectively false, if this is implied by φ.

Note that, from a logical perspective, the transformation from ENF to INF creates a lot of redundancy. Each INF sentence derived from $\forall \overline{x}\ (L[\overline{x}] \Leftrightarrow \psi)$ is either logically

Table I. From ENF to INF

φ	INF(φ)	
$\forall \overline{x}\ (L \Leftrightarrow L_1 \wedge \ldots \wedge L_n)$	$\forall \overline{x}\ (L_1 \wedge \ldots \wedge L_n \Rightarrow L)$	
	$\forall \overline{x}\ (\neg L_i \Rightarrow \neg L)$	$1 \leq i \leq n$
	$\forall \overline{x}\ (L \Rightarrow L_i)$	$1 \leq i \leq n$
	$\forall \overline{x}\ (\neg L \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_n \Rightarrow \neg L_i)$	$1 \leq i \leq n$
$\forall \overline{x}\ (L \Leftrightarrow L_1 \vee \ldots \vee L_n)$	$\forall \overline{x}\ (\neg L_1 \wedge \ldots \wedge \neg L_n \Rightarrow \neg L)$	
	$\forall \overline{x}\ (L_i \Rightarrow L)$	$1 \leq i \leq n$
	$\forall \overline{x}\ (\neg L \Rightarrow \neg L_i)$	$1 \leq i \leq n$
	$\forall \overline{x}\ (L \wedge \neg L_1 \wedge \ldots \wedge \neg L_{i-1} \wedge \neg L_{i+1} \wedge \ldots \wedge \neg L_n \Rightarrow L_i)$	$1 \leq i \leq n$
$\forall \overline{x}\ (L[\overline{x}] \Leftrightarrow \forall \overline{y}\ L'[\overline{x}, \overline{y}])$	$\forall \overline{x}\ ((\forall \overline{y}\ L'[\overline{x}, \overline{y}]) \Rightarrow L[\overline{x}])$	
	$\forall \overline{x}(\exists \overline{y}\ \neg L'[\overline{x}, \overline{y}]) \Rightarrow \neg L[\overline{x}])$	
	$\forall \overline{x} \forall \overline{y}\ (L[\overline{x}] \Rightarrow L'[\overline{x}, \overline{y}])$	
	$\forall \overline{x} \forall \overline{y}\ ((\neg L[\overline{x}] \wedge \forall \overline{z}\ (\overline{y} \neq \overline{z} \Rightarrow L'[\overline{x}, \overline{y}][\overline{y}/\overline{z}])) \Rightarrow \neg L'[\overline{x}, \overline{y}])$	
$\forall \overline{x}\ (L[\overline{x}] \Leftrightarrow \exists \overline{y}\ L'[\overline{x}, \overline{y}])$	$\forall \overline{x}\ ((\forall \overline{y}\ \neg L'[\overline{x}, \overline{y}]) \Rightarrow \neg L[\overline{x}])$	
	$\forall \overline{x}(\exists \overline{y}\ L'[\overline{x}, \overline{y}]) \Rightarrow L[\overline{x}])$	
	$\forall \overline{x} \forall \overline{y}\ (\neg L[\overline{x}] \Rightarrow \neg L'[\overline{x}, \overline{y}])$	
	$\forall \overline{x} \forall \overline{y}\ ((L[\overline{x}] \wedge \forall \overline{z}\ (\overline{y} \neq \overline{z} \Rightarrow \neg L'[\overline{x}, \overline{y}][\overline{y}/\overline{z}])) \Rightarrow L'[\overline{x}, \overline{y}])$	

equivalent to $\forall \overline{x}\ (L[\overline{x}] \Rightarrow \psi)$ or to $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$. Yet, dropping a logically redundant sentence deletes a propagator and may decrease precision.

It is straightforward to verify the following proposition.

PROPOSITION 4.13. *For every ENF sentence φ,* INF(φ) *is logically equivalent to φ. For every ENF theory T,* INF(T) *is logically equivalent to T.*

It follows that if T is an ENF theory, any propagator for INF(T) is a propagator for T. In particular, for every $\varphi \in$ INF(T), $\mathscr{I}^{\varphi}$ is a polynomial-time propagator for T. As a corollary of Theorem 4.5, we have:

PROPOSITION 4.14. *If T is an ENF theory, then the operator $\lim_{\mathscr{I}(\text{INF}(T))}$ is a propagator for T. For finite ENF theories T and finite structures $\tilde{I}$, $\lim_{\mathscr{I}(\text{INF}(T))}(\tilde{I})$ can be computed in time polynomial in $|\tilde{I}|$ and exponential in $|T|$.*

PROOF. Let V be $INF(T)$. Consider the bound on computing a stabilizing refinement sequence for V from the proof of Theorem 4.5.

$$2 \cdot \#(\Sigma) \cdot \#(V) \cdot |\tilde{I}|^{2 \cdot MaxAr(\Sigma)} \cdot O(|\tilde{I}|^{MaxBod(V)})$$

The maximal size $MaxBod(V)$ of the bodies of V is linear in the maximal size of a righthand formula in the equivalences of T which we denote similarly as $MaxBod(T)$. The transformation from ENF to INF preserves Σ and translates one sentence of T in a set of n sentences with n at most linear in $MaxBod(T)$. Hence, $\#(V) \leq \#(T) \cdot O(MaxBod(T))$. Thus, we can compute the sequence in time:

$$2 \cdot \#(\Sigma) \cdot \#(T) \cdot O(MaxBod(T)) \cdot |\tilde{I}|^{2 \cdot MaxAr(\Sigma)} \cdot O(|\tilde{I}|^{O(MaxBod(T))})$$

□

4.4. Summary

Combining Proposition 3.5, Proposition 4.11 and Proposition 4.13 yields the following result.

THEOREM 4.15. *Let T be a theory over vocabulary Σ and $\tilde{I}$ be a four-valued Σ-structure. For any $\mathscr{I}$(*INF*($ENF(T)$))-refinement sequence from $\tilde{I}$ with limit $\tilde{J}$, $(\text{INCO}(\tilde{J}))|_{\Sigma}$ is a propagation of T from $\tilde{I}$.*

In the context of finite theories and structures, the theorem suggests the following non-deterministic propagation algorithm.

Algorithm 4.16. Input: a finite theory T over Σ and a finite Σ-structure $\tilde{I}$.

(1) Construct $T' = \mathscr{I}(\text{INF}(ENF(T)))$ using Definition 4.9 and Definition 4.12.
(2) Construct a T'-refinement sequence from $\tilde{I}$.[2] Denote the last element by $\tilde{J}$.
(3) Return $(\text{INCO}(\tilde{J}))|_\Sigma$.

The refinement sequence in the algorithm is arbitrary and does not need to be stabilizing. Hence, this is a non-deterministic any-time algorithm for computing propagation of FO theories in structures. Obviously, a good implementation should stop the refinement sequence and apply INCO as soon as an inconsistency is discovered.

By Proposition 3.9, the refined algorithm that computes stabilizing refinement sequences is deterministic and defines a propagator. From Proposition 4.14, it follows that this algorithm has polynomial-time data complexity:

PROPOSITION 4.17. *For finite theories T over Σ and finite structures $\tilde{I}$, the refined Algorithm 4.16 that computes stabilizing refinements can be executed in time polynomial in $|\tilde{I}|$ and exponential in $|T|$.*

PROOF. Let T' be theory $ENF(T)$. By Proposition 4.14, we have an algorithm that computes stabilizing refinement sequences for T' in $\tilde{I}$ in time:

$$2 \cdot \#(\Sigma') \cdot \#(T') \cdot O(MaxBod(T')) \cdot |\tilde{I}|^{2 \cdot MaxAr(\Sigma')} \cdot O(|\tilde{I}|^{O(MaxBod(T'))})$$

where Σ' is Σ augmented with all auxiliary symbols A_φ. The transformation from T to T' takes time $O(|T|)$ and can be ignored. We have $\#(\Sigma') \leq \#(\Sigma) \cdot |T|$, $\#(T') \leq |T|$, $MaxAr(\Sigma') = width(T)$ and $MaxBody(T') \leq MaxFSize(T)$. Combining all these cost factors we obtain that the following bound for the algorithm.

$$2 \cdot \#(\Sigma) \cdot |T|^2 \cdot O(MaxFSize(T)) \cdot |\tilde{I}|^{2 \cdot width(T)} \cdot O(|\tilde{I}|^{O(MaxFSize(T))})$$

□

The exponential complexity of the algorithm is a worst case complexity. If we assume that the size of formulas in T is bounded – a reasonable assumption for theories developed by human experts – then the algorithm becomes polynomial in the size of T. The complexity could be further refined by improving the factor $O(|\tilde{I}|^{O(MaxFSize(T))})$ which is the cost of evaluating bodies of INF formulas. These are conjunctive or disjunctive queries or universal and existential projections for which better complexity results are available.

Since only INF propagators are used, the second step of Algorithm 4.16 can be executed by representing $\lim_{\mathscr{I}(V)}$ as a positive rule set and computing the least model of that set. In the following, we call Algorithm 4.16 the *propagation algorithm*. In what follows, we use $INF(T)$ as a shorthand for $INF(ENF(T))$.

Algorithm 4.16 can be seen as an algorithm that propagates information up and down the parse tree of the input theory T. Indeed, let A be a predicate and $\forall \overline{x}\ (A(\overline{x}) \Leftrightarrow \varphi)$ a sentence, introduced while transforming T to ENF. As mentioned, A represents the subformula φ of T. Hence, INF sentences in INF$(\forall \overline{x}\ (A(\overline{x}) \Leftrightarrow \varphi))$ of the form $\forall \overline{x}\ (\psi \Rightarrow A(\overline{x}))$ or $\forall \overline{x}\ (\psi \Rightarrow \neg A(\overline{x}))$ propagate information derived about subformulas of φ to φ itself. That is, they propagate information upwards in the parse tree of T. The other INF sentences in INF$(\forall \overline{x}\ (A(\overline{x}) \Leftrightarrow \varphi))$ propagate information about φ downwards.

As an illustration, we apply the propagation algorithm on the theory and structure from Example 3.12.

[2]Recall that Σ-structure $\tilde{I}$ can be seen as a structure of the extended vocabulary.

Example 4.18. Let T_1 and $\tilde{I}_0$ be the theory and structure from Example 3.12. Transforming T_1 to ENF produces the theory shown in Example 4.10. According to Definition 4.12, the set of INF sentences associated with this theory contains, amongst others, the sentences

$$\forall x \forall y\ (true \Rightarrow A_1(x,y)), \tag{4}$$
$$\forall x \forall y\ (A_1(x,y) \wedge \textbf{\textit{MutExcl}}\,(x,y) \wedge \textbf{\textit{Selected}}\,(x) \Rightarrow \neg \textbf{\textit{Selected}}\,(y)), \tag{5}$$
$$\forall c\ (true \Rightarrow A_3(c)), \tag{6}$$
$$\forall c\ (A_3(c) \wedge \textbf{\textit{Course}}\,(c) \wedge \neg \textbf{\textit{Selected}}\,(c) \Rightarrow A_4(c)), \tag{7}$$
$$\forall c \forall m\ (A_4(c) \Rightarrow A_5(m,c)), \tag{8}$$
$$\forall m\ (\textbf{\textit{Module}}\,(m) \wedge \exists c\ (A_5(m,c) \wedge \textbf{\textit{In}}\,(c,m)) \Rightarrow \neg \textbf{\textit{Selected}}\,(m)), \tag{9}$$
$$\forall m\ (\neg \textbf{\textit{Selected}}\,(m) \Rightarrow \neg A_2(m)), \tag{10}$$
$$\forall m\ (\neg \textbf{\textit{Module}}\,(m) \Rightarrow \neg A_2(m)), \tag{11}$$
$$\forall m\ (true \wedge \forall m'\ (m \neq m' \Rightarrow \neg A_2(m')) \Rightarrow A_2(m)), \tag{12}$$
$$\forall m\ (A_2(m) \Rightarrow \textbf{\textit{Selected}}\,(m)), \tag{13}$$
$$\forall m \forall c\ (\textbf{\textit{Module}}\,(m) \wedge \textbf{\textit{Selected}}\,(m) \wedge \textbf{\textit{In}}\,(c,m) \Rightarrow \neg A_5(m,c)), \tag{14}$$
$$\forall c\ (\exists m\ \neg A_5(m,c) \Rightarrow \neg A_4(c)), \tag{15}$$
$$\forall c\ (\textbf{\textit{Course}}\,(c) \wedge A_3(c) \wedge \neg A_4(c) \Rightarrow \textbf{\textit{Selected}}\,(c)). \tag{16}$$

If one applies the associated INF propagators on $\tilde{I}_0$ in the order of the sentences above, the following information is derived. First, propagator $\mathscr{I}^{(5)} \circ \mathscr{I}^{(4)}$ derives that $A_1(c_1,c_2)$ is certainly true and that c_2 is certainly not selected. Next, $\mathscr{I}^{(6)}$ derives that $A_3(c)$ is certainly true for all courses c. The propagator $\mathscr{I}^{(7)}$ combines the derived information and concludes that $A_4(c_2)$ is certainly true. This in turn implies, by $\mathscr{I}^{(8)}$, that $A_5(m,c)$ is certainly true for $m = m_2$ and $c = c_2$. The propagator $\mathscr{I}^{(9)}$ derives from the fact that c_2 belongs to m_2, that m_2 cannot be selected. Next, it is derived that m_1 is certainly selected by applying $\mathscr{I}^{(13)} \circ \cdots \circ \mathscr{I}^{(10)}$, and finally, applying $\mathscr{I}^{(16)} \circ \mathscr{I}^{(15)} \circ \mathscr{I}^{(14)}$ yields that c_3 is certainly selected. As such, exactly the same information as in $\mathscr{O}^{T_1}(\tilde{I}_0)$ is derived.

The following example gives another illustration of what the propagation algorithm can achieve.

