

Reducing Inductive Definitions to Propositional Satisfiability

Nikolay Pelov and Eugenia Ternovska

School of Computing Science, Simon Fraser University
Vancouver, Canada

Abstract. The FO(ID) logic is an extension of classical first-order logic with a uniform representation of various forms of inductive definitions. The definitions are represented as sets of rules and they are interpreted by two-valued well-founded models. For a large class of combinatorial and search problems, knowledge representation in FO(ID) offers a viable alternative to the paradigm of Answer Set Programming. The main reasons are that (i) the logic is an extension of classical logic and (ii) the semantics of the language is based on well-understood principles of mathematical induction.

In this paper, we define a reduction from the propositional fragment of FO(ID) to SAT. The reduction is based on a novel characterization of two-valued well-founded models using a set of inequality constraints on level mappings associated with the atoms. We also show how the reduction to SAT can be adapted for logic programs under the stable model semantics. Our experiments show that when using a state of the art SAT solver both reductions are competitive with other answer set programming systems — both direct implementations and SAT based.

1 Introduction

Definitions, and inductive definitions in particular, are common in human reasoning [3]. Examples of inductive definitions include the definitions of the set of well-formed formulas and the satisfaction relation \models in logic. An example of an inductive definition from common-sense reasoning concerns reasoning about actions. There, a description of the initial situation represents the base case, and causal laws specifying direct and indirect effects of action, represent inductive cases [6, 7, 21, 22]. Inductive definitions can be monotone (e.g. the definition of a well-formed formula) or non-monotone (e.g. the definition of \models and many definitions in common-sense reasoning). Both monotone and non-monotone induction are formalized in a natural way in the logic FO(ID) which is an extension of first-order logic (FO) with (non-monotone) inductive definitions (ID) [4, 6]. The semantics of FO(ID) is a combination of the semantics of first-order logic and the well-founded semantics of logic programming. The usefulness of FO(ID) in knowledge representation has been demonstrated in several applications. The authors of [5] show that the situation calculus can be formalized as a (non-monotone) iterated inductive definition in the well-ordered set of situations. The

resulting formalism provides a very general solution to the ramification problem. Another natural application of FO(ID) is to data integration in database theory [24].

Recently, the FO(ID) logic has been used as the underlying logic of a declarative constraint programming framework [17]. The approach is formalized as a model expansion problem, which is based on the classical notion of expansion of a structure by new relations. A parameterized version of this problem captures precisely NP search problems.

The FO(ID) logic is similar [16] to the framework of Answer Set Programming (ASP) [1]. It can be argued that the representation of many problems in FO(ID) is more intuitive than the representation in logic programming under the answer set semantics. To a large extent this is due to the fact that FO(ID) is an extension of classical logic which is well-understood and widely used. Another reason is that the use of recursion is limited only to concepts where it is really needed and predicates which are defined recursively are interpreted as inductive definitions. On the other hand, recursion (very often over negation) is used much more frequently in ASP, for example for defining the set of possible solutions. So, for someone who is not familiar with the knowledge representation methodology of ASP, this is often confusing. Although recent extensions of the language, for example with choice rules [20], alleviate the problem to some extent, the basic critique remains.

Despite its attractive properties, no full-featured implementation of FO(ID) exists. Mariën, Gilis and Denecker [16] define a reduction from FO(ID) to ASP, however only total definitions are supported¹. In this paper, we discuss an implementation of the propositional fragment of FO(ID) called PC(ID), which stands for *propositional calculus with inductive definitions*. Our approach is to define a reduction from the satisfiability problem of PC(ID) to propositional satisfiability (SAT). Since part of a PC(ID) theory is already a set of propositional formulas, the main challenge is the translation of inductive definitions.

The use of SAT solvers has already been shown to be successful for implementing ASP systems [12, 13, 14]. The approach which is typically used is to compute the Clark's completion of the program and call a SAT solver to return a model. Then, the model is verified if it is stable and, if it is not, a special type of formulas, called loop formulas, are added to the theory to eliminate the model. This approach works quite well in practice, although for some problems there may be an exponential number of loop formulas. There are also several "off-line" reductions to SAT [2, 11, 13, 15] but their performance is not yet well studied.

An important property of the reductions to SAT, called *faithfulness*, refers to the case where there is a one-to-one correspondence between the models of the original theory and its reduction. The reductions of Lin and Zhao [13] and Janhunen [11] are faithful while those of Ben-Eliyahu and Dechter [2] and Linke, Tompits and Woltran [15] are not. Faithful reductions are useful when one is

¹ A definition D is total if it has a two-valued well-founded model for every interpretation of the open predicates of D .

interested to compute or to count all solutions of a problem. However, faithful reductions are normally larger than non-faithful ones. What influence this difference has on the speed of SAT solvers is still unknown. We are unaware of any comparison on the performance of faithful and non-faithful reductions and the experiments in this paper are a first step in this direction. The reduction of PC(ID) to SAT which we define is non-faithful. The main reason is that the difference between the size of a faithful and non-faithful reduction for the well-founded semantics is much greater than the difference between faithful and non-faithful reductions for the stable semantics. The main difficulty comes from the computation of greatest unfounded sets.

