An Incremental Answer Set Programming Based System for Finite Model Computation

Martin Gebser, Orkunt Sabuncu and Torsten Schaub
Universität Potsdam, Potsdam, Germany
E-mail: {gebser,orkunt,torsten}@cs.uni-potsdam.de

We address the problem of Finite Model Computation (FMC) of first-order theories and show that FMC can efficiently and transparently be solved by taking advantage of a recent extension of Answer Set Programming (ASP), called incremental Answer Set Programming (iASP). The idea is to use the incremental parameter in iASP programs to account for the domain size of a model. The FMC problem is then successively addressed for increasing domain sizes until an answer set, representing a finite model of the original first-order theory, is found. We implemented a system based on the iASP solver *iClingo* and demonstrate its competitiveness by showing that it slightly outperforms the winner of the FNT division of CADE's 2009 Automated Theorem Proving (ATP) competition on the respective benchmark collection.

Keywords: Incremental Answer Set Programming, Finite Model Computation

1. Introduction

While Finite Model Computation (FMC;[1]) constitutes an established research area in the field of Automated Theorem Proving (ATP;[2]), Answer Set Programming (ASP;[3]) has become a widely used approach for declarative problem solving, featuring manifold applications in the field of Knowledge Representation and Reasoning. Up to now, however, both FMC and ASP have been studied in separation, presumably due to their distinct hosting research fields. We address this gap and show that FMC can efficiently and transparently be solved by taking advantage of a recent extension of ASP, called incremental Answer Set Programming (iASP;[4]).

Approaches to FMC for first-order theories [5,6] fall in two major categories, translational and constraint solving approaches. In translational approaches [7,8], the FMC problem is divided into multiple satisfiability problems in propositional logic. This division is based on the size of the finite domain. A Satisfiability (SAT;[9]) solver searches in turn for a model of the subproblem having a finite domain of fixed size, which is gradually increased until a model is found for the subproblem at hand. In the constraint solving approach [10,11], a system computes a model by incrementally casting FMC into a constraint satisfaction problem. While systems based on constraint solving are efficient for problems with many unit equalities, translation-based ones are applicable to a much wider range of problems [6].

In fact, translational approaches to FMC bear a strong resemblance to iASP. The latter was developed for dealing with dynamic problems like model checking and planning. To this end, iASP foresees an integer-valued parameter that is consecutively increased until a problem is found to be satisfiable. Likewise, in translation-based FMC, the size of the interpretations' domain is increased until a model is found. This similarity in methodologies motivates us to encode and solve FMC by means of iASP.

The idea is to use the incremental parameter in iASP to account for the domain size. Separate subproblems considered in translational approaches are obtained by grounding an iASP encoding, where care is taken to avoid redundancies between subproblems. The parameter capturing the domain size is then successively incremented until an answer set is found. In the successful case, an answer set obtained for parameter value i provides a finite model of the input theory with domain size i.

We implemented a system based on the iASP solver *iClingo* [4] and compared its performance to various FMC systems. To this end, we used problems from the FNT (First-order form Non-Theorems) division of CADE's 2009 and 2010 ATP competitions. The results demonstrate the competitiveness of our system. On the benchmark collection used in 2009, *iClingo* solved the same number of problems as *Paradox* [8] in approximately half of its run time on average. Note that *Paradox* won first places in the FNT division each year from 2007 to 2010.

The paper is organized as follows. The next section introduces basic concepts about the translational approach to FMC and about iASP. Section 3 describes our incremental encoding of FMC and how it is generated from a given set of clauses. Information about our system can be found in Section 4. We empirically evaluate our system in Section 5 and conclude in Section 6. Proofs and an input first-order theory along with logic programs, as used by our FMC system based on iASP, are provided in Appendix A and B, respectively.[1]

2. Background

We assume the reader to be familiar with the terminology and basic definitions of first-order logic and ASP. In what follows, we thus focus on the introduction of concepts needed in the remainder of this paper.

In our method, we translate first-order theories into sets of flat clauses. A clause is *flat* if (i) all its predicates and functions have only variables as arguments, (ii) all occurrences of constants and functions are within equality predicates, and (iii) each equality predicate has at least one variable as an argument. Any first-order clause can be transformed into an equisatisfiable flat clause via *flattening* [7,8,6], done by repeatedly applying the rewrite rule $C[t] \rightsquigarrow (C[X] \vee (X \neq t))$, where t is a term offending flatness and X is a fresh variable. For instance, the clause $(f(X) = g(Y))$ can be turned into the flat clause $(Z = g(Y)) \vee (Z \neq f(X))$. In the translational approach to FMC, flattening is used to bring the input into a form that is easy to instantiate using domain elements.