Example 4.19. Assume that in a planning context, a precedence relationship $Prec(a_1,a_2)$ is given between actions which states that if action a_2 is performed ($Do(a_2,t)$) then a_1 needs to be performed before. This is expressed by the following theory T_2:

$$\forall a_1 \forall a_2 \forall t_2\ (\textbf{\textit{Action}}\,(a_1) \wedge \textbf{\textit{Action}}\,(a_2) \wedge \textbf{\textit{Time}}\,(t_1) \wedge \textbf{\textit{Prec}}\,(a_1,a_2) \wedge \textbf{\textit{Do}}\,(a_2,t_2) \\ \Rightarrow \exists t_1\ (\textbf{\textit{Time}}\,(t_1) \wedge t_1 < t_2 \wedge \textbf{\textit{Do}}\,(a_1,t_1))).$$

Let $\tilde{I}_2$ be a structure such that

$$\tilde{I}_2(Prec(d_0,d_1) \wedge \ldots \wedge Prec(d_{n-1},d_n)) = \mathbf{t}.$$

$\tilde{I}_2$ indicates that there is a chain of n actions that need to be performed before d_n can be performed. The propagation algorithm can derive for input T_2 and $\tilde{I}_2$ that d_n can certainly not be performed before the $(n+1)$th time point. More in general, actions have fluent preconditions and fluents are caused by actions. When some fluents are caused by only one action, propagation may trace back a chain of necessary actions to obtain a goal fluent. The propagation algorithm can derive this.

As mentioned before, many INF sentences are logically equivalent to each other, but the associated propagators are not. As a consequence, by dropping equivalent INF sentences, propagation may be lost. Still, the resulting theory often can be greatly simplified and many auxiliary predicates eliminated without losing propagation. In particular, if follows from Proposition 4.7 that any transformation of INF theories V that preserves $\mathrm{tf}(\Sigma)$-equivalence of the associated rule set Δ_V of Definition 4.6 preserves all propagation. Techniques inspired by logic programming can be used. For instance, in Example 4.18, we can apply *unfolding* for predicate A_1, replacing sentence (5) by the shorter sentence $\forall x \forall y\ (\boldsymbol{MutExcl}\,(x,y) \wedge \boldsymbol{Selected}\,(x) \Rightarrow \neg \boldsymbol{Selected}\,(y))$, and (4) can be omitted. Similarly, unfolding can be applied to omit (6)–(8) and replace (9) by $\forall m\ (\boldsymbol{Module}\,(m) \wedge \exists c\ (\neg \boldsymbol{Selected}\,(c) \wedge \boldsymbol{In}\,(c,m)) \Rightarrow \neg \boldsymbol{Selected}\,(m))$. Sentences in $\mathrm{INF}(T_1)$ of the form $\forall x \forall y\ (\varphi \Rightarrow A_1(x,y))$ can be omitted because they are *subsumed* by (4). Sentences of the form $\varphi \Rightarrow \mathit{true}$ can be omitted because they are tautologies, and so on. It depends on the practical implementation of the propagation algorithm whether optimizing the set of INF sentences in this manner leads to a significant speed-up.

For sentences φ of some specific form, it is easy to directly associate sets of INF sentences that are smaller than $\mathrm{INF}(\varphi)$ but produce the same propagation. For instance, clauses $\forall \overline{x}\ (L_1 \vee \ldots \vee L_n)$ can be translated as in the Prolog-based theorem prover of [Stickel 1986] to obtain the set $\{\forall \overline{x}\ (\neg L_1 \wedge \ldots \wedge \neg L_{i-1} \wedge \neg L_{i+1} \wedge \ldots \wedge \neg L_n \Rightarrow L_i) \mid 1 \leq i \leq n\}$. For sentence (1) of Example 3.12, this is the set

$$\forall x \forall y\ (\boldsymbol{MutExcl}\,(x,y) \wedge \boldsymbol{Selected}\,(x) \Rightarrow \neg \boldsymbol{Selected}\,(y)), \tag{17}$$

$$\forall x \forall y\ (\boldsymbol{MutExcl}\,(x,y) \wedge \boldsymbol{Selected}\,(y) \Rightarrow \neg \boldsymbol{Selected}\,(x)), \tag{18}$$

$$\forall x \forall y\ (\boldsymbol{Selected}\,(x) \wedge \boldsymbol{Selected}\,(y) \Rightarrow \neg \boldsymbol{MutExcl}\,(x,y)), \tag{19}$$

instead of the fourteen sentences in $\mathrm{INF}((1))$. By extensively applying simplification techniques as described in the previous paragraph, the set $\mathrm{INF}((1))$ reduces to the three sentences (17)–(19).

4.5. Notes on Precision

By Proposition 3.11, the result $\tilde{J}$ of applying Algorithm 4.16 on input theory T and structure $\tilde{I}$ cannot be more precise than $\mathscr{O}^T(\tilde{I})$. As we will show in Example 4.21, there are cases where $\tilde{J}$ is strictly less precise than $\mathscr{O}^T(\tilde{I})$. For applications like, e.g., configuration systems and approximate query answering (see Section 7), it is an important question for which T and $\tilde{I}$ precision is lost.

The loss of precision in Algorithm 4.16 on an input theory T compared to $\mathscr{O}^T$, is in principle due to three factors:

(1) Instead of propagating the theory T as a whole, Algorithm 4.16 considers propagators for individual sentences, and combines them in a refinement sequence. As Example 3.15 shows, this may lead to a loss in precision.
(2) The theory is translated to ENF.
(3) Instead of applying the complete propagator $\mathscr{O}^\varphi$ for an ENF sentence φ, the incomplete propagators $\mathscr{I}^\psi$ for INF sentences $\psi \in \mathrm{INF}(\varphi)$ are applied.

The following theorem states that under some easy-to-verify conditions, the third factor does not contribute to the loss in precision.

THEOREM 4.20. *If φ is an ENF sentence such that no predicate occurs more than once in it, $\mathscr{O}^\varphi = \mathrm{INCO} \circ \lim_{\mathscr{I}(\mathrm{INF}(\varphi))}$.*

The inconsistency propagator INCO is applied here to cope with the fact that the only strictly four-valued structure computed by the complete propagator is the most precise

inconsistent structure $\top^{\leq_p}$ while in such case the propagator $\lim_{\mathscr{I}(\text{INF}(\varphi))}$ computes a non-maximal but strictly four-valued structure. The proof of this theorem is technical and relies on concepts and techniques that stand ortogonal to the topics of this paper. We refer to [Wittocx 2010], Theorem 4.12 for details.

Concerning the loss in precision due to the first two factors mentioned above, it is worth noting that predicate introduction may actually lead to more precise propagation. We illustrate this on an example.

Example 4.21. Consider the propositional theory T consisting of the two sentences $(P \vee Q)$ and $((P \vee Q) \Rightarrow R)$. Clearly, R is true in every model of T and therefore $\mathscr{O}^T(\bot^{\leq_p})(R) = \mathbf{t}$. However, $\mathscr{L}_T(\bot^{\leq_p}) = \bot^{\leq_p}$. Intuitively, this loss in precision is due to the fact that a three-valued structure cannot "store" the information that $(P \vee Q)$ is true in every model of T if neither P nor Q is true in every model. However, if we apply predicate introduction to translate T to the theory T' consisting of the sentences

$$A \Leftrightarrow P \vee Q, \qquad A, \qquad A \Rightarrow R,$$

there is no loss in precision: $\mathscr{L}_{T'}(\bot^{\leq_p})(R) = \mathbf{t}$. The fact that $(P \vee Q)$ must be true is "stored" in the interpretation of the introduced predicate A.

It remains a topic for future research to investigate the precision of our algorithm in a more general context. Such a study may be inspired by precision results obtained for the approximate query answering method in incomplete databases presented in Denecker et al. [2010] which in many aspects is a special case of our algorithm (see Section 7.5).

5. SYMBOLIC PROPAGATION

In the previous section, we constructed propagators that operate on concrete four-valued structures and map them to a refinement with respect to the given theory. In this section, we *lift* this operator to a symbolic propagator that operates on *symbolic four-valued structures*.

A symbolic structure Φ of Σ is a special case of an *interpretation between theories* [Enderton 2001]. It assigns to each predicate symbol P/n a formula, or more precisely an n-ary query over some auxiliary vocabulary Υ. This way, a symbolic structure defines the meaning of symbols of Σ in terms of those of Υ. This is reminiscent of deductive databases, with symbolic structures corresponding to FO definitions of intensional predicates in Σ in terms of extensional predicates in Υ. As such, a symbolic interpretation induces a mapping from Υ-structures to Σ-structures ($\sim$ view materialization).

Symbolic four-valued structures $\tilde{\Phi}$ of Σ in terms of Υ are just symbolic structures of $\text{tf}(\Sigma)$ in terms of Υ. Concrete two-valued $\text{tf}(\Sigma)$-structures correspond one to one to four-valued Σ-structures. Thus, a four-valued structure $\tilde{\Phi}$ induces a mapping from Υ-structures to four-valued structures of Σ. In this paper, Υ can be an arbitrary vocabulary. In some applications, Υ will be a set of symbols with a known interpretation. In other applications, it will be $\text{tf}(\Sigma)$ itself, and $\tilde{\Phi}$ will be a symbolic representation of a refinement of a given four-valued Σ-structure.

We will define a class of propagators on such symbolic four-valued structures that *lift* the INF propagators on concrete four-valued structures. The lifting property is that if Φ' is obtained by symbolic INF propagation from a four-valued symbolic structure Φ, then for arbitrary Υ-structure E, if $\tilde{I}, \tilde{J}$ are induced by E using respectively the symbolic structures Φ and Φ', then $\tilde{J}$ is the result of an INF propagator applied on $\tilde{I}$.

As will become clear, symbolic propagation is beneficial when precision is less important than efficiency, when only parts of the result of propagation are of interest, or

when propagation for a fixed theory needs to be performed for arbitrary and/or potentially infinite structures.

5.1. Symbolic Structures

Definition 5.1 (*Symbolic structure* Φ). A *symbolic two-valued* Σ*-structure* Φ over Υ consists of a query P^Φ for each predicate $P \in \Sigma$. The query P^Φ for a predicate P/n is of the form $\{(x_1, \ldots, x_n) \mid \varphi\}$ with φ a formula over Υ.

For the rest of this section, when we use the term *symbolic structure*, we mean a symbolic Σ-structure over Υ.

A symbolic two-valued structure Φ induces an obvious mapping from two-valued Υ-structures E with domain D to two-valued Σ-structures, denoted $\Phi(E)$, over the same domain D and interpreting predicates $P \in \Sigma$ by the answer to the corresponding query P^Φ in E.

Definition 5.2 ($\Phi(E)$). Let E be a Υ-structure. Then $\Phi(E)$ denotes a Σ-structure which, for each predicate in $P \in \Sigma$, is defined as $(P)^{\Phi(E)} = (P^\Phi)^E$.

Example 5.3. Let $\Sigma = \{\textit{Rhombus}/1\}$ and $\Upsilon = \{\textit{Quadrilateral}/1, \textit{EqualSides}/1\}$. An example of a symbolic Σ-structure over Υ is the symbolic structure Φ that assigns $\textit{Rhombus}^\Phi = \{x \mid \textit{Quadrilateral}(x) \wedge \textit{EqualSides}(x)\}$. If E is the Υ-structure with domain $D = \{a, b, c\}$ that assigns $\textit{Quadrilateral}^E = \{a, b\}$ and $\textit{EqualSides}^E = \{b, c\}$, then $\Phi(E)$ is the Σ-structure with domain D that assigns $\textit{Rhombus}^{\Phi(E)} = \{x \mid \textit{Quadrilateral}(x) \wedge \textit{EqualSides}(x)\}^E = \{b\}$.

Definition 5.4 ($\Phi(\varphi)$). Let φ be a formula over Σ and Φ a symbolic structure. Then $\Phi(\varphi)$ denotes the formula over Υ obtained by replacing each occurrence of an atom $P(\bar{t})$ in φ by $\psi[\bar{x}/\bar{t}]$, where $P^\Phi = \{\bar{x} \mid \psi\}$.

The following proposition relates models of φ with models of $\Phi(\varphi)$.

PROPOSITION 5.5. *For every formula* φ *over* Σ*, symbolic structure* Φ*,* Υ*-structure* E *and variable assignment* θ*,* $(\Phi(E))\theta \models \varphi$ *iff* $E\theta \models \Phi(\varphi)$.

PROOF. If φ is the atomic formula $P(\bar{y})$ and $P^\Phi = \{\bar{x} \mid \psi\}$, then $\Phi(E)\theta \models P(\bar{y})$ iff $\theta(\bar{y}) \in P^{\Phi(E)}$ iff $\theta(\bar{y}) \in \{\bar{x} \mid \psi\}^E$ iff $E\theta[\bar{x}/\theta(\bar{y})] \models \psi$ iff $E\theta \models \psi[\bar{x}/\bar{y}]$ iff $E\theta \models \Phi(P(\bar{y}))$. The cases were φ is not atomic easily follow by induction. □

Example 5.6. Let Σ, Υ, Φ, and E be as in Example 5.3, and let φ be the sentence $\exists y\ \textit{Rhombus}(y)$. Then $\Phi(\varphi)$ is the sentence $\exists y\ (\textit{Quadrilateral}(y) \wedge \textit{EqualSides}(y))$. Clearly, $E \models \Phi(\varphi)$ and $\Phi(E) \models \varphi$.

Recall that $\mathrm{tf}(\Sigma)$ is the vocabulary of the two-valued encoding of a four-valued structure over Σ and that four-valued Σ-structures $\tilde{I}$ correspond in a one-to-one way with two-valued $\mathrm{tf}(\Sigma)$-structures $\mathrm{tf}(\tilde{I})$.

Definition 5.7 (*Symbolic four-valued structure* $\tilde{\Phi}$). A symbolic four-valued Σ-structure $\tilde{\Phi}$ (in terms of Υ) is a symbolic structure of $\mathrm{tf}(\Sigma)$ (in terms of Υ).

Such a structure $\tilde{\Phi}$ can be used to map a two-valued Υ-structure E to a four-valued Σ-structure $\tilde{I}$, namely the structure such that $\mathrm{tf}(\tilde{I}) = \tilde{\Phi}(E)$. Abusing notation, we will identify $\tilde{\Phi}(E)$ with the four-valued Σ-structure that it encodes.