The reduction from PC(ID) to SAT is based on a novel characterization of two-valued well-founded models by a set of inequality constraints on level mappings associated with the atoms in the theory. The level mapping of an atom a is related to the step of the well-founded operator W_P at which the truth value of a is derived. The reduction is very similar to the one by Ben-Eliyahu and Dechter [2] for head-cycle free disjunctive logic programs under the stable model semantics. Besides the difference in the language and semantics, the other main difference between the two reductions is that we use a binary encoding of level mappings as in [11] while [2] uses a unary encoding.

The paper is organized as follows. We start by recalling in Section 2 some preliminaries from logic programming. In Section 3 we recall the syntax and semantics of the logics FO(ID) and PC(ID). In Section 4 we develop the theoretical foundation of the reduction by characterizing well-founded models by certain types of level mappings. The reduction itself is defined in Section 5 and in Section 6 we evaluate its performance and compare it with other ASP systems.

2 Preliminaries

Let At be a set of atoms. A *literal* is an atom $a \in At$ or its negation $\neg a$. A *rule* is an expression of the form

$$a \leftarrow l_1 \wedge \dots \wedge l_n \tag{1}$$

where $a \in At$ and l_i are literals over At . For a rule r of the form (1) we denote $hd(r) = a$ and $body(r) = \{l_1, \dots, l_n\}$. The set of positive literals in the body of r is denoted with $pos(r)$ and the set of negative literals with $neg(r)$. A *logic program* is a finite set of rules. A *definite rule* is a rule without negative literals in the body and a *definite program* is a program consisting only of definite rules.

With $th(P)$ we denote the set of formulas obtained by replacing “ \leftarrow ” with “ \supset ” in every rule in P . The *only-if* part for a logic program P , denoted with $only_if(P)$, is defined as the set of formulas $a \supset B_1 \vee \dots \vee B_n$ for every atom $a \in At$ where $a \leftarrow B_1, \dots, a \leftarrow B_n$ are all the rules with a in the head. If the body B_i of some rule for a is empty then B_i is understood as the constant \mathbf{t} . Finally, the *completion* of a program P is defined as $comp(P) = th(P) \cup only_if(P)$.

The *dependency graph* G_P of a logic program P is a signed directed graph defined as follows. The atoms of At form the vertices of G_P . For every rule $r \in P$

there is an edge from $hd(r)$ to b for every positive literal $b \in pos(r)$ and there is an edge from $hd(r)$ to b labeled with “-” for every negative literal $\neg b \in neg(r)$. We say that an atom a *depends negatively on itself* if G_P contains a (directed) cycle containing a and an edge labeled with “-”. A *strongly connected component (SCC)* is a maximal set of atoms A such that there is a directed path in G_P for every pair of atoms in A . The set of all strongly connected components of G_P is denoted with $SCC(P)$ and the SCC to which an atom a belongs with $SCC(a)$. For a strongly connected component $S \in SCC(P)$ we denote with P_S the sub-program of P which is restricted only to rules with atoms in S in the head.

An *interpretation* I is a function $I: At \rightarrow \{\mathbf{f}, \mathbf{t}\}$. Frequently, we identify an interpretation with the set of atoms which are assigned the value \mathbf{t} . The *complement* \bar{I} of an interpretation I is taken with respect to At , i.e., $\bar{I} = At \setminus I$. A *three-valued interpretation* is a consistent set of literals \tilde{I} . We denote the subset of positive literals of \tilde{I} with \tilde{I}^+ and the subset of negative literals with \tilde{I}^- . For a set of literals L , we denote with $\neg L$ the set which consists of all literals from L with reversed polarity.

The *program reduct* P^I [10] of a program P with respect to an interpretation I is a program obtained from P by:

- deleting all rules which have a negative literal not satisfied by I ;
- deleting all negative literals from the remaining rules.

The program reduct P^I is a definite logic program and it has a unique least model, denoted with $lm(P^I)$. An interpretation I is a *stable model* of P if $I = lm(P^I)$.

Next, we recall the definition of the well-founded semantics [23].

Definition 1 (Unfounded Set). *Let P be a logic program and let \tilde{I} be a three-valued interpretation. We say that a set of atoms $A \subseteq At$ is an unfounded set (of P) with respect to \tilde{I} if each atom $a \in A$ satisfies the following condition. For each rule $r \in P$ such that $hd(r) = a$ one of the following holds:*

1. $\neg l_i \in \tilde{I}$ for some literal $l_i \in body(r)$;
2. some positive literal in $body(r)$ occurs in A , i.e., $pos(r) \cap A \neq \emptyset$.