As regards ASP, we rely on the language supported by grounders *lparse* [12] and *gringo* [13], providing normal and choice rules as well as cardinality and integrity constraints. As usual, rules with variables are regarded as representatives for all respective ground instances. Beyond that, our approach makes use of iASP [4] that allows for dealing with incrementally growing domains. In iASP, a parameterized domain description is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over positive integers. Hence, we sometimes denote P and Q by $P[k]$ and $Q[k]$. The base program B describes static knowledge, independent of parameter k. The role of P is to capture knowledge accumulating with increasing k, whereas Q is specific for each value of k. Our goal is then to decide whether the program

$$R[i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i] \qquad (1)$$

has an answer set for some (minimum) integer $i \geq 1$, where $P[k/j]$ and $Q[k/i]$ refer to the programs obtained from P and Q by replacing each occurrence of constant k with j or i, respectively. In what follows, we refer to rules in B, $P[k]$, and $Q[k]$ as being *static*, *cumulative*, and *volatile*, respectively.

3. Approach

In this section, we present our encoding of FMC in iASP. The first task, associating terms with domain elements, is dealt with in Section 3.1. Based on this, Section 3.2 describes the evaluation of (flat) clauses within iASP programs. In Section 3.3, we explain how a model of a first-order theory is then read off from an answer set. Section 3.4 presents an encoding optimization by means of symmetry breaking. Finally, we show the soundness and completeness of our approach in Section 3.5.

Throughout this section, we illustrate our approach on a running example. Assume that the following first-order theory is given as starting point:

$$\begin{array}{l} p(a) \\ (\forall X)\ \neg q(X,X) \\ (\forall X)\ (p(X) \rightarrow (\exists Y)\ q(X,Y)). \end{array} \qquad (2)$$

The first preprocessing step, clausification of the theory, yields the following:

$$\begin{array}{l} p(a) \\ \neg q(X,X) \\ \neg p(X) \vee q(X, sko(X)). \end{array}$$

The second step, flattening, transforms these clauses into the following ones:

$$\begin{array}{l} p(X) \vee (X \neq a) \\ \neg q(X,X) \\ \neg p(X) \vee q(X,Y) \vee (Y \neq sko(X)). \end{array} \qquad (3)$$

[1]This paper extends a previous JELIA'10 contribution with the same title by providing additional benchmark results and proofs.

Such flat clauses form the basis for our iASP encoding. Before we present it, note that the theory in (3) has a model I over domain $\{1, 2\}$ given by:

$$\begin{aligned} a^I &= 1 \\ \mathit{sko}^I &= \{1 \mapsto 2, 2 \mapsto 2\} \\ p^I &= \{1\} \\ q^I &= \{(1, 2)\}. \end{aligned} \tag{4}$$

Importantly, I is also a model of the original theory in (2), even if sko^I is dropped.

3.1. Interpreting Terms

In order to determine a model, we need to associate the (non-variable) terms in the input with domain elements. To this end, every constant c is represented by a fact $\mathit{cons}(c).$, belonging to the *static* part of our iASP program. For instance, the constant a found in (3) gives rise to the following fact:

$$\mathit{cons}(a). \tag{5}$$

Our iASP encoding uses the predicate $\mathit{assign}(T, D)$ to represent that a term T is mapped to a domain element D. Here and in the following, we write k to refer to the incremental variable in an iASP program. Unless stated otherwise, all rules provided in the sequel are *cumulative* by default. For constants, the following (choice) rule then allows for mapping them to the kth domain element:

$$\{\mathit{assign}(T, k)\} \leftarrow \mathit{cons}(T). \tag{6}$$

Note that, by using k in $\mathit{assign}(T, k)$, it is guaranteed that instances of the rule are particular to each incremental step.

Unlike with constants, the argument tuples of (non-zero arity) functions grow when k increases. To deal with this, we first declare auxiliary facts to represent available domain elements:

$$\mathit{dom}(k). \qquad \mathit{arg}(k, k). \tag{7}$$

Predicates dom and arg are then used to qualify the arguments of an n-ary function f in the following rule:

$$\begin{aligned} &\mathit{func}(f(X_1, \ldots, X_n)) \leftarrow \\ &\quad \mathit{dom}(X_1), \ldots, \mathit{dom}(X_n), \\ &\quad 1\{\mathit{arg}(X_1, k), \ldots, \mathit{arg}(X_n, k)\}. \end{aligned} \tag{8}$$

The cardinality constraint $1\{\mathit{arg}(X_1, k), \ldots, \mathit{arg}(X_n, k)\}$ stipulates at least one of the arguments $X_1, \ldots, X_n$ of f to be k. As in (6), though using a different methodology, this makes sure that the (relevant) instances are particular to a value of k. However, note that rules of the above form need to be provided separately for each function in the input, given that the arities of functions matter. For the unary function sko in (3), applying the described scheme leads to the following rule:

$$\begin{aligned} &\mathit{func}(\mathit{sko}(X)) \leftarrow \\ &\quad \mathit{dom}(X), 1\{\mathit{arg}(X, k)\}. \end{aligned} \tag{9}$$

To represent new mappings via a function when k increases, the previous methodology can easily be extended to requiring some