Given a formula φ over Σ, the pair $(\varphi_{\mathrm{ct}}, \varphi_{\mathrm{cf}})$ consists of formulas over $\mathrm{tf}(\Sigma)$. Applying a four-valued symbolic Σ-structure $\tilde{\Phi}$ in terms of Υ to this pair maps it to a pair of formulas over Υ, namely to $(\tilde{\Phi}(\varphi_{\mathrm{ct}}), \tilde{\Phi}(\varphi_{\mathrm{cf}}))$. In the sequel, we denote this pair as $\tilde{\Phi}(\varphi)$.

By evaluating this $\tilde{\Phi}(\varphi)$ in a Υ-structure E and variable assignment θ, we obtain a pair of thruth-values which encodes for a four-valued truth value. Alternatively, evaluating φ directly in the four-valued Σ-structure $\tilde{\Phi}(E)$ and θ also yields a four-valued truth value. Combining Proposition 5.5 and Proposition 2.5 one can observe that these truth values are identical. That is, for every Υ-structure E and variable assignment θ, $\tilde{\Phi}(E)\theta(\varphi) = E\theta(\tilde{\Phi}(\varphi))$. In other words, to evaluate φ in structure $\tilde{\Phi}(E)$ and variable assignment θ, one can first evaluate φ symbolically in $\tilde{\Phi}$ and then in $E\theta$.

Example 5.8. Let Σ be $\{\textit{Module}/1, Course/1, \textit{Selected}/1, \textit{In}/2, \textit{MutExcl}/2\}$ (the vocabulary from Example 3.12) and let Υ be $\{\textit{Module}/1, \textit{In}/2, \textit{MutExcl}/2, \textit{Selected}_{\,\mathrm{ct}}/1\}$. Let $\tilde{I}_0$ be the Σ-structure from Example 3.12 and let E be the two-valued Υ-structure that assigns $\{c_1\}$ to $\textit{Selected}_{\,\mathrm{ct}}$ and corresponds to $\tilde{I}_0$ on the symbols of $\Sigma \cap \Upsilon$. Define the four-valued symbolic Σ-structure $\tilde{\Phi}$ over Υ by

$$(\textit{In}_{\,\mathrm{ct}})^{\tilde{\Phi}} = \{(c,m) \mid \textit{In}\,(c,m)\} \qquad (\textit{MutExcl}_{\,\mathrm{ct}})^{\tilde{\Phi}} = \{(x,y) \mid \textit{MutExcl}\,(x,y)\}$$
$$(\textit{In}_{\,\mathrm{cf}})^{\tilde{\Phi}} = \{(c,m) \mid \neg\textit{In}\,(c,m)\} \qquad (\textit{MutExcl}_{\,\mathrm{cf}})^{\tilde{\Phi}} = \{(x,y) \mid \neg\textit{MutExcl}\,(x,y)\}$$
$$(\textit{Module}_{\,\mathrm{ct}})^{\tilde{\Phi}} = \{m \mid \textit{Module}\,(m)\} \qquad (\textit{Course}_{\,\mathrm{ct}})^{\tilde{\Phi}} = \{m \mid \textit{Course}\,(m)\}$$
$$(\textit{Module}_{\,\mathrm{cf}})^{\tilde{\Phi}} = \{m \mid \neg\textit{Module}\,(m)\} \qquad (\textit{Course}_{\,\mathrm{cf}})^{\tilde{\Phi}} = \{m \mid \neg\textit{Course}\,(m)\}$$
$$(\textit{Selected}_{\,\mathrm{ct}})^{\tilde{\Phi}} = \{c \mid \textit{Selected}_{\,\mathrm{ct}}(c)\} \qquad (\textit{Selected}_{\,\mathrm{cf}})^{\tilde{\Phi}} = \{c \mid \mathit{false}\}$$

It can be checked that $\tilde{\Phi}(E)$ corresponds to $\tilde{I}_0$. Let φ be the sentence

$$\forall c \forall m\ (\neg\textit{Selected}\,(m) \vee \neg\textit{In}\,(m,c) \vee \textit{Selected}\,(c)).$$

Then φ_{ct} and φ_{cf} are given by, respectively,

$$\forall c \forall m\ (\textit{Selected}_{\,\mathrm{cf}}(m) \vee \textit{In}_{\,\mathrm{cf}}(m,c) \vee \textit{Selected}_{\,\mathrm{ct}}(c)),$$
$$\exists c \exists m\ (\textit{Selected}_{\,\mathrm{ct}}(m) \wedge \textit{In}_{\,\mathrm{ct}}(m,c) \wedge \textit{Selected}_{\,\mathrm{cf}}(c)).$$

The evaluation of φ_{ct} and φ_{cf} in $\tilde{\Phi}$ are, respectively, the sentences $\forall c \forall m\ (\mathit{false} \vee \neg\textit{In}\,(c,m) \vee \textit{Selected}_{\,\mathrm{ct}}(c))$ and $\exists c \exists m\ (\textit{Selected}_{\,\mathrm{ct}}(m) \wedge \textit{In}\,(c,m) \wedge \mathit{false})$. These two sentences are false in E, and therefore φ is unknown in $\tilde{I}_0$.

5.2. Symbolic Propagators

To construct the desired four-valued symbolic structures, we now lift propagators to the symbolic level.

Definition 5.9 (*Symbolic propagator* S). A *symbolic propagator* S for a theory T is an operator on the set of four-valued symbolic structures over Υ such that, for each Υ-structure E and symbolic structure $\tilde{\Phi}$, the following conditions are satisfied:

— $\tilde{\Phi}(E) \leq_p S(\tilde{\Phi})(E)$
— for every model M of T such that $\tilde{\Phi}(E) \leq_p M$, also $S(\tilde{\Phi})(E) \leq_p M$.

Note that these two conditions on symbolic propagators are similar to the conditions on non-symbolic propagators. As is the case for non-symbolic propagators, the composition $S_2 \circ S_1$ of two symbolic propagators for theory T is again a symbolic propagator for T.

We say that a symbolic propagator S *describes* a non-symbolic propagator O if for every symbolic four-valued structure $\tilde{\Phi}$ over Υ and every Υ-structure E, $S(\tilde{\Phi})(E) = \tilde{\Phi}(O(E))$. It is straightforward to check that if S_1 describes O_1 and S_2 describes O_2, then $S_2 \circ S_1$ describes $O_2 \circ O_1$. It follows that symbolic propagators can be used to

describe finite refinement sequences. Indeed, let V be a set of propagators such that for each $O \in V$, there exists a symbolic propagator S describing O. Let $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$ be a V-refinement sequence from $\tilde{\Phi}(E)$ and denote by O_i a propagator such that $O_i(\tilde{J}_i) = \tilde{J}_{i+1}$. Then $\tilde{J}_n = S_{n-1}(\ldots(S_0(\tilde{\Phi}))\ldots)(E)$ where S_i denotes the symbolic propagator that describes O_i for $0 \leq i < n$. As such, $S_{n-1}(\ldots(S_0(\tilde{\Phi}))\ldots)$ can be seen as a symbolic representation of the refinement sequence $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$. To describe transfinite refinement sequences with symbolic propagators, we would in general need symbolic Σ-structures that assign queries over an infinitary logic to the symbols of Σ.

We now introduce symbolic INF propagators. For the rest of this section, let $\tilde{\Phi}$ be a four-valued symbolic Σ-structure over Υ and E a Υ-structure. If two queries $\{\overline{x} \mid \psi\}$ and $\{\overline{y} \mid \chi\}$ have the same arity, i.e., $|\overline{x}| = |\overline{y}|$, we denote by $\{\overline{x} \mid \psi\} \cup \{\overline{y} \mid \chi\}$ the query $\{\overline{z} \mid \psi[\overline{x}/\overline{z}] \vee \chi[\overline{y}/\overline{z}]\}$, where $\overline{z}$ is a tuple of new variables. Note that $(\{\overline{x} \mid \psi\} \cup \{\overline{y} \mid \chi\})^E = \{\overline{x} \mid \psi\}^E \cup \{\overline{y} \mid \chi\}^E$ for every structure E.

Definition 5.10 (*Symbolic INF propagator*). Let φ be the INF sentence $\forall \overline{x}\ (\psi \Rightarrow P(\overline{x}))$. The *symbolic INF propagator* $\mathscr{I}_s^{\varphi}$ is defined by

$$\begin{array}{rcl} P_{\mathrm{ct}}^{\mathscr{I}_s^{\varphi}(\tilde{\Phi})} & = & P_{\mathrm{ct}}^{\tilde{\Phi}} \cup \{\overline{x} \mid \tilde{\Phi}(\psi_{\mathrm{ct}})\} \\ Q^{\mathscr{I}_s^{\varphi}(\tilde{\Phi})} & = & Q^{\tilde{\Phi}} \text{if } Q \neq P_{\mathrm{ct}}. \end{array}$$

That is, the queries for predicates different from P_{ct} are copied from $\tilde{\Phi}$ and the query for P_{ct} is extended with the mapping upon vocabulary Υ of ψ_{ct}. If φ is the INF sentence $\forall \overline{x}\ (\psi \Rightarrow \neg P(\overline{x}))$, then $\mathscr{I}_s^{\varphi}$ is defined by

$$\begin{array}{rcl} P_{\mathrm{cf}}^{\mathscr{I}_s^{\varphi}(\tilde{\Phi})} & = & P_{\mathrm{cf}}^{\tilde{\Phi}} \cup \{\overline{x} \mid \tilde{\Phi}(\psi_{\mathrm{ct}})\} \\ Q^{\mathscr{I}_s^{\varphi}(\tilde{\Phi})} & = & Q^{\tilde{\Phi}} \text{if } Q \neq P_{\mathrm{cf}}. \end{array}$$

The following result states the desired property that symbolic INF propagators are the symbolic counterpart of non-symbolic INF propagators.

PROPOSITION 5.11. *$\mathscr{I}_s^{\varphi}$ describes $\mathscr{I}^{\varphi}$ for every INF sentence φ.*

PROOF. We prove the case where φ is of the form $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$ and $L[\overline{x}]$ is a positive literal. The proof is similar in case $L[\overline{x}]$ is a negative literal.

Let E be a Υ-structure and $\tilde{\Phi}$ a four-valued symbolic Σ-structure over Υ. Let φ be the INF sentence $\forall \overline{x}\ (\psi \Rightarrow P(\overline{x}))$. We have to show that $\mathscr{I}_s^{\varphi}(\tilde{\Phi})(E) = \mathscr{I}^{\varphi}(\tilde{\Phi}(E))$. Therefore, we must prove that $Q^{\mathrm{tf}(\mathscr{I}_s^{\varphi}(\tilde{\Phi})(E))} = Q^{\mathrm{tf}(\mathscr{I}^{\varphi}(\tilde{\Phi}(E)))}$ for every predicate $Q \in \mathrm{tf}(\Sigma)$.

First assume $Q \neq P_{\mathrm{ct}}$. Then the following is a correct chain of equations.

$$Q^{\mathrm{tf}(\mathscr{I}^{\varphi}(\tilde{\Phi}(E)))} = Q^{\mathrm{tf}(\tilde{\Phi}(E))} = (Q^{\tilde{\Phi}})^E = (Q^{\mathscr{I}_s^{\varphi}(\tilde{\Phi})})^E = Q^{\mathrm{tf}(\mathscr{I}_s^{\varphi}(\tilde{\Phi})(E))} \tag{20}$$

The first and third equality follow from the definitions of $\mathscr{I}^{\varphi}$, respectively $\mathscr{I}_s^{\varphi}$, and the assumption that $Q \neq P_{\mathrm{ct}}$. The second and the fourth equality apply the definition of $\tilde{\Phi}(E)$.

The following chain shows that also $(P_{\rm ct})^{{\rm tf}(\mathscr{I}_s^{\varphi}(\tilde{\Phi})(E))} = (P_{\rm ct})^{{\rm tf}(\mathscr{I}^{\varphi}(\tilde{\Phi}(E)))}$:

$$\begin{aligned}
(P_{\rm ct})^{{\rm tf}(\mathscr{I}_s^{\varphi}(\tilde{\Phi})(E))} &= \Big((P_{\rm ct})^{\mathscr{I}_s^{\varphi}(\tilde{\Phi})}\Big)^E \\
&= \Big((P_{\rm ct})^{\tilde{\Phi}} \cup \{\overline{x} \mid \tilde{\Phi}(\psi_{\rm ct})\}\Big)^E \\
&= \Big((P_{\rm ct})^{\tilde{\Phi}}\Big)^E \cup \Big(\{\overline{x} \mid \tilde{\Phi}(\psi_{\rm ct})\}\Big)^E \\
&= (P_{\rm ct})^{{\rm tf}(\tilde{\Phi}(E))} \cup \{\overline{d} \mid E[\overline{x}/\overline{d}] \models \tilde{\Phi}(\psi_{\rm ct})\} \\
&= (P_{\rm ct})^{{\rm tf}(\tilde{\Phi}(E))} \cup \{\overline{d} \mid (\tilde{\Phi}(E))[\overline{x}/\overline{d}])(\psi) \geq_p \mathbf{t}\} \\
&= (P_{\rm ct})^{{\rm tf}(\mathscr{I}^{\varphi}(\tilde{\Phi}(E)))}
\end{aligned}$$

The first equality follows from the definition of $\tilde{\Phi}(E)$, the second one from the definition of $\mathscr{I}_s^{\varphi}$, the third one from the definition of union of queries, the fourth one from the definition of $\tilde{\Phi}(E)$ and of query evaluation, the fifth one from Proposition 2.5, and the final one from the definition of $\mathscr{I}^{\varphi}$. □

Proposition 5.11 implies that one can execute the propagation algorithm (Algorithm 4.16) using symbolic INF propagators in step 2 instead of non-symbolic ones. We refer to this symbolic version of the algorithm as the *symbolic propagation algorithm*.