Taking the union of any collection of unfounded sets is also an unfounded set. So, for any given three-valued interpretation \tilde{I} , a program P has a *greatest unfounded set* with respect to \tilde{I} , denoted with $U_P(\tilde{I})$. Next we define the following three-valued operators:

$$\begin{aligned} \mathcal{T}_P(\tilde{I}) &= \{hd(r) : r \in P \text{ and } body(r) \subseteq \tilde{I}\} \\ W_P(\tilde{I}) &= \mathcal{T}_P(\tilde{I}) \cup \neg U_P(\tilde{I}) \end{aligned}$$

The W_P operator is monotone in the sense that if $\tilde{I}_1 \subseteq \tilde{I}_2$ then $W_P(\tilde{I}_1) \subseteq W_P(\tilde{I}_2)$. By a well-known result of Tarski, follows that W_P has a least fixpoint denoted with $lfp(W_P)$. The *well-founded model* of a logic program P is defined

as $\text{lfp}(W_P)$ and denoted with $WF(P)$. The least fixpoint of the W_P operator can also be computed constructively by the following iteration of the W_P operator:

$$\begin{aligned} W_P^0 &= \emptyset \\ W_P^{i+1} &= W_P(W_P^i) \end{aligned} \quad \text{for } i \in \mathbb{N}$$

Proposition 1. *There exists a natural number n that $W_P^n = W_P^{n+1} = \text{lfp}(W_P)$.*

The least natural number n which satisfies the conditions of the above proposition, i.e., $W_P^n = \text{lfp}(W_P)$, is called the *closure number* of W_P . The *stage* of a literal $l \in WF(P)$, denoted with $|l|_{W_P}$, is defined as the least number $i \in \mathbb{N}$ such that $l \in W_P^i$.

3 Propositional Calculus with Inductive Definitions

An FO(ID) theory is a pair $\langle \mathcal{D}, \mathcal{A} \rangle$ where \mathcal{D} is a set of definitions and \mathcal{A} is a set of FO sentences². A definition $D \in \mathcal{D}$ is a set of rules of the form $\forall \bar{x}(p(\bar{t}) \leftarrow \varphi)$ where $p(\bar{t})$ is an atom and φ is a FO formula. The set of predicates which appear in the heads of the rules in a definition D are *defined* by D and denoted with $\text{def}(D)$. All other predicate symbols in D are called *open* and their set is denoted with $\text{open}(D)$.

The propositional fragment of FO(ID) is denoted with PC(ID) and stands for *propositional calculus with inductive definitions*. For simplicity, we assume that a definition is a propositional logic program, i.e., the body of every rule is a conjunction of literals and every sentence in \mathcal{A} is a propositional clause.

The semantics of FO(ID) and PC(ID) is defined as a combination of classical first-order semantics for the set of FO sentences and two-valued well-founded semantics for the definitions.

Definition 2. *An interpretation I is a model of a PC(ID) theory $\langle \mathcal{D}, \mathcal{A} \rangle$ if:*

1. *for every definition $D \in \mathcal{D}$, I is the (two-valued) well-founded model of $D \cup (I \cap \text{open}(D))$ where $I \cap \text{open}(D)$ is the set of open atoms in D which are true in I ;*
2. $I \models \mathcal{A}$.

We illustrate the syntax of the logic with a formulation of the Hamiltonian Cycle problem [16].

Example 1 (Hamiltonian Cycle). The problem of finding a Hamiltonian cycle in a directed graph is encoded by the following FO(ID) theory $\langle \{D_1, D_2\}, \mathcal{A} \rangle$. The

² In this paper we use a restricted syntax of FO(ID) and its propositional fragment PC(ID) which is more suitable for programming and implementation.

first definition D_1 encodes the input graph and a designated initial node as a set of facts:

$$D_1 = \left\{ \begin{array}{l} vertex(v). \\ \dots \\ arc(u, v). \\ \dots \\ initialnode(v). \end{array} \right\}.$$

The Hamiltonian cycle is described by a binary predicate $hc(x, y)$. The second definition defines, by a positive induction, the the set of nodes reachable through the $hc(x, y)$ relation starting from a designated initial node:

$$D_2 = \left\{ \begin{array}{l} \forall xy(reached(y) \leftarrow initialnode(x) \wedge arc(x, y) \wedge hc(x, y)) \\ \forall xy(reached(y) \leftarrow reached(x) \wedge arc(x, y) \wedge hc(x, y) \wedge \\ \quad \neg initialnode(x)) \end{array} \right\}.$$

Finally, the formulas \mathcal{A} are the following axioms:

$$\begin{aligned} & \forall x(vertex(x) \supset reached(x)), \\ & \forall xyz(arc(x, y) \wedge arc(x, z) \wedge hc(x, y) \wedge hc(x, z) \supset y = z), \\ & \forall xyz(arc(x, y) \wedge arc(z, y) \wedge hc(x, y) \wedge hc(z, y) \supset x = z), \\ & \forall xy(hc(x, y) \supset arc(x, y)). \end{aligned}$$

□

A PC(ID) theory is obtained from the above FO(ID) formulation by a standard process of grounding which eliminates quantifiers by substituting all possible values in their domain.