Example 5.12. Consider the INF formulas:

$$\forall y\ (\exists x(\textbf{\textit{MutExcl}}\,(x,y) \wedge \textbf{\textit{Selected}}\,(x)) \Rightarrow \neg\textbf{\textit{Selected}}\,(y)). \tag{21}$$

$$\forall m\ (\exists c(\neg\textbf{\textit{Selected}}\,(c) \wedge \textbf{\textit{In}}\,(c,m)) \Rightarrow \neg\textbf{\textit{Selected}}\,(m)). \tag{22}$$

These are (simplifications of) INF formulas derived from respectively formula (1) and (3). Let $\tilde{\Phi}_0$ be the symbolic structure $\tilde{\Phi}$ of Example 5.8. When applying $\mathscr{I}_s^{(21)}$ on $\tilde{\Phi}_0$, note that ψ_{ct} is $\exists x(\textbf{\textit{MutExcl}}_{\,ct}(x,y) \wedge \textbf{\textit{Selected}}_{\,ct}(x))$ and hence $\tilde{\Phi}_0(\psi_{ct}) = \exists x(\textbf{\textit{MutExcl}}\,(x,y) \wedge \textbf{\textit{Selected}}_{\,ct}(x))$. The result of the symbolic propagation is a symbolic structure $\tilde{\Phi}_1$ that equals $\tilde{\Phi}_0$ except for $\textbf{\textit{Selected}}_{\,cf}$ that is assigned $\{x|\mathit{false} \vee \exists z(\textbf{\textit{MutExcl}}\,(z,x) \wedge \textbf{\textit{Selected}}_{\,ct}(z))\}$. The course c_2 is an answer for this query in $\tilde{I}_0$, the structure from Example 3.12 in which the certainly selected course c_1 is exclusive with c_2. Applying the same propagator adds an already existing disjunct to the symbolic interpretation of $\textbf{\textit{Selected}}_{\,cf}$:

$$\{x|\mathit{false} \vee \exists z(\textbf{\textit{MutExcl}}\,(z,x) \wedge \textbf{\textit{Selected}}_{\,ct}(z)) \vee \exists z(\textbf{\textit{MutExcl}}\,(z,x) \wedge \textbf{\textit{Selected}}_{\,ct}(z))\} \tag{23}$$

So repeated application of this symbolic propagator stabilizes.

Next, symbolic propagation of (22) from $\tilde{\Phi}_0$ will update the symbolic interpretation of $\textbf{\textit{Selected}}_{\,cf}$ by adding the symbolic interpretation of $\exists z(\textbf{\textit{Selected}}_{\,cf}(z) \wedge In_{ct}(z,s))$ as a new disjunct. This yields the assignment

$$\begin{aligned}
\{x \mid\ & \mathit{false} \vee \exists z(\textbf{\textit{MutExcl}}\,(z,x) \wedge \textbf{\textit{Selected}}_{\,ct}(z)) \vee \\
& \exists z(\mathit{false} \vee \exists z_1(\textbf{\textit{MutExcl}}\,(z_1,z) \wedge \textbf{\textit{Selected}}_{\,ct}(z_1)) \wedge In(z,x))\}
\end{aligned} \tag{24}$$

The module m_2 is an answer to this query in structure $\tilde{I}_0$. New propagation steps with the same formula yields more and more complex formulas and does not stabilize. The reason is that the derived predicate $\textbf{\textit{Selected}}_{\,cf}$ occurs in the condition $\psi_{\rm ct}$ of this rule.

Symbolic INF propagation takes a theory T and symbolic structure Φ as input, and its answer yields sound propagations from all concrete structures $\tilde{I}$ that are repre-

sented by $\tilde{\Phi}$, including infinite ones. Hence, this method can be used to solve tasks where non-symbolic INF propagation does not apply, e.g., when no structure $\tilde{I}$ is given or when $\tilde{I}$ is infinite. Wittocx et al. [2010] gives a detailed description of an implementation of the symbolic propagation algorithm that uses first-order binary decision diagrams for the representation of formulas and queries [Goubault 1995].

For finite theories, computing $\mathscr{I}_s^{\varphi}(\tilde{\Phi})$ takes time $\mathcal{O}(|\varphi| \cdot |\tilde{\Phi}|)$ while computing $\mathscr{I}^{\varphi}(\tilde{I})$ for finite $\tilde{I}$ takes time $\mathcal{O}(|\tilde{I}|^{|\varphi|})$ since evaluating a formula φ in a structure I takes time $\mathcal{O}(|I|^{|\varphi|})$ [Grädel et al. 2007]. This shows that, if formulas φ and Φ are not too large, applying a symbolic INF propagator can be done much more efficiently than applying a non-symbolic one. However, it also shows that iterating symbolic propagation steps blows up the symbolic structures at an exponential rate. In practice, it is important to simplify symbolic structures, by replacing the queries assigned by a structure by equivalent, but smaller queries. For example, (23) could be replaced by the shorter, equivalent query $\{x|\exists z(\boldsymbol{MutExcl}\,(z,x) \wedge \boldsymbol{Selected}_{ct}(z))\}$. First order binary decision diagrams turn out to be quite useful for simplifying FO formulas [Wittocx et al. 2010]. However, no simplication algorithm can prevent in general the exponential growth of the symbolic structures. Moreover, testing whether a sequence of symbolic structures is stabilizing is undecidable, because it boils down to testing logical equivalence of FO formulas. In contrast, the size of the result of a non-symbolic refinement sequence from $\tilde{I}$ is always polynomial in the size of $\tilde{I}$ and stability is polynomially decidable.

Fortunately, even a small number of applications of symbolic propagations may often yield valuable information. For example, consider a problem where, for a given graph $Edge$, some subgraph P must be computed as expressed by:

$$\forall x \forall y(\neg P(x,y) \vee Edge(x,y))$$

Let Υ include $Edge, P_{cf}$. One step of one of the symbolic INF propagators of this clause infers $(P_{cf})^{\Phi} = \{(x,y) \mid \neg Edge(x,y)\}$. In the context of a concrete $\tilde{I}$, this may reduce the number of unknown domain atoms $P(d_1,d_2)$ from $\#(D)^2$ where D is the domain, to $\#(Edge^{\tilde{I}})$. This may be a large reduction, even infinite if D is infinite and $Edge^{\tilde{I}}$ finite.

Symbolic propagation may be interesting in applications where a relatively small number of steps yield valuable information about predicates, where precision is less important than efficiency, and where the evaluation $\tilde{\Phi}(\tilde{I})$ of the computed symbolic structure $\tilde{\Phi}$ need not be computed completely. Grounding (Section 7.2) and approximate query answering (Section 7.5) are two examples of such applications.

6. PROPAGATION FOR FO(ID)

Inductive definitions appear in many real-life computational problems such as automated planning or problems involving dynamic systems [Denecker and Ternovska 2007; 2008]. A famous example of a concept that is inductively definable but not expressible in FO is the concept of reachability in a graph. In fact many inductively (i.e., recursively) definable concepts cannot be expressed in FO. In this section, we extend the propagation algorithm to the logic FO(ID) [Denecker 2000; Denecker and Ternovska 2008], an extension of FO with inductive, i.e., recursive, definitions.

The logic FO(ID) extends FO with a language construct to represent the most common forms of inductive definitions in mathematics: monotone inductive definitions and (potentially non-monotone) definitions by induction over a well-founded order (e.g., the standard definitions of the satisfaction relation $\models$ of FO). Conform the conventions of informal mathematical language, definitions in FO(ID) are represented as sets of formal rules. Starting with [Denecker 1998] and refined later in [Denecker et al. 2001; Denecker and Ternovska 2008], Denecker et al. argued that the well-founded seman-

tics uniformally formalizes the above two types of definitions. The key argument is that the construction of the well-founded model correctly formalizes monotone inductions as well as (potentially non-monotone) inductions over a well-founded order.

6.1. Inductive Definitions

A *definition* Δ is a finite set of rules of the form $\forall \overline{x}\ (P(\overline{x}) \leftarrow \varphi[\overline{y}])$. Predicates that appear in the head of a rule of Δ are called *defined predicates* of Δ. The set of all defined predicates of Δ is denoted $\mathrm{Def}(\Delta)$. All other symbols are called *open* with respect to Δ. The set of all open symbols of Δ is denoted $\mathrm{Open}(\Delta)$.

Example 6.1. The following definition defines the predicate *Reach* in terms of an open predicate *Edge* .

$$\left\{ \begin{array}{l} \forall x \forall y\ (\textit{Reach}\,(x,y) \leftarrow \textit{Edge}\,(x,y)), \\ \forall x \forall y\ (\textit{Reach}\,(x,y) \leftarrow \exists z\ (\textit{Reach}\,(x,z) \wedge \textit{Reach}\,(z,y))) \end{array} \right\} \tag{25}$$

Informally, this definition expresses that y can be reached from x in the graph represented by *Edge* , if either there is an edge between x and y, i.e., *Edge* (x,y) is true, or if there is some intermediate node z such that z can be reached from x and y can be reached from z.

The well-founded semantics [Van Gelder et al. 1991] formalizes the informal semantics of rule sets as inductive definitions. We borrow the presentation of this semantics from Denecker and Vennekens [2007].

Definition 6.2. Let Δ be a definition and $\tilde{I}$ a finite three-valued structure. A *well-founded induction for* Δ *extending* $\tilde{I}$ is a (possibly transfinite) sequence $\langle \tilde{J}_\xi \rangle_{0 \leq \xi \leq \alpha}$ of three-valued structures such that

(1) $\tilde{J}_0 = \tilde{I}|_{\mathrm{Open}(\Delta)} + \bot^{\leq_p}_{\mathrm{Def}(\Delta)}$;

(2) For every successor ordinal $\xi+1 \leq \alpha$, $\tilde{J}_{\xi+1}$ relates to $\tilde{J}_\xi$ in one of the following ways:
   (a) $\tilde{J}_{\xi+1} = \tilde{J}_\xi[V/\mathbf{t}]$, where V is a set of domain atoms such that for each $P(\overline{d}) \in V$, $P^{\tilde{J}_\xi}(\overline{d}) = \mathbf{u}$ and there exists a rule $\forall \overline{x}\ (P(\overline{x}) \leftarrow \varphi)$ in Δ such that $\tilde{J}_\xi[\overline{x}/\overline{d}](\varphi) = \mathbf{t}$.
   (b) $\tilde{J}_{\xi+1} = \tilde{J}_\xi[U/\mathbf{f}]$, where U is a set of domain atoms, such that for each $P(\overline{d}) \in U$, $P^{\tilde{J}_\xi}(\overline{d}) = \mathbf{u}$ and for all rules $\forall \overline{x}\ (P(\overline{t}) \leftarrow \varphi)$ in Δ, $\tilde{J}_{\xi+1}[\overline{x}/\overline{d}](\varphi) = \mathbf{f}$.

(3) For every limit ordinal $\lambda \leq \alpha$: $\tilde{J}_\lambda = \mathrm{lub}_{\leq_p}(\{\tilde{J}_\xi \mid \xi < \lambda\})$;

Intuitively, (2a) says that domain atoms $P(\overline{d})$ can be made true if there is a rule with $P(\overline{x})$ in the head and body φ such that φ is already true, given a variable assignment that interprets $\overline{x}$ by $\overline{d}$. On the other hand (2b) explains that $P(\overline{d})$ can be made false if there is no possibility of making a corresponding body true, except by circular reasoning. The set U, called an *unfounded set*, is a witness to this: making all atoms in U false also makes all corresponding bodies false.

A well-founded induction is called *terminal* if it cannot be extended anymore. The limit of a terminal well-founded induction is its last element. Denecker and Vennekens [2007] show that each terminal well-founded induction for Δ extending $\tilde{I}$ has the same limit, which corresponds to the well-founded model of Δ extending $\tilde{I}|_{\mathrm{Open}(\Delta)}$. The well-founded model is denoted by $\mathrm{wfm}_\Delta(\tilde{I})$. In general, $\mathrm{wfm}_\Delta(\tilde{I})$ is three-valued.

A two-valued structure I satisfies definition Δ, denoted $I \models \Delta$, if $I = \mathrm{wfm}_\Delta(I)$. The extension of FO with inductive definition is called FO(ID). An FO(ID) theory is a set of FO sentences and definitions. A two-valued structure satisfies an FO(ID) theory T if it satisfies every sentence and every definition of T.

The *completion* of a definition Δ is an FO(ID) theory that is weaker than Δ:

Definition 6.3. The *completion* of a definition Δ is the FO theory that contains for every $P \in \mathrm{Def}(\Delta)$ the sentence

$$\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow (\exists \overline{y}_1(\overline{x} = \overline{y}_1 \wedge \varphi_1) \vee \ldots \vee \exists \overline{y}_n(\overline{x} = \overline{y}_n \wedge \varphi_n))),$$

where $\forall \overline{y}_1\ (P(\overline{y}_1) \leftarrow \varphi_1), \ldots, \forall \overline{y}_n\ (P(\overline{y}_n) \leftarrow \varphi_n)$ are the rules in Δ with P in the head.

We denote the completion of Δ by $\mathrm{Comp}(\Delta)$. If T is an FO(ID) theory then we denote by $\mathrm{Comp}(T)$ the result of replacing in T all definitions by their completion. The following result states that the completion of T is weaker than T.

THEOREM 6.4 ([DENECKER AND TERNOVSKA 2008]). $\Delta \models \mathrm{Comp}(\Delta)$ *and* $T \models \mathrm{Comp}(T)$ *for every definition* Δ *and* FO(ID) *theory* T.

6.2. Propagation for Definitions

In this section, we consider two approaches to extend the propagation method to FO(ID). First, we discuss the application of our propagation method for FO on the completion of FO(ID) theories. Secondly, we define an INF propagator for definitions.