4 Weak Level Mappings

The reduction of inductive definitions under the well-founded semantics to propositional satisfiability is based on modeling of the computation of the well-founded operator W_P . If we encode precisely the stages induced by the operator W_P (the function $|\cdot|_{W_P}$), we can obtain a faithful reduction. However, such reduction will be very expensive in terms of its size. Instead, we use a function $|\cdot|_{wk}: WF(P) \rightarrow \mathbb{N}$ called a weak level mapping which only captures the ordering of literals induced by $|\cdot|_{W_P}$: if $|l_1|_{W_P} < |l_2|_{W_P}$ then $|\cdot|_{wk}$ should satisfy $|l_1|_{wk} < |l_2|_{wk}$. Since there will be many such functions, the reductions which we obtain is non-faithful.

Definition 3. *Let P be a logic program and L a consistent set of literals. A weak level mapping is any function $|\cdot|_{wk}: L \rightarrow \mathbb{N}$ such that L and $|\cdot|_{wk}$ satisfy the following conditions.*

1. For every positive literal $a \in L$, there exists a rule $r \in P$ such that $hd(r) = a$, $body(r) \subseteq L$ and for every literal $l_i \in body(r)$, $|a|_{wk} > |l_i|_{wk}$.
2. For every negative literal $\neg a \in L$ and for every rule $r \in P$ with $hd(r) = a$, there exists a literal $l_i \in body(r)$ such that $\neg l_i \in L$, and if a depends negatively on itself then:
 - (a) $|\neg a|_{wk} \geq |\neg l_i|_{wk}$ if l_i is positive;
 - (b) $|\neg a|_{wk} > |\neg l_i|_{wk}$ if l_i is negative.

Example 2. Consider the following logic program:

$$\begin{aligned} a &\leftarrow a \wedge c \\ b &\leftarrow a \\ c &\leftarrow \neg b. \end{aligned}$$

Its well-founded model is $\{-a, \neg b, c\}$ and is computed as:

$$\begin{aligned} W_P^1 &= W_P(\emptyset) = \{-a, \neg b\} \\ W_P^2 &= W_P(W_P^1) = \{-a, \neg b, c\} \end{aligned}$$

so the stages of the literals in the well-founded model are

$$|\neg a|_{W_P} = 1, \quad |\neg b|_{W_P} = 1, \quad |c|_{W_P} = 2.$$

One possible weak level mapping is the following:

$$|\neg a|_{wk} = 2, \quad |\neg b|_{wk} = 3, \quad |c|_{wk} = 5.$$

The above example illustrates several differences between a weak level mapping $|\cdot|_{wk}$ and the level mapping $|\cdot|_{W_P}$:

- the smallest value of $|\cdot|_{wk}$ may be higher than 1;
- $|\cdot|_{wk}$ may have gaps between levels;
- literals that have the same value of $|\cdot|_{W_P}$ may have different values of $|\cdot|_{wk}$.

The first result is soundness of the weak level mappings with respect to the well-founded semantics.

Proposition 2. *Let P be a logic program and L a consistent set of literals. If there exists a weak level mapping with domain L then $L \subseteq WF(P)$.*

For models of inductive definitions, we are interested only in two-valued well-founded models, so we have the following corollary.

Corollary 1. *Let P be a logic program and L a consistent set of literals which is two-valued, i.e., for every $a \in At$ either $a \in L$ or $\neg a \in L$. If there exists a weak level mapping with domain L then L is the well-founded model of P .*

Note that under the conditions of the corollary, the set L is also the unique stable model of P .

The next proposition is a completeness result, meaning that there exists a weak level mapping whose domain is the well-founded model of a program.

Proposition 3. *Let P be a logic program. The level mapping $|\cdot|_{W_P} : WF(P) \rightarrow \mathbb{N}$ is a weak level mapping.*

5 Reduction

The reduction from PC(ID) to SAT is obtained by encoding, as a set of clauses, the conditions on weak level mappings from Definition 3.