6.2.1. Propagation on the Completion. It follows from Theorem 6.4 that the propagators obtained by applying Algorithm 4.16 on $\mathrm{Comp}(T)$ are propagators for the theory T. However, note that a complete propagator for $\mathrm{Comp}(T)$ can be incomplete for T. For example, consider the definition $\Delta := \{P \leftarrow P\}$. This definition has only one model, in which P is false. Hence, $(\mathscr{O}^{\Delta}(\bot^{\leq_p}))(P) = \mathbf{f}$. The completion of Δ is the sentence $(P \Leftrightarrow P)$, which has a model making P true and one making P false. Therefore $(\mathscr{O}^{\mathrm{Comp}(\Delta)}(\bot^{\leq_p}))(P) = \mathbf{u}$. We conclude that $\mathscr{O}^{\mathrm{Comp}(\Delta)} \neq \mathscr{O}^{\Delta}$. Moreover, $\mathscr{O}^{\mathrm{Comp}(\Delta)}$ is not inducing for Δ, that is, it may not recognize that a given two-valued structure is not a model of Δ.

If $\tilde{I}$ is a finite structure and T an FO(ID) theory, then there exists an FO theory T' such that the models of T approximated by $\tilde{I}$ are precisely the models of T' approximated by $\tilde{I}$. Such a theory can be constructed by applying propositionalization (see, e.g., [Wittocx et al. 2010]), followed by the transformations described by Janhunen [2004] or by Pelov and Ternovska [2005]. Propagation on T and $\tilde{I}$ can then be obtained by applying propagation on T' and $\tilde{I}$. The benefit of this approach is a gain in precision. In particular, the resulting propagator is inducing. On the other hand, T' is considerably larger than T, which has repercussions on the efficiency of (symbolic) propagation.

6.2.2. Propagators for Definitions. A second way to extend our propagation method to FO(ID) is to introduce special purpose propagators involving definitions.

Definition 6.5. The propagator $\mathscr{I}^{\Delta}$ for a definition Δ is defined by

$$P^{\mathscr{I}^{\Delta}(\tilde{I})}(\overline{d}) = \begin{cases} \mathrm{lub}_{\leq_p}\{\mathbf{t}, P^{\tilde{I}}\} & \text{if } P^{\mathrm{wfm}_{\Delta}(\tilde{I})}(\overline{d}) = \mathbf{t} \\ \mathrm{lub}_{\leq_p}\{\mathbf{f}, P^{\tilde{I}}\} & \text{if } P^{\mathrm{wfm}_{\Delta}(\tilde{I})}(\overline{d}) = \mathbf{f} \\ P^{\tilde{I}} & \text{otherwise.} \end{cases}$$

It follows from the definition of well-founded induction that $\mathscr{I}^{\Delta}$ is a monotone propagator for every definition Δ. Moreover, for finite structures $\tilde{I}$, $\mathrm{wfm}_{\Delta}(\tilde{I})$ can be computed in polynomial time in $|\tilde{I}|$. As such, $\mathscr{I}^{\Delta}$ is a propagator with polynomial-time data complexity. Note that this propagator only propagates information from the body of the definition to the head; to propagate from head to body, one needs propagators derived from the completion.

It is an open question whether the propagator $\lim_{\mathscr{J}(V)}$ can be represented by a (positive or monotone) rule set if V may contain both INF sentences and definitions. Results from fixpoint logics (see, e.g., [Grädel et al. 2007]) suggest that this will be possible when only finite structures are considered, but impossible in general. We expect that even if it is possible to represent $\lim_{\mathscr{J}(V)}$ by a rule set Δ_V, this result will not be of practical importance, since Δ_V will be rather complicated. The same remark applies for symbolic propagators simulating $\lim_{\mathscr{J}(V)}$.

Recently, Vlaeminck et al. [2010] showed how to represent the propagator $\lim_{\mathscr{J}(V)}$ by a *nested fixpoint definition* [Hou et al. 2010], a rule-based logic of the family of nested least fixpoint logic [Libkin 2004]. The extent to which theoretical and practical results about monotone rule sets can be adapted to nested fixpoint definitions will determine the usefulness of representing $\lim_{\mathscr{J}(V)}$ by such definitions.

7. APPLICATIONS

The constraint propagation framework has a broad potential. We report on several applications and systems that were built using it for finite model generation, improved grounding, approximate solving of universal second-order logic ($\forall$SO) model generation problems, declarative programming of configuration systems, and approximate query answering in incomplete databases. For details we refer to the appropriate publications.

7.1. Model Expansion for Solving Constraint Satisfaction Problems

In the context of CSP, the obvious role for constraint propagation is as a component of a backtracking procedure to solve CSPs. In the context of logic, the corresponding role is as a component of a model expansion solver. Many real-life computational problems are naturally cast as CSPs. Well-known examples are scheduling, planning, diagnosis, and lightweight dynamic system verification. Using expressive logics to express constraints in such problems often leads to natural and more compact specifications.

At this moment, several MX solvers for extensions of FO have been built: ps-grnd/aspps [East and Truszczynski 2001], the IDP system [Wittocx et al. 2008c] and the Enfragmo system [Aavani et al. 2012]. They all include some support for inductive definitions, aggregates, arithmetic, as well as other language extensions. The IDP system is the first and, at present, only system that supports full FO(ID) and, in the 2011 ASP system competition, scored as good as the best ASP solver for NP search problems [Calimeri et al. 2011].

Similarly as for solving CSPs, a model expansion solver for FO or FO(ID) could be built by integrating an implementation of our propagation method and branching heuristics in a backtracking algorithm. In fact, our constraint propagation method is used in two state-of-the-art MX solvers (see next section) but not for propagation. Current MX(FO(ID)) solvers are built following an approach similar to current ASP systems. First, they *ground* the input theory and input structure $\tilde{I}$ to an equivalent propositional theory T_g. Next, an (extended) SAT solver is applied to efficiently find a model for T_g. If found, this model then corresponds to a solution of the original problem. Thus, constraint propagation is implemented at the propositional level, by unit propagation. The benefit is that highly optimized SAT technology can be exploited.

Still, in some special contexts our constraint propagation method can be useful for MX solving. One such use is illustrated in [Wittocx et al. 2009], in the context of debugging FO(ID) theories. Discovering why a solver unexpectedly answers "unsatisfiable" is often hard for the programmer. In that paper, a debugging method is developed which extracts from a failed run of the solvers a proof of inconsistency. The core inference rules of this proof correspond to the propagators of INF formulas associated to the

theory T. In this context, where efficiency is often not so important, a prototype MX solver was built along the lines sketched above. It is a backtracking solver including INF propagators that operate on BDD's representing lower- and upperapproximations of formulas of T.

A potential future application of FO constraint propagation is to alleviate a problem of all grounding MX solvers in large problems: that the grounding explodes. The idea is to extend a SAT solver with FO constraint propagation for subsets of T in a way similar to DPLL(T) in SMT solvers [Nieuwenhuis et al. 2006]. In this combined approach, some sentences of the theory are grounded to propositional theory T_g, while the others — those with a large grounding— are transformed to a set of INF sentences V. Next, SAT solving is applied on T_g. Whenever the solver derives that a certain propositional literal L is true, and L unifies with a literal in the condition of an INF φ sentence in V, $\mathscr{I}^{\varphi}$ can be applied to derive the truth of other literals. In turn, these literals can be communicated back to the SAT solver.

7.2. Improved Grounding

While FO(ID) propagation is not used for propagation in current MX(FO(ID)) solvers, it is used already in two grounders to reduce grounding size and time [Wittocx et al. 2010]. In the context of a model expansion problem with input theory T and structure $\tilde{I}$, one obviously needs to ground only those instances of subformulas of T that are unknown in $\tilde{I}$. Exploiting this may substantially reduce the size of the grounding T_g. Obviously, the more precise $\tilde{I}$ is, the less subformulas need to be grounded and the smaller T_g can be. Thus, applying FO(ID) constraint propagation to refine $\tilde{I}$ into $\tilde{J}$ and grounding in $\tilde{J}$ may lead to reduced grounding size and time.

Both symbolic and non-symbolic propagation have been used for this. The first application of this idea was in the context of the grounder GIDL [Wittocx et al. 2010], the grounder of IDP. There, we opted for the symbolic propagation algorithm. Experiments with GIDL show that the time taken by symbolic propagation is negligible compared to the overall grounding time, while on average, it reduces grounding size and time by 30%, respectively 40%. In some cases, symbolic propagation makes the difference between being able to ground a theory in less than 20 seconds, compared to not being able to ground it at all.

Recently also non-symbolic propagation was used for the same purpose, in the context of Enfragmo [Vaezipoor et al. 2011]. This method applies semi-naive bottom up evaluaton on the rule set Δ_V associated to $INF(T)$. Since this method computes a fixpoint whereas the symbolic method does not, a more precise $\tilde{I}$ is obtained, which for some benchmarks may lead to substantially smaller groundings and more efficient model expansion. The explanation is interesting: grounding with more precise structures often avoid the construction of *autarkies*, irrelevant subtheories whose solution is independent of the rest of the theory.

The strength of the symbolic propagation is that it is capable of computing bounds on (tuples of) variables in quantified formulas. For example, if it holds that $\forall x \forall y(p(x,y) \Rightarrow edge(x,y))$, where $edge$ is given, the symbolic method derives that $p(u,v)$ is false for all tuples outside $edge^{\tilde{I}}$, and this may yield a huge, even infinite reduction of the grounding. To improve this even further, work is underway in the context of IDP system to integrate decision techniques for Presburger arithmetic in symbolic propagation. Thus, also the symbolic method has its merits.

The strenghts of both methods can be easily combined. First, we perform symbolic propagation, producing a symbolic refinement $\tilde{J}$, then computing the concrete three-valued structure $\tilde{K}$ represented by $\tilde{J}$ in the context of the input structure $\tilde{I}$, and finally

applying non-symbolic propagation from $\tilde{K}$ to obtain the more refined fixpoint $\tilde{L}$ to be used for grounding.

7.3. Interactive Search Systems

The application presented in the introduction is an example of an *interactive configuration problem*. An interactive configuration problem is a constraint solving problem solved by the user in interaction with a software system that monitors a set of constraints. A broad range of problems in many areas are of this nature: configuring course programs, computers in an on-line shop, computer software, schedules, etc. As noted by Vlaeminck et al. [2009], knowledge in such problems is often complex and tends to evolve fast, making the construction and maintenance of such systems using procedural programming methods often very tedious. A considerable gain can be made by expressing the background knowledge – the constraints – in a logic knowledge base T and applying suitable automated reasoning methods to support the user in his search process. In particular, the user makes the choices and may decide to backtrack while the role of constraint propagation is the same as in constraint solving: feasible computation of implied facts and early discovery of inconsistencies. During the process, a three-valued structure $\tilde{I}$ is constructed, facts are chosen to be true or false by the user, and constraint propagation derives entailed facts from T to refine $\tilde{I}$.

The prototype presented in Vlaeminck et al. [2009] and available as "KB-system demo" at `http://dtai.cs.kuleuven.be/krr/software` is the first system that we know of that implements interactive configuration using an FO(AGG) knowledge base. The system has a GUI that allows students to select courses and modules. The checkboxes correspond to atoms of the underlying theory. The state of these checkboxes in the GUI corresponds to a three-valued structure. Constraint propagation on this structure is implemented using a combination of symbolic and non-symbolic propagation. The role of the GUI is mostly to communicate user choices to the IDP system and the computed propagations back to the user window. Also this can be automated to some extend. A GUI builder for configuration systems was built as a Java™library [Calus and Vennekens 2011]. It takes as input a theory, a set of atoms to be visualized as check- or dropboxes, graphical information where to put these on the screen, and it generates the GUI. The library is available from `http://dtai.cs.kuleuven.be/krr/software/configid`.

The underlying reasoner in this application is the IDP system. In this application its role is as a (finite domain) *knowledge base system* (KBS) in the sense described in [Denecker and Vennekens 2008]. The system is not only used for propagation, but applies several other forms of inference on the background theory T for supporting the user: model checking to check if the theory is satisfied in the selection obtained by disselecting all not explicitly selected courses, model expansion to automatically complete the current partial selection, model expansion with optimisation to compute an optimal completion according to some optimisation function, and generation of explanations when inconsistent choices are made. For more details we refer to [Vlaeminck et al. 2009]. This application shows that knowledge base systems for rich extensions of FO are feasible and can greatly reduce programming effort in knowledge intense applications where the complexity is more due to the underlying knowledge and the variety of the computational tasks than to the computational complexity of the latter. It is also a demonstration of the feasibility of a principle that is unique to declarative languages: the possibility of reusing the same logic theory to implement different functionalities by applying different forms of inference to it.

With respect to constraint propagation, there are two main, albeit somewhat contradictory requirements in such applications. First, since a configuration system is

interactive, the propagation should be efficient in order to respond sufficiently fast. Secondly, when an ideal system offers the user a choice on an atom (i.e., the atom is unknown), whatever value the user chooses, is a valid choice in the sense that the expanded partial structure can be completed to a model. To obtain this, the propagation has to be the complete propagator $\mathscr{O}^T$. Indeed, if $\tilde{J} = \mathscr{O}^T(\tilde{I})$ and a choice $P(\overline{d})$ is unknown in $\tilde{J}$, then there exists a model of T, i.e., a valid configuration, where $P(\overline{d})$ is true, and one where $P(\overline{d})$ is false. As such, neither selecting nor deselecting $P(\overline{d})$ is an invalid choice since in both cases a valid configuration remains reachable. The combination of both requirements shows the importance of investigating the precision of efficient propagators.

7.4. Approximate Solving of ∀SO Model Expansion Problems

Inspired by the approximate solution method of [Morales et al. 2007] for conformant planning in Answer Set Programming, Vlaeminck et al. [2010; 2012] developed a general approximate method for solving model expansion problems for ∀SO theories. Whereas MX(FO(ID)) problems with a fixed FO(ID) input theory and variable finite input structures are **NP** search problems, MX(∀SO) problems are Σ_2^P problems, and some of these problems contain Σ_2^P-hard subproblems.