The reduction T_{WEAK} of a PC(ID) theory $\langle \mathcal{D}, \mathcal{A} \rangle$ to a CNF formula is defined as

$$T_{\text{WEAK}}(\langle \mathcal{D}, \mathcal{A} \rangle) = \left(\bigcup_{D \in \mathcal{D}} T_{\text{WEAK}}(D) \right) \cup \mathcal{A}$$

where the reduction $T_{\text{WEAK}}(D)$ of a definition D is defined as

$$T_{\text{WEAK}}(D) = \bigcup_{a \in \text{def}(D)} \{\text{clauses (2), \dots, (9)}\}.$$

Clauses (2), ..., (9) are defined as follows. Let a be an atom and $r_1: a \leftarrow B_1, \dots, r_n: a \leftarrow B_n$ be all the rules in D with a in the head. For every rule r_i we introduce a new propositional variable r_i and the clause:

$$a \supset r_1 \vee \dots \vee r_n. \quad (2)$$

For every rule $r_i: a \leftarrow B_i$ partition the literals in B_i in the set $\text{def}(r_i) = \text{body}(r_i) \cap \text{def}(D)$ of literals which are defined in D and the set $\text{open}(r_i) = \text{body}(r_i) \setminus \text{def}(r_i)$ of literals which are open in D . Then we add the clauses:

$$r_i \supset l_j \quad \text{for every } l_j \in \text{body}(r_i) \quad (3)$$

$$r_i \supset |a|_{wk} > |l_j|_{wk} \quad \text{for every } l_j \in \text{def}(r_i) \quad (4)$$

If a does not depend negatively on itself, we add the ‘‘if’’ part of r_i :

$$a \subset B_i. \quad (5)$$

Otherwise, if a depends negatively on itself, we introduce a new variable c_j for every literal $l_j \in \text{def}(r_i)$ and add the following clauses:

$$a \vee \bigvee_{l_j \in \text{def}(r_i)} c_j \vee \bigvee_{l_j \in \text{open}(r_i)} \neg l_j \quad (6)$$

$$c_j \supset \neg l_j \quad \text{for every } l_j \in \text{def}(r_i) \quad (7)$$

$$c_j \supset |a|_{wk} \geq |l_j|_{wk} \quad \text{for every pos. literal } l_j \in \text{def}(r_i) \quad (8)$$

$$c_j \supset |a|_{wk} > |l_j|_{wk} \quad \text{for every neg. literal } l_j \in \text{def}(r_i) \quad (9)$$

Clauses (2)–(4) encode the first condition of weak level mappings (justification of true atoms) while clauses (5)–(9) encode the second condition (justification of false atoms). In case when there is no recursion through negation, false atoms are justified by (the contra-positive of) clause (5). Notice that by resolving (6) with the clauses (7) for all $c_j \in \text{def}(r_i)$ we obtain the clause

$$a \vee \bigvee \{\neg l_j : l_j \in \text{body}(r_i)\}$$

which is equivalent to (5).

We now explain the encoding of the comparisons between level mappings used in the definition of the reduction. The weak level mapping $|\cdot|_{wk} : At \rightarrow [1, n]$ is encoded by a set of vectors $\vec{a} = a_k, \dots, a_1$ of propositional variables for each $a \in At$, representing the binary encoding of the value of $|a|_{wk}$. The length k of the vectors is $k = \lceil 1 + \log_2 n \rceil$. Since the well-founded operator W_P reaches a fixpoint in at most $|At|$ steps³ we can take $n = |At|$. For two vectors \vec{a} and \vec{b} the encoding of $|a|_{wk} < |b|_{wk}$ is given by the following logic program $LT(\vec{a}, \vec{b}) =$

$$\begin{aligned} lt(a, b)_i &\leftarrow \neg a_i \wedge b_i & i \in [1, k] \\ lt(a, b)_i &\leftarrow a_i \wedge b_i \wedge lt(a, b)_{i-1} & i \in [2, k] \\ lt(a, b)_i &\leftarrow \neg a_i \wedge \neg b_i \wedge lt(a, b)_{i-1} & i \in [2, k] \end{aligned}$$

For a variable assignment $v : \{a_1, \dots, a_k\} \rightarrow \{0, 1\}$ we denote with $v(\vec{a})$ the number whose binary representation is $v(a_k) \dots v(a_1)$.

Lemma 1. *Let $\vec{a} = a_k, \dots, a_1$ and $\vec{b} = a_k, \dots, a_1$ be two vectors of propositional variables and let v be a variable assignment for \vec{a} and \vec{b} . Then*

- $v \models lt(a, b)_k \wedge \text{only_if}(LT(\vec{a}, \vec{b}))$ implies $v(\vec{a}) < v(\vec{b})$;
- $v \models \neg lt(b, a)_k \wedge \text{th}(LT(\vec{b}, \vec{a}))$ implies $v(\vec{a}) \leq v(\vec{b})$.

So, in the clauses of the reduction $|a|_{wk} < |b|_{wk}$ stands for the variable $lt(a, b)_k$ and $\text{only_if}(LT(\vec{a}, \vec{b}))$ is added to the output of the translation. Similarly, $|a|_{wk} \leq |b|_{wk}$ stands for the variable $\neg lt(b, a)_k$ and $\text{th}(LT(\vec{b}, \vec{a}))$ is added to the output of the translation.