The approximate method consists of translating a model expansion problem for a ∀SO theory into a stronger MX(FO(ID)) problem, in the sense that solutions of the latter are solutions of the former (but not necessarily vice versa). In this method, the use of the inductive propagation definition $\Delta_{INF(T)}$ is crucial. To give a hint of this method, consider a simplified conformant planning method with theory T describing effects of drinking from dirty glasses:

$$\begin{aligned} &\mathit{drink} \Rightarrow \neg\, \mathit{thirsty} \\ &\mathit{drink} \wedge \neg\, \mathit{dirty} \Rightarrow \neg\, \mathit{sick} \\ &\mathit{wipe} \Rightarrow \neg\, \mathit{dirty} \end{aligned}$$

A goal is to find a plan in terms of actions *drink* and *wipe* that guarantees that I am not sick and not thirsty, whether the glass is dirty or not. Formally, this is the model expansion problem of the ∀SO sentence $\forall dirty \forall thirsty \forall sick(T \Rightarrow \neg \mathit{thirsty} \wedge \neg \mathit{sick})$. A solution assigns values to *drink* and *wipe* so that this formula holds. Consider the following rule set, obtained as a subset of the rule set associated to $INF(T)$ in which the "action" predicates are unmodified:

$$\begin{aligned} &\mathit{thirsty}_{\,\mathrm{cf}} \leftarrow \mathit{drink} \\ &\mathit{sick}_{\,\mathrm{cf}} \leftarrow \mathit{drink} \wedge \mathit{dirty}_{\,\mathrm{cf}} \\ &\mathit{dirty}_{\,\mathrm{cf}} \leftarrow \mathit{wipe} \end{aligned}$$

It is easy to see that in models of this rule set that satisfy the planning goal $\mathit{thirsty}_{\,\mathrm{cf}} \wedge \mathit{sick}_{\,\mathrm{cf}}$, the interpretation of action predicates specifies a correct solution. Here there is only one model and *drink* and *wipe* are true in it. Vlaeminck et al. [2010; 2012] extend this method to MX(∀SO) problems and show that this method is quite succesful for conformant planning problems [Vlaeminck et al. 2010; Vlaeminck et al. 2012].

7.5. Approximate Query Answering

Incomplete databases may arise from the use of null values, or from integrating a collection of local databases each based on its own *local schema* into one virtual database over a *global schema* [Grahne and Mendelzon 1999], or it arises from using a restricted form of closed world assumption [Cortés-Calabuig et al. 2007], etc. In all these cases, the data complexity of certain and possible query answering is computationally hard (**coNP**, respectively **NP**). For this reason more efficient (and often very precise) poly-

nomial approximate query answering methods have been developed, which compute an underestimation of the certain, and an overestimation of the possible answers.

The tables of an incomplete database are naturally represented as a three-valued structure $\tilde{I}$. The integrity constraints, local closed world assumption or mediator scheme corresponds to a logic theory T. Answering a query $\{\overline{x} \mid \varphi\}$ boils down to computing the set of tuples $\overline{d}$ such that $M[\overline{x}/\overline{d}] \models \varphi$ in every model M of T approximated by $\tilde{I}$ (certain answers) and the set of tuples $\overline{d}$ such that $M[\overline{x}/\overline{d}] \models \varphi$ for at least one $M \models T$ approximated by $\tilde{I}$ (possible answers). These sets can be approximated by $\{\overline{d} \mid \tilde{J}[\overline{x}/\overline{d}](\varphi) = \mathbf{t}\}$, respectively $\{\overline{d} \mid \tilde{J}[\overline{x}/\overline{d}](\varphi) \neq \mathbf{f}\}$, where $\tilde{J}$ is obtained by applying constraint propagation for T from $\tilde{I}$. If a constraint propagation method with polynomial-time data complexity is used to compute $\tilde{J}$, computing the approximate query answers above also requires polynomial time in the size of the database. Of course, the more precise $\tilde{J}$ is, the more precise the obtained answers to the query are.

Our results suggest three different ways in which approximate query answering in such a general context could be implemented. A first, naive way is to apply the non-symbolic method to compute $\tilde{J}$ and evaluate φ in it. Although polynomial, this method seems unfeasible. First of all, the size of real-life databases makes the application of non-symbolic propagation often too slow in practice, since it requires the storage of large intermediate tables. More importantly, each time the data is changed, $\tilde{J}$ needs to be recomputed. Also, the computation of $\tilde{J}$ is not goal directed towards the query and may compute three-valued relations that are irrelevant for it. A second way is to run the symbolic propagation algorithm. Symbolic propagation can be used for query rewriting. Indeed, given a symbolic structure $\tilde{\Phi}$, computed by propagation, an evaluation structure E and a query $\{\overline{x}|\varphi\}$, the approximation to the certain answers for φ are given by the set $\{\overline{d} \mid \tilde{\Phi}(E)[\overline{x}/\overline{d}](\varphi) = \mathbf{t}\}$. This set is equal to $\{\overline{x} \mid (\tilde{\Phi}(\varphi))_{\mathrm{ct}}\}^E$. Hence the query $\{\overline{x} \mid \varphi\}$ can be rewritten to a new query $\{\overline{x} \mid (\tilde{\Phi}(\varphi))_{\mathrm{ct}}\}$, which is then evaluated in the database E. Next, one can use the various optimization strategies in current database management systems to efficiently compute the answers to the new query. Possible answers to φ are obtained in a similar way. Difficulties to be overcome here are that stabilization cannot be determined and queries in the symbolic structures may get very large after a number of iterations. The third and perhaps most promising method for approximate query answering is to use the rule set $\Delta_{INF(T)}$ representing the propagation process. We can view this rule set as an intensional database in terms of the given, extensional incomplete database, and use deductive database techniques for solving queries $\{\overline{x} \mid \varphi_{\mathrm{ct}}\}$ with respect to it. This method is goal directed and its results are more precise than the symbolic method because it computes the answer of the query in the fixpoint of the propagation process.

Applying the non-symbolic version of Algorithm 4.16 generalizes the algorithm of Cortés Calabuig et al. [2006] for approximate query answering in the context of incomplete databases with local closed world assumptions. Applying the symbolic version and rewriting the query generalizes the query rewriting technique presented by Cortés-Calabuig et al. [2007]. Our third method generalizes the rule-based method for this type of databases that was presented in [Denecker et al. 2010].

One important issue in the context of query answering is the incompleteness of the approximate query answering methods. Unless one can prove for the given theory T that the approximate method is optimally precise, its use is limited to applications where completeness is not strictly necessary (although in practice there may be many such applications). This stresses again the importance of analysing conditions that

ensure the completeness of the propagation methods. In the context of incomplete, locally closed databases, such conditions were investigated by Denecker et al. [2010].

8. CONCLUSIONS

In this paper we presented constraint propagation as a basic form of inference for FO theories. We introduced a general notion of constraint propagators and briefly discussed the complete propagator for a theory. Due to its high computational complexity, the complete propagator cannot be applied in most real-life applications. Therefore we investigated incomplete propagators, called INF propagators. These propagators generalize the propagators for propositional logic presented by McAllester [1990] and Apt [1999b]. Similar propagators were proposed in the context of locally closed databases, where approximative query answering in polynomial time was studied in a series of papers [Cortés Calabuig et al. 2006; Cortés-Calabuig et al. 2007; Cortés-Calabuig et al. 2008; Denecker et al. 2010]. A first version of INF propagators for full FO was presented in the context of grounding [Wittocx et al. 2008b]. Later we improved the propagators and presented them in a more general context [Wittocx et al. 2008a]. The link with constraint programming is new in the current paper. Besides their lower computational complexity, INF propagators for FO have other interesting properties: propagation using INF propagators can be represented by a monotone rule set and can be executed in a symbolic manner. The former property allows us to use existing systems and extensively studied methods to make efficient implementations of propagation. The latter property is important in contexts where data changes regularly or where only part of the results obtained by propagation is needed.

We extended the results about propagation using INF propagators to the logic FO(ID) that extends FO with inductive definitions. Whether the results about representation by a monotone definition or symbolic propagation carry over to inductive definitions, is an open question. Further transfer of techniques developed in the constraint programming community to improve propagation for FO, is also an interesting direction for future work.

FO and FO(ID) can also be extended with aggregates [Pelov et al. 2007]. In many cases, the use of aggregates yields more succinct theories [Simons et al. 2002], and often faster reasoning [Faber et al. 2008]. Aggregates appear in many real-life applications. The extension of our propagation method to aggregates is described in Appendix A.

Finally, we discussed applications of the constraint propagation methods in state-of-the-art systems, prototypes and applications and showed that constraint propagation has a broad range of applications.

APPENDIX

A. AGGREGATES

Aggregates are (partial) functions that have a set as argument. An example is the function CARD returning the cardinality of a set. Aggregates are very useful in many real-life applications. In many cases, the use of aggregates yields more succinct theories [Simons et al. 2002], and often faster reasoning [Faber et al. 2008]. Adding aggregates to FO is straightforward; however, aggregates are nonmonotonic and, just like negation, allowing them in inductive definitions poses a semantic problem that has been studied extensively in the context of Logic Programming and Answer Set Programming [Pelov et al. 2007; Faber et al. 2011]. In [Pelov et al. 2007], well-founded and stable semantics were extended to rule sets with aggregates and the logic FO(ID) was extended to FO(ID,AGG) allowing for recursion over (possibly nested) aggregates. In this appendix, we study propagation for FO(AGG).

In this section, we assume that theories contain two types of variables: the standard ones and numerical variables that range only over real numbers in the domain of the structure. For the sake of the complexity analysis, we assume that the domains of structures are finite and contain a finite set of positive real numbers, and that assignments $\theta(x)$ to numerical variables range over these numbers. Furthermore, we assume that formulas and structures are well-typed, in the sense that terms occurring at a position where only a number can sensibly occur, evaluate in every structure to a number. For instance, in a term $x+1$, x should be a numerical variable. Ternovska and Mitchell [2009] and Wittocx [2010] provide more detailed descriptions about including arithmetics in FO.

A.1. FO with Aggregates

We denote the extension of FO with aggregates by FO(AGG). The syntax of FO(AGG) is defined using the standard induction of terms and formulas in FO augmented with some new cases. A *set expression* is an expression of the form $\{\overline{x} \mid \varphi\}$, where $\overline{x}$ is a tuple of variables and φ a formula (that may contain free variables not in $\overline{x}$). In this paper, we consider the aggregate function symbols CARD, SUM, PROD, MIN and MAX. Terms are defined as usual. An *aggregate term* is an expression of the form $\text{CARD}(V)$, $\text{SUM}(V)$, $\text{PROD}(V)$, $\text{MIN}(V)$ or $\text{MAX}(V)$, where V is a set expression. Formulas are defined as usual with one new case: an *aggregate atom* is a formula of the form $x \leq \text{F}(V)$ or $x \geq \text{F}(V)$, where x is a numerical variable and $\text{F}(V)$ an aggregate term. Note that this syntax allows nested aggregates. Formulas of the form $t \leq \text{F}(V)$ and $t < \text{F}(V)$, where t is a term and $\text{F}(V)$ an aggregate term are seen as shorthands for the formulas $\exists x\ (t = x \wedge x \leq \text{F}(V))$, respectively $\exists x \exists y\ (t = x \wedge x < y \wedge y \leq \text{F}(V))$. Similarly for formulas of the form $t \geq \text{F}(V)$ and $t > \text{F}(V)$.

Next we define the value of terms, set expressions, aggregate terms and formulas in a structure $\tilde{I}$ under variable assignment θ. Structures $\tilde{I}$ are finite and contain a finite set of positive real numbers. Assignments $\theta(x)$ to numerical variables belong to this set. The definitions below extend the standard induction with some new cases.

The value of set expression $\{\overline{x} \mid \varphi\}$ in structure I under variable assignment θ is denoted by $I\theta(\{\overline{x} \mid \varphi\})$ and defined by $\{\overline{d} \mid I\theta[\overline{x}/\overline{d}] \models \varphi\}$.

An n-ary relation $\mathtt{V}$ on some domain is *numeric* if for each $(d_1, \dots) \in \mathtt{V}$, d_1 is a positive real number.

For an n-ary relation $\mathtt{V}$, we define $\text{CARD}(\mathtt{V})$ to be the cardinality of $\mathtt{V}$. If $\mathtt{V}$ is numeric, we define:

— $\text{SUM}(\mathtt{V}) = 0$ if $\mathtt{V} = \emptyset$ and $\text{SUM}(\mathtt{V}) = \sum_{(a_1,\dots,a_n)\in\mathtt{V}}(a_1)$ otherwise;
— $\text{PROD}(\mathtt{V}) = 1$ if $\mathtt{V} = \emptyset$ and $\text{PROD}(\mathtt{V}) = \prod_{(a_1,\dots,a_n)\in\mathtt{V}}(a_1)$ otherwise.

Essentially, the n-ary relation $\mathtt{V}$ is viewed here as a multi-set of the first arguments of its tuples containing each a as many times as $\mathtt{V}$ contains tuples $(a, \dots)$. Thus, we add and multiply only the first argument.

Furthermore, if $\mathtt{V}$ is a (finite) set of numbers, we define:

— $\text{MIN}(\mathtt{V}) = +\infty$ if V is empty and $\text{MIN}(\mathtt{V}) = glb(\mathtt{V})$ otherwise;
— $\text{MAX}(\mathtt{V}) = 0$ if V is empty and $\text{MAX}(\mathtt{V}) = lub(\mathtt{V})$ otherwise.

Let I be a structure with domain D. The satisfaction relation for FO(AGG) is defined by adding the following base cases to the satisfaction relation for FO:

— $I\theta \models x \leq \text{F}(V)$ if $\theta(x) \leq \text{F}(I\theta(V))$;
— $I\theta \models x \geq \text{F}(V)$ if $\theta(x) \geq \text{F}(I\theta(V))$.

To define the value of an aggregate atom in a three-valued structure, we first introduce three-valued sets. A *three-valued set* $\tilde{\mathtt{V}}$ in domain D is a function from D to $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. A three-valued set $\tilde{\mathtt{V}}$ *approximates* a class of sets $\mathtt{V}$, namely all sets consisting of all d such that $\tilde{\mathtt{V}}(d) = \mathbf{t}$ and a subset of elements d such that $\tilde{\mathtt{V}}(d) = \mathbf{u}$. For example, $\{a^{\mathbf{t}}, b^{\mathbf{f}}, c^{\mathbf{u}}\}$ denotes a three-valued set, approximating the sets $\{a\}$ and $\{a, c\}$.