Theorem 1. *Let \mathcal{T} be a PC(ID) theory. Then: (i) for every model M of \mathcal{T} there exists a model M' of $T_{\text{WEAK}}(\mathcal{T})$ such that $M = M' \cap At$; and (ii) the restriction $M' \cap At$ of every model M' of $T_{\text{WEAK}}(\mathcal{T})$ is a model of \mathcal{T} .*

To reduce the size of the translation, we split every definition $D \in \mathcal{D}$ in a PC(ID) theory $\langle \mathcal{D}, \mathcal{A} \rangle$ to a set of definitions $\{D_S : S \in SCC(D)\}$ — one for each strongly connected component S of D . This is an equivalence preserving transformation as shown in [6].

Proposition 4. *Let $\mathcal{T} = \langle \mathcal{D}, \mathcal{A} \rangle$ be a PC(ID) theory. The number of literals in $T_{\text{WEAK}}(\mathcal{T})$ is $O(\text{size}(\mathcal{T}) \times \log m)$ where $\text{size}(\mathcal{T})$ is the total number of literals in \mathcal{T} and $m = \max\{|def(D)| : D \in \mathcal{D}\}$.*

5.1 Stable Semantics

The T_{WEAK} reduction can be adapted for logic programs under the stable semantics by assuming that the program does not contain a recursion over negation

³ A tighter bound on the number of steps is the length of the longest path in any strongly connected component [2], however this is an NP-complete problem.

(even if it actually does). The intuition is that, under the stable semantics, no justification is necessary for atoms which are false — they can be assumed “false by default”. So, clauses (6)–(9) are never used and clause (5) is used instead. In addition, atoms without rules should be set to false. Let T_{SM} denote this modified translation:

$$T_{\text{SM}}(P) = \bigcup_{a \in \text{def}(P)} \{\text{clauses (2), \dots, (5)}\} \cup \bigcup_{a \in \text{open}(P)} \{\neg a\}.$$

Theorem 2. *Let P be a logic program. Then: (i) for every stable model M of P there exists a model M' of $T_{\text{SM}}(P)$ such that $M = M' \cap \text{At}$; and (ii) the restriction $M' \cap \text{At}$ of every model M' of $T_{\text{SM}}(P)$ is a stable model of P .*

Similarly to the T_{WEAK} reduction, the T_{SM} reduction can be applied separately for each strongly connected component of the program. It is possible to further improve the reduction by exploiting a well-known theorem by Fages [9] stating that the stable models of a tight⁴ logic program P are equal to the models of $\text{comp}(P)$. The obvious application of this result is to change the reduction such that for a SCC which is tight to compute its completion. However, even for a non-tight component it is possible to avoid assigning level mapping to some atoms. This is done by refining the condition of tightness to individual atoms.

Definition 4. *Let P be a logic program. An atom a is tight if every cycle in the dependency graph of P which passes through a contains negation. A logic program P is tight if every atom in P is tight.*

The above definition of a tight logic program is equivalent to the one of [9] and the one of Erdem and Lifschitz [8] based on level mappings.

Example 3. Consider the following program:

$$\begin{aligned} r_1: p &\leftarrow \neg q \\ r_2: p &\leftarrow p \wedge r \\ r_3: q &\leftarrow \neg p. \end{aligned}$$

The atom q is tight, however the atom p is not because of the positive dependence of p on itself from the second rule. Consequently, the program is not tight. \square

The importance of tight atoms is that in the reduction it is not necessary to assign to them a level mapping. This optimization is easy to implement by changing the definition of the set $\text{def}(r)$ used in clause (4) as follows:

$$\text{def}(r) = \{a \in \text{pos}(r) : \text{SCC}(a) = \text{SCC}(\text{hd}(r)) \text{ and } a \text{ is not tight}\}.$$

⁴ Originally called *positive-order-consistent*.

Example 4. Applying the optimized reduction to the program from Example 3 we obtain the following theory:

$p \supset r_1 \vee r_2$	by rule (2)
$r_1 \supset \neg q$	by rule (3)
$p \subset \neg q$	by rule (5)
$r_2 \supset \neg p$	by rule (3)
$r_2 \supset p _{wk} > p _{wk}$	by rule (4)
$r_2 \supset r$	by rule (3)
$p \subset p \wedge r$	by rule (5)
$q \supset r_3$	by rule (2)
$r_3 \supset \neg p$	by rule (3)
$q \subset \neg p$	by rule (5)
$\neg r$	

We conclude the section by giving a result on the size of the reduction.

Proposition 5. *Let P be a logic program. The number of literals in $T_{SM}(P)$ is $O(\text{size}(P) \times \log|At|)$ where $\text{size}(P)$ is the total number of literals in P .*

6 Experiments

To test the different reductions we implemented a prototype system in perl called `IDSAT`⁵. As a front-end we used `LPARSE` for grounding, however only `PC(ID)` theories consisting of a single definition D can be encoded in the input language of `LPARSE` and the set of sentences must be written as a set of integrity constraints. To be able to support correctly open predicates we needed to make some changes to `LPARSE`. The problem is that under the stable semantics, predicates without definition are assumed to be false (while they are open for `PC(ID)`). Consequently, `LPARSE` will remove all rules which have such predicates in the body. After the change such rules were left in the output.

All experiments were performed on a 2GHz Intel Pentium 4 PC with 256MB memory running Linux with kernel 2.4.20. We report the number of seconds for finding first solution (or showing that no solution exists) averaged over 5 runs. Since all systems use `LPARSE` for grounding, `LPARSE` time is not included. A “-” means that no solution was found within 2 hours. We used `SMODELS` version 2.28 [20] and `ASSAT` version 2.02 [14]. The `LP2SAT` is an implementation of the faithful reduction of Janhunen [11]. For all SAT based systems we used the SAT solver *siege* variant 4 [19].

Table 6 reports the results for the Hamiltonian Cycle problem. For the T_{WEAK} reduction, we used the formulation in Example 1, and for all other systems

⁵ <http://nik.pelov.name/idsat/>

(including the T_{SM} reduction), the encoding by Niemelä [18]. The second column “HC” shows whether the given graph has a Hamiltonian cycle or not. ASSAT is the only system which is able to solve all problems. The three reductions are faster than ASSAT on smaller graphs (p20–p30) where ASSAT needs to call a SAT solver several times before finding the first solution. However, on larger graphs, ASSAT is the fastest SAT based system. The performance of the three off-line reductions (T_{WEAK} , T_{SM} , and LP2SAT), is close to that of ASSAT with LP2SAT being the fastest among them. It is interesting that all three reductions to SAT have problems with graphs without a Hamiltonian cycle.

Graph	HC	PC(ID)	ASP			
		T_{WEAK}	T_{SM}	LP2SAT	ASSAT	S MODELS
p20	y	0.84	0.46	0.19	5.03	0.07
p25	y	0.77	0.62	0.94	9.27	0.09
p29	y	0.97	1.15	0.20	9.59	1.19
p30	y	0.96	1.33	2.10	13.28	0.13
2xp30	n	-	-	-	0.21	0.18
2xp30.1	y	188.62	237.84	197.56	84.04	0.33
2xp30.2	y	168.53	212.76	112.86	112.17	-
2xp30.3	y	218.63	169.26	133.15	103.83	-
2xp30.4	n	-	-	-	98.91	-
4xp20	n	208.07	266.18	190.85	0.29	0.22
4xp20.1	n	266.30	314.07	183.66	2.45	-
4xp20.2	y	196.69	147.08	81.37	29.43	0.43
4xp20.3	n	-	-	202.87	5.16	0.22

Table 1. Times for the Hamiltonian Cycle problem: pN — a graph with N nodes⁶; KxpN.i — K copies of the the graph pN with some extra edges added, i stands for a variation of the graph⁷.

Table 6 compares the input sizes for the formulation of the problem in PC(ID) and in ASP (columns “input rules”) and the increase in size for all reductions to SAT (columns “ T_{WEAK} factor”, “ T_{SM} factor”, “LP2SAT factor”, and “ASSAT factor”). The input size is measured as the total number of rules and sentences for PC(ID) and the total number of rules for ASP after grounding with `lparse -d none`. The ASSAT reduction is measured as the number of clauses in the completion plus the number of loop formulas and is averaged over the 5 runs. The two “SCCs” columns give the number of strongly connected components with size greater than 1 for the two formulations. The additional SCCs for the ASP formulation come from the rules defining $hc(x, y)$ as an “open” predicate. The presence of more than one additional SCC for graphs 2xp30 and 4xp20 indicates that they consist of several disconnected components and hence they do not contain a Hamiltonian cycle.

⁶ <http://www.tcs.hut.fi/Software/smodels/tests/lp-csp-tests.tar.gz>

⁷ <http://assat.cs.ust.hk/Assat-2.0/hc-2.0.html>

Graph	PC(ID)			ASP				
	SCCs of size > 1	input rules	T_{WEAK} factor	SCCs of size > 1	input rules	T_{SM} factor	LP2SAT factor	ASSAT factor
p20	1	950	2.89	2	1048	3.79	18.64	2.59
p25	1	1297	2.82	2	1425	3.73	18.06	4.22
p29	1	1577	2.78	2	1729	3.70	17.73	5.02
p30	1	1644	2.78	2	1802	3.70	17.73	5.43
2xp30	2	3288	2.86	4	3604	3.77	18.34	1.47
2xp30.1	2	3350	2.83	3	3668	3.74	18.07	7.20
2xp30.2	1	3330	3.12	2	3648	4.01	21.24	8.96
2xp30.3	1	3330	3.12	2	3648	4.01	21.24	9.14
2xp30.4	1	3330	3.12	2	3648	4.01	21.24	2.01
4xp20	4	3800	3.04	8	4192	3.92	19.78	1.51
4xp20.1	4	3857	3.01	5	4252	3.89	19.55	1.50
4xp20.2	1	3876	3.61	2	4272	4.45	26.18	5.35
4xp20.3	1	3884	3.61	2	4280	4.44	26.13	1.69

Table 2. Size of the input and output of the reductions.