In a three-valued structure $\tilde{I}$, a set expression $\{\overline{x} \mid \varphi\}$ evaluates under variable assignment θ to the three-valued set $\tilde{I}\theta(\{\overline{x} \mid \varphi\}) := \{\overline{d}^{\mathbf{v}} \mid \tilde{I}\theta[\overline{x}/\overline{d}](\varphi) = \mathbf{v}\}$. The *infimum value* $\tilde{I}\theta(\mathtt{F}(V))_{inf}$ of an aggregate term $\mathtt{F}(V)$ in $\tilde{I}$ under θ is defined by

$$\tilde{I}\theta(\mathtt{F}(V))_{inf} = glb\{n \mid n = \mathtt{F}(\mathtt{V}) \text{ for some } \mathtt{V} \text{ approximated by } \tilde{I}\theta(V)\}.$$

Similarly, the *supremum value* of $\mathtt{F}(V)$ is defined by

$$\tilde{I}\theta(\mathtt{F}(V))_{sup} = \mathrm{lub}\{n \mid n = \mathtt{F}(\mathtt{V}) \text{ for some } \mathtt{V} \text{ approximated by } \tilde{I}\theta(V)\}.$$

The truth value of an FO(AGG) formula φ in structure $\tilde{I}$ under variable assignment θ is defined by adding the following cases to the definition of the truth value of an FO formula:

— $\tilde{I}\theta(x \geq \mathtt{F}(V)) = \begin{cases} \mathbf{t} & \text{if } \theta(x) \geq \tilde{I}\theta(\mathtt{F}(V))_{sup} \\ \mathbf{u} & \text{if } \theta(x) \geq \tilde{I}\theta(\mathtt{F}(V))_{inf} \text{ and} \\ & \theta(x) < \tilde{I}\theta(\mathtt{F}(V))_{sup} \\ \mathbf{f} & \text{otherwise.} \end{cases}$

— $\tilde{I}\theta(x \leq \mathtt{F}(V)) = \begin{cases} \mathbf{t} & \text{if } \theta(x) \leq \tilde{I}\theta(\mathtt{F}(V))_{inf} \\ \mathbf{u} & \text{if } \theta(x) > \tilde{I}\theta(\mathtt{F}(V))_{inf} \text{ and} \\ & \theta(x) \leq \tilde{I}\theta(\mathtt{F}(V))_{sup} \\ \mathbf{f} & \text{otherwise.} \end{cases}$

Example A.1. Let $P^{\tilde{I}} = \{a^{\mathbf{t}}, b^{\mathbf{f}}, c^{\mathbf{u}}\}$. Then $\tilde{I}(\textsc{card}(\{x \mid P(x)\})_{inf} = 1$ and $\tilde{I}(\textsc{card}(\{x \mid P(x)\})_{sup} = 2$. It follows that $\tilde{I}(2 \geq \textsc{card}(\{x \mid P(x)\}) = \mathbf{u}$ while $\tilde{I}(1 > \textsc{card}(\{x \mid P(x)\}) = \mathbf{f}$.

In Pelov et al. [2007], it is shown that the value of an FO(AGG) formula in a three-valued structure $\tilde{I}$ can be computed in time polynomial in $|\tilde{I}|$.

A.2. Propagation for FO(AGG)

To extend the propagation method to theories containing aggregates, the definition of INF sentences is extended to include aggregates. As in the case of FO, a propagator with polynomial-time data complexity is associated to each of these sentences. In the next section, it is shown that every FO(AGG) theory over a vocabulary Σ can be converted to a Σ-equivalent theory of INF sentences.

Definition A.2. An FO(AGG) sentence φ is in *implicational normal form* (INF) if it is of the form $\forall\overline{x}(\psi \Rightarrow L[\overline{x}])$, where $L[\overline{x}]$ is a literal that does not contain an aggregate and ψ is a formula. The result of applying the INF propagator $\mathscr{I}^{\varphi}$ associated to φ on a three-valued structure $\tilde{I}$ is defined as in Definition 4.2. If $\tilde{I}$ is strictly four-valued, then we define $\mathscr{I}^{\varphi}(\tilde{I}) = \top^{\leq_p}$.

PROPOSITION A.3. *For every INF sentence φ, $\mathscr{I}^{\varphi}$ is a monotone propagator with polynomial-time data complexity.*

The proof of this proposition is analogous to the proof of Proposition 4.4.

We now show that every FO(AGG) theory T over vocabulary Σ can be converted to a Σ-equivalent theory containing only INF sentences. Similarly as in the case of FO theories, we present a conversion in several steps. None of the steps preserves complete propagation. The following example indicates that even for very simple theories, complete polynomial-time propagation is impossible (unless $\mathbf{P} = \mathbf{NP}$).

Example A.4. Let T be the theory consisting of the sentence $\text{SUM}\{x \mid P(x)\} = n$, where n is a natural number. Let $\tilde{I}$ be the structure $\bot_D^{\leq_P}$ with D a finite set of numbers and $P^{\tilde{I}}(d) = \mathbf{u}$ for every $d \in D$. Then $\mathscr{O}^T(\tilde{I}) \neq \top^{\leq_P}$ iff $\sum_{d \in \mathtt{V}} d = n$ for some subset $\mathtt{V} \subseteq D$. Deciding whether such a subset exists is **NP**-complete in this class [Sipser 2005]. Hence if $\mathbf{P} \neq \mathbf{NP}$, $\mathscr{O}^T$ cannot be implemented by a polynomial-time algorithm.

Definition A.5. An FO(AGG) sentence φ is in *equivalence normal form* (ENF) if φ is an FO sentence in ENF or φ is of the form $\forall \overline{x} \forall z\ (H[\overline{x}, z] \Leftrightarrow z \geq \mathtt{F}\{\overline{y} \mid L[\overline{x}, \overline{y}]\})$ or $\forall \overline{x} \forall z\ (H[\overline{x}, z] \Leftrightarrow z \leq \mathtt{F}\{\overline{y} \mid L[\overline{x}, \overline{y}]\})$, with H, L literals.

Definition 4.9 can be easily extended to define for every FO(AGG) theory T over Σ a Σ-equivalent theory in ENF. The intuition is still to represent nodes in the parse tree by new Tseitin predicates, and to express the logical relationship between such nodes and its children through an ENF formula.

Similarly as for FO sentences in ENF, a set INF(φ) of INF sentences is associated to each ENF sentence φ containing an aggregate. Our definition of this set is inspired by the propagation algorithms for propositional aggregates in the model generator MINISAT(ID) [Mariën 2009]. These algorithms aim to restore bounds consistency. Intuitively, the propagators associated to the INF sentences we present, express that if some formula $y \geq \mathtt{F}\{\overline{x} \mid L[\overline{x}]\}$ must be true and the assumption that $L[\overline{d}]$ is true (respectively false) would imply that y is strictly smaller than $\mathtt{F}(\{\overline{x} \mid L[\overline{x}]\})$, then $L[\overline{d}]$ must be false (respectively true). Similarly for formulas of the form $y \leq \mathtt{F}(\{\overline{x} \mid L[\overline{x}]\})$.

The following definition extends the definition of INF(φ) to the case where φ is an ENF sentences containing an aggregate expression.

Definition A.6. Let φ be an ENF sentence of the form $\forall \overline{x} \forall z\ (H[\overline{x}, z] \Leftrightarrow z \geq \mathtt{F}(\{\overline{y} \mid L[\overline{x}, \overline{y}]\}))$. Then INF($\varphi$) is the set of INF sentences

$$\forall \overline{x} \forall z\ (z \geq \mathtt{F}(\{\overline{y} \mid L[\overline{x}, \overline{y}]\}) \Rightarrow H[\overline{x}, z]), \tag{26}$$

$$\forall \overline{x} \forall z\ (z < \mathtt{F}(\{\overline{y} \mid L[\overline{x}, \overline{y}]\}) \Rightarrow \neg H[\overline{x}, z]), \tag{27}$$

$$\forall \overline{x} \forall z \forall \overline{y}'\ (H[\overline{x}, z] \wedge z < \mathtt{F}(\{\overline{y} \mid \overline{y} \neq \overline{y}' \wedge L[\overline{x}, \overline{y}]\}) \Rightarrow L[\overline{x}, \overline{y}']), \tag{28}$$

$$\forall \overline{x} \forall z \forall \overline{y}'\ (H[\overline{x}, z] \wedge z < \mathtt{F}(\{\overline{y} \mid \overline{y} = \overline{y}' \vee L[\overline{x}, \overline{y}]\}) \Rightarrow \neg L[\overline{x}, \overline{y}']), \tag{29}$$

$$\forall \overline{x} \forall z \forall \overline{y}'\ (\neg H[\overline{x}, z] \wedge z \geq \mathtt{F}(\{\overline{y} \mid \overline{y} \neq \overline{y}' \wedge L[\overline{x}, \overline{y}]\}) \Rightarrow L[\overline{x}, \overline{y}']), \tag{30}$$

$$\forall \overline{x} \forall z \forall \overline{y}'\ (\neg H[\overline{x}, z] \wedge z \geq \mathtt{F}(\{\overline{y} \mid \overline{y} = \overline{y}' \vee L[\overline{x}, \overline{y}]\}) \Rightarrow \neg L[\overline{x}, \overline{y}']). \tag{31}$$

For an ENF sentence φ of the form $\forall \overline{x} \forall z\ (H[\overline{x}, z] \Leftrightarrow z \leq \mathtt{F}(V))$, INF($\varphi$) is defined similarly (it suffices to replace '$<$' by '$>$' and '$\geq$' by '$\leq$' in sentences (26)–(31)).

The INF sentences 28 and 30 evaluate the aggregate on the set of tuples selected by the set expression but $\overline{y}'$ and express conditions for which the original ENF sentence φ cannot be true unless $\overline{y}'$ is selected by the set expression. Similarly, the INF sentences 29 and 31 evaluate the aggregate on the set of the tuples selected by the set expression extended with $\overline{y}'$ and expresses conditions for which the original ENF sentence φ cannot be true unless $\overline{y}'$ is not selected by the set expression.

Each of the sentences in INF(φ) is implied by φ. Vice versa, INF(φ) clearly implies φ for every ENF sentence φ. Hence, we obtain the following proposition.

PROPOSITION A.7. INF(φ) *is equivalent to* φ *for every ENF sentence* φ.

PROOF. We prove the case where φ is of the form $\forall\overline{x}\forall z\ (H(\overline{x},z) \Leftrightarrow z \geq \mathtt{F}(\{\overline{y} \mid L[\overline{x},\overline{y}]\}))$. Clearly, φ is equivalent to the conjunction of (26) and (27). Hence we only have to show that (28)–(31) are implied by φ.

We show that (28) is implied by φ. The proofs that (29)–(31) are implied by φ are similar. Let I be a structure and θ a variable assignment such that $I \models \varphi$ and $I\theta \models H(\overline{x},z) \wedge z < \mathtt{F}(\overline{y} \mid \overline{y} \neq \overline{y}' \wedge L[\overline{x},\overline{y}])$. Since $I \models \varphi$ and $I\theta \models H(\overline{x},z)$, $I\theta \models z \geq \mathtt{F}(\{\overline{y} \mid L[\overline{x},\overline{y}]\})$. Because $I\theta \models z < \mathtt{F}(\overline{y} \mid \overline{y} \neq \overline{y}' \wedge L[\overline{x},\overline{y}])$, it follows that $I\theta(\{\overline{y} \mid L[\overline{x},\overline{y}]\}) \neq I\theta(\{\overline{y}) \mid \overline{y} \neq \overline{y}' \wedge L[\overline{x},\overline{y}])\}$. Hence $I\theta \models L[\overline{x}/\overline{y}']$. We conclude that $I \models$ (28).

The other case where φ is of the form $\forall\overline{x}\forall z\ (H(\overline{x},z) \Leftrightarrow z \leq \mathtt{F}(\{\overline{y} \mid L[\overline{x},\overline{y}]\}))$, is analogous. □

A.3. Rule Sets and Symbolic Propagation

To represent propagation for FO as a rule set and to define symbolic propagation for FO theories, we relied on the fact that the value of an FO formula φ in a three-valued structure can be found by computing the value of the negation-free formulas φ_{ct} and φ_{cf}. Under certain conditions, it is possible to extend the definition of φ_{ct} and φ_{cf} to FO(AGG) formulas φ. This immediately lifts the results of Section 4.2 and Section 5 to FO(AGG).

It is beyond the scope of this paper to properly state the definition of φ_{ct} and φ_{cf} for an FO(AGG) formula φ, and the conditions under which this definition is the *correct* one. We refer the reader to [Wittocx 2010, pages 90–91 and 181–184] for these results. Relatively simple formulas φ_{ct} and φ_{cf} are obtained under the extra condition that all numbers that occur in structures are strictly positive and larger than 1. If arbitrary real numbers are allowed, the formulas φ_{ct} and φ_{cf} become so complicated that they seem not useful in practice.

ACKNOWLEDGMENTS

We are grateful for the suggestions of our anonymous reviewers.

REFERENCES

AAVANI, A., WU, X. N., TASHARROFI, S., TERNOVSKA, E., AND MITCHELL, D. G. 2012. Enfragmo: A system for modelling and solving search problems with logic. In *LPAR*, N. Bjørner and A. Voronkov, Eds. Lecture Notes in Computer Science Series, vol. 7180. Springer, 15–22.

ABITEBOUL, S. AND VIANU, V. 1991. Datalog extensions for database queries and updates. *J. Comput. Syst. Sci. 43,* 1, 62–124.

APT, K. R. 1999a. The essence of constraint propagation. *Theoretical Computer Science 221,* 1-2, 179–210.

APT, K. R. 1999b. Some remarks on Boolean constraint propagation. In *New Trends in Constraints*, K. R. Apt, A. C. Kakas, E. Monfroy, and F. Rossi, Eds. Lecture Notes in Computer Science Series, vol. 1865. Springer, 91–107.

APT, K. R. 2003. *Principles of Constraint Programming*. Cambridge University Press.