Comparing the size of the non-faithful reduction T_{SM} and the faithful reduction LP2SAT confirms the conjecture that faithful reductions are much bigger — for this example about 5 times. However, the performance of the SAT solver is not proportional to the size of the theory. Even to the contrary, the LP2SAT reduction has a consistently better performance.

7 Conclusion

In this paper we reported on an implementation of a system for finding models of the propositional fragment of FO(ID) by doing a reduction to propositional satisfiability. Our experiments showed that, on satisfiable problems, this approach is competitive with ASP systems — both direct implementations like SMOBELS and SAT based systems like ASSAT and LP2SAT.

We also showed how the reduction from PC(ID) to SAT can be adapted for the stable model semantics. In our experiments we compared this reduction to the faithful reduction LP2SAT [11] and confirmed our hypothesis that developing a one-to-one reduction is much more costly in terms of its size. However, despite this difference in size, the faithful reduction performed better on all examples. To better understand the performance of the different reductions it is necessary to do experiments with other problems and different SAT solvers.

An interesting direction for future research is to follow the approach of ASSAT and CMODELS-2 and try to define loop formulas for two-valued well-founded semantics.

References

- [1] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
- [2] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12(1–2):53–87, 1994.
- [3] R. J. Brachman and H. Levesque. Competence in knowledge representation. In *Proc. of the National Conference on Artificial Intelligence*, pages 189–192, 1982.
- [4] M. Denecker. Extending classical logic with inductive definitions. In *Computational Logic, First International Conference*, volume 1861 of *Lecture Notes in Computer Science*, pages 703–717. Springer, 2000.
- [5] M. Denecker and E. Ternovska. Inductive situation calculus. In *Principles of Knowledge Representation and Reasoning: Proc. of the 9th International Conference*, pages 545–553. AAAI Press, 2004.
- [6] M. Denecker and E. Ternovska. A logic of non-monotone inductive definitions and its modularity properties. In *Logic Programming and Nonmonotonic Reasoning: 7th International Conference*, volume 2923 of *Lecture Notes in Computer Science*, pages 47–60. Springer, 2004.
- [7] M. Denecker, D. Theseider Dupré, and K. Van Belleghem. An inductive definition approach to ramifications. *Linköping Electronic Articles in Computer and Information Science*, 3(7):1–43, 1998. <http://www.ep.liu.se/ea/cis/1998/007/>.
- [8] E. Erdem and V. Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3(4–5):499–518, 2003.
- [9] F. Fages. Consistency of Clark’s completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.
- [10] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, 5th International Conference and Symposium*, pages 1070–1080, 1988.
- [11] T. Janhunen. Representing normal programs with clauses. In *Proc. of the 16th European Conference on Artificial Intelligence*, pages 358–362, 2004.
- [12] Y. Lierler and M. Maratea. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In *Logic Programming and Nonmonotonic Reasoning, 7th International Conference*, volume 2923 of *Lecture Notes in Computer Science*, pages 346–350. Springer, 2004.
- [13] F. Lin and J. Zhao. On tight logic programs and yet another translation from normal logic programs to propositional logic. In *International Joint Conference on Artificial Intelligence*, pages 853–858. Morgan Kaufmann, 2003.
- [14] F. Lin and Y. Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1–2):115–137, 2004.
- [15] T. Linke, H. Tompits, and S. Woltran. On acyclic and head-cycle free nested logic programs. In *20th International Conference on Logic Programming*, volume 3132 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2004.
- [16] M. Mariën, D. Gilis, and M. Denecker. On the relation between ID-Logic and Answer Set Programming. In *Logics in Artificial Intelligence, 9th European Conference (JELIA)*, volume 3229 of *Lecture Notes in Computer Science*, pages 108–120. Springer, 2004.
- [17] D. Mitchell and E. Ternovska. A framework for representing and solving NP-search problems. In *Proc. of the National Conference on Artificial Intelligence*, pages 430–435, 2005.

- [18] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
- [19] L. Ryan. Efficient algorithms for clause-learning SAT solvers. Master’s thesis, Simon Fraser University, Burnaby, Canada, 2004.
- [20] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.
- [21] E. Ternovskaia. Causality via inductive definitions. In *Working Notes of "Prospects for a Commonsense Theory of Causation"*, *AAAI Spring Symposium Series*, pages 94–100, 1998.
- [22] E. Ternovskaia. ID-logic and the ramification problem for the situation calculus. In *Proc. of the 14th European Conference on Artificial Intelligence*, pages 563–567, 2000.
- [23] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
- [24] B. Van Nuffelen, A. Cortés-Calabuig, M. Denecker, O. Arieli, and M. Bruynooghe. Data integration using ID-logic. In *Advanced Information Systems Engineering, Proc. of the 16th International Conference (CAiSE)*, volume 3084 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2004.