BARAL, C., BREWKA, G., AND SCHLIPF, J. S., Eds. 2007. *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*. LNCS Series, vol. 4483. Springer.

BELNAP, N. D. 1977. A useful four-valued logic. In *Modern Uses of Multiple-Valued Logic*, J. M. Dunn and G. Epstein, Eds. Reidel, Dordrecht, 8–37. Invited papers from the Fifth International Symposium on Multiple-Valued Logic, held at Indiana University, Bloomington, Indiana, May 13-16, 1975.

BREWKA, G. AND LANG, J., Eds. 2008. *Principles of Knowledge Representation and Reasoning: Proceedings of the 11th International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*. AAAI Press.

BRUYNOOGHE, M. 1991. A practical framework for the abstract interpretation of logic programs. *Journal of Logic Programming 10,* 1/2/3&4, 91–124.

CALIMERI, F., IANNI, G., RICCA, F., ALVIANO, M., BRIA, A., CATALANO, G., COZZA, S., FABER, W., FEBBRARO, O., LEONE, N., MANNA, M., MARTELLO, A., PANETTA, C., PERRI, S., REALE, K., SANTORO,

M. C., SIRIANNI, M., TERRACINA, G., AND VELTRI, P. 2011. The third answer set programming system competition: Preliminary report of the system competition track. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. Springer, 388–403.

CALUS, N. AND VENNEKENS, J. 2011. A Java API for a knowledge base system. In *"Proceedings of the 23rd Benelux Conference on Artificial +Intelligence*. 453–454.

CORTÉS CALABUIG, Á., DENECKER, M., ARIELI, O., AND BRUYNOOGHE, M. 2006. Representation of partial knowledge and query answering in locally complete databases. In *LPAR*, M. Hermann and A. Voronkov, Eds. Lecture Notes in Computer Science Series, vol. 4246. Springer, 407–421.

CORTÉS-CALABUIG, A., DENECKER, M., ARIELI, O., AND BRUYNOOGHE, M. 2007. Approximate query answering in locally closed databases. In *AAAI*, R. C. Holte and A. Howe, Eds. AAAI Press, 397–402.

CORTÉS-CALABUIG, Á., DENECKER, M., ARIELI, O., AND BRUYNOOGHE, M. 2008. Accuracy and efficiency of fixpoint methods for approximate query answering in locally complete databases. See Brewka and Lang [2008], 81–91.

DENECKER, M. 1998. The well-founded semantics is the principle of inductive definition. In *JELIA*, J. Dix, L. F. del Cerro, and U. Furbach, Eds. LNCS Series, vol. 1489. Springer, 1–16.

DENECKER, M. 2000. Extending classical logic with inductive definitions. In *CL*, J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, Eds. LNCS Series, vol. 1861. Springer, 703–717.

DENECKER, M., BRUYNOOGHE, M., AND MAREK, V. W. 2001. Logic programming revisited: Logic programs as inductive definitions. *ACM Transactions on Computational Logic (TOCL) 2,* 4, 623–654.

DENECKER, M., CORTÉS CALABUIG, Á., BRUYNOOGHE, M., AND ARIELI, O. 2010. Towards a logical reconstruction of a theory for locally closed databases. *ACM Transactions on Database Systems 35,* 3, 22:1–22:60.

DENECKER, M. AND TERNOVSKA, E. 2007. Inductive situation calculus. *Artificial Intelligence 171,* 5-6, 332–360.

DENECKER, M. AND TERNOVSKA, E. 2008. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic (TOCL) 9,* 2, Article 14.

DENECKER, M. AND VENNEKENS, J. 2007. Well-founded semantics and the algebraic theory of nonmonotone inductive definitions. See Baral et al. [2007], 84–96.

DENECKER, M. AND VENNEKENS, J. 2008. Building a knowledge base system for an integration of logic programming and classical logic. In *ICLP*, M. García de la Banda and E. Pontelli, Eds. LNCS Series, vol. 5366. Springer, 71–76.

DIETERICH, T. AND SWARTOUT, W., Eds. 1990. *8th National Conference on Artificial Intelligence, Boston, Massachusetts, July 29 - August 3, 1990, Proceedings*. AAAI/MIT Press.

EAST, D. AND TRUSZCZYNSKI, M. 2001. Propositional satisfiability in answer-set programming. In *KI/ÖGAI*, F. Baader, G. Brewka, and T. Eiter, Eds. Lecture Notes in Computer Science Series, vol. 2174. Springer, 138–153.

EBBINGHAUS, H.-D., FLUM, J., AND THOMAS, W. 1984. *Mathematical logic*. Undergraduate texts in mathematics. Springer.

ENDERTON, H. B. 2001. *A Mathematical Introduction To Logic* Second Ed. Academic Press.

FABER, W., PFEIFER, G., AND LEONE, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell. 175,* 1, 278–298.

FABER, W., PFEIFER, G., LEONE, N., DELL'ARMI, T., AND IELPA, G. 2008. Design and implementation of aggregate functions in the DLV system. *Theory and Practice of Logic Programming (TPLP) 8,* 5-6, 545–580.

FAGIN, R. 1974. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of Computation 7*, 43–74.

FAUSTINO DA SILVA, A. AND SANTOS COSTA, V. 2006. The design of the YAP compiler: An optimizing compiler for logic programming languages. *Journal of Universal Computer Science 12,* 7, 764–787.

FEFERMAN, S. 1984. Toward useful type-free theories. *Journal of Symbolic Logic 49,* 1, 75–111.

FORGY, C. 1979. On the efficient implementation of production systems. Ph.D. thesis, Carnegie Mellon University.

FORGY, C. 1982. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence 19,* 1, 17–37.

FRISCH, A. M., HARVEY, W., JEFFERSON, C., HERNÁNDEZ, B. M., AND MIGUEL, I. 2008. ESSENCE: a constraint language for specifying combinatorial problems. *Constraints 13,* 3, 268–306.

GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2007. *clasp*: A conflict-driven answer set solver. See Baral et al. [2007], 260–265.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9,* 3/4, 365–386.

GOUBAULT, J. 1995. A BDD-based simplification and skolemization procedure. *Logic Journal of IGPL 3,* 6, 827–855.

GRÄDEL, E., KOLAITIS, P. G., LIBKIN, L., MARX, M., SPENCER, J., VARDI, M. Y., VENEMA, Y., AND WEISTEIN, S. 2007. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. Springer.

GRAHNE, G. AND MENDELZON, A. O. 1999. Tableau techniques for querying information sources through global schemas. In *ICDT*, C. Beeri and P. Buneman, Eds. Lecture Notes in Computer Science Series, vol. 1540. Springer, 332–347.

HILL, P. M. AND WARREN, D. S., Eds. 2009. *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*. LNCS Series, vol. 5649. Springer.

HOU, P., DE CAT, B., AND DENECKER, M. 2010. FO(FD): Extending classical logic with rule-based fixpoint definitions. *Theory and Practice of Logic Programming 10,* 4-6, 581–596.

JACKSON, D. 2002. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM'02) 11,* 2, 256–290.

JANHUNEN, T. 2004. Representing normal programs with clauses. In *ECAI*, R. L. de Mántaras and L. Saitta, Eds. IOS Press, 358–362.

KLEENE, S. C. 1952. *Introduction to Metamathematics*. Van Nostrand.

LEONE, N., PFEIFER, G., FABER, W., EITER, T., GOTTLOB, G., PERRI, S., AND SCARCELLO, F. 2002. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7*, 499–562.

LEUSCHEL, M. 1997. Advanced techniques for logic program specialisation. *AI Communications 10,* 2, 127–128.

LIBKIN, L. 2004. *Elements of Finite Model Theory*. Springer.

MAREK, V. W. AND TRUSZCZYŃSKI, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, K. R. Apt, V. W. Marek, M. Truszczyński, and D. S. Warren, Eds. Springer-Verlag, 375–398.

MARIËN, M. 2009. Model generation for ID-logic. Ph.D. thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium.

MARIËN, M., MITRA, R., DENECKER, M., AND BRUYNOOGHE, M. 2005. Satisfiability checking for PC(ID). In *LPAR*, G. Sutcliffe and A. Voronkov, Eds. LNCS Series, vol. 3835. Springer, 565–579.

MARRIOTT, K., NETHERCOTE, N., RAFEH, R., STUCKEY, P. J., DE LA BANDA, M. G., AND WALLACE, M. 2008. The design of the Zinc modelling language. *Constraints 13,* 3, 229–267.

MCALLESTER, D. A. 1990. Truth maintenance. See Dieterich and Swartout [1990], 1109–1116.

MIRANKER, D. P., BRANT, D. A., LOFASO, B. J., AND GADBOIS, D. 1990. On the performance of lazy matching in production systems. See Dieterich and Swartout [1990], 685–692.

MITCHELL, D. G. AND TERNOVSKA, E. 2005. A framework for representing and solving NP search problems. In *AAAI*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press / The MIT Press, 430–435.

MITCHELL, D. G. AND TERNOVSKA, E. 2008. Expressive power and abstraction in ESSENCE. *Constraints 13,* 3, 343–384.

MORALES, A. R., TU, P. H., AND SON, T. C. 2007. An extension to conformant planning using logic programming. In *IJCAI*, M. M. Veloso, Ed. 1991–1996.

NIEMELÄ, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3-4, 241–273.

NIEMELÄ, I., SIMONS, P., AND SYRJÄNEN, T. 2000. Smodels: A system for answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*. Breckenridge, Colorado, USA. CoRR, cs.AI/0003033.

NIEUWENHUIS, R., OLIVERAS, A., AND TINELLI, C. 2006. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM 53,* 6, 937–977.

PELOV, N., DENECKER, M., AND BRUYNOOGHE, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming (TPLP) 7,* 3, 301–353.

PELOV, N. AND TERNOVSKA, E. 2005. Reducing inductive definitions to propositional satisfiability. In *ICLP*, M. Gabbrielli and G. Gupta, Eds. LNCS Series, vol. 3668. Springer, 221–234.

PERRI, S., SCARCELLO, F., CATALANO, G., AND LEONE, N. 2007. Enhancing DLV instantiator by backjumping techniques. *Annals of Mathematics and Artificial Intelligence 51,* 2-4, 195–228.

PETTOROSSI, A. AND PROIETTI, M. 1998. Transformation of logic programs. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Vol. 5. Oxford University Press, 697–787.

SIMONS, P., NIEMELÄ, I., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1-2, 181–234.

SIPSER, M. 2005. *Introduction to the Theory of Computation* 2nd Ed. Course Technology.

STICKEL, M. E. 1986. A Prolog technology theorem prover: Implementation by an extended Prolog compiler. In *CADE*, J. H. Siekmann, Ed. Lecture Notes in Computer Science Series, vol. 230. Springer, 573–587.

SWIFT, T. 2009. An engine for computing well-founded models. See Hill and Warren [2009], 514–518.

TERNOVSKA, E. AND MITCHELL, D. G. 2009. Declarative programming of search problems with built-in arithmetic. In *IJCAI*, C. Boutilier, Ed. 942–947.

TORLAK, E. AND JACKSON, D. 2007. Kodkod: A relational model finder. In *TACAS*, O. Grumberg and M. Huth, Eds. LNCS Series, vol. 4424. Springer, 632–647.

TSEITIN, G. S. 1968. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic II*, A. O. Slisenko, Ed. Consultants Bureau, N.Y., 115–125.

ULLMAN, J. D. 1988. *Principles of database and knowledge-base systems, Vol. I*. Computer Science Press, Inc., New York, NY, USA.

VAEZIPOOR, P., MITCHELL, D., AND MARIËN, M. 2011. Lifted unit propagation for effective grounding. *CoRR abs/1109.1317*.

VAN GELDER, A., ROSS, K. A., AND SCHLIPF, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.

VAN HENTENRYCK, P. 1999. *The OPL Optimization Programming Language*. MIT Press.

VAN WEERT, P. 2010. Efficient lazy evaluation of rule-based programs. *IEEE Transactions on Knowledge and Data Engineering 22,* 11, 1521–1534.

VENNEKENS, J., MARIËN, M., WITTOCX, J., AND DENECKER, M. 2007. Predicate introduction for logics with a fixpoint semantics. Part I: Logic programming. *Fundamenta Informaticae 79,* 1-2, 187–208.

VLAEMINCK, H., VENNEKENS, J., AND DENECKER, M. 2009. A logical framework for configuration software. In *PPDP*, A. Porto and F. J. López-Fraguas, Eds. ACM, 141–148.

VLAEMINCK, H., VENNEKENS, J., DENECKER, M., AND BRUYNOOGHE, M. 2012. An approximative inference method for solving ∃∀SO satisfiability problems. *The Journal of Artificial Intelligence Research 45*, 79–124.

VLAEMINCK, H., WITTOCX, J., VENNEKENS, J., DENECKER, M., AND BRUYNOOGHE, M. 2010. An approximate method for solving ∃∀SO problems. In *JELIA*, M. Fisher, W. van der Hoek, B. Konev, and A. Lisitsa, Eds. Lecture Notes in Computer Science. Springer, 326–338.

WITTOCX, J. 2010. Finite domain and symbolic inference methods for extensions of first-order logic. Ph.D. thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium.

WITTOCX, J., MARIËN, M., AND DENECKER, M. 2008a. Approximate reasoning in first-order logic theories. See Brewka and Lang [2008], 103–112.

WITTOCX, J., MARIËN, M., AND DENECKER, M. 2008b. Grounding with bounds. In *AAAI*, D. Fox and C. P. Gomes, Eds. AAAI Press, 572–577.

WITTOCX, J., MARIËN, M., AND DENECKER, M. 2008c. The IDP system: a model expansion system for an extension of classical logic. In *LaSh*, M. Denecker, Ed. 153–165.

WITTOCX, J., MARIËN, M., AND DENECKER, M. 2010. Grounding FO and FO(ID) with bounds. *Journal of Artificial Intelligence Research 38*, 223–269.

WITTOCX, J., VLAEMINCK, H., AND DENECKER, M. 2009. Debugging for model expansion. See Hill and Warren [2009], 296–311.

Received August 2010; revised July 2012; accepted November 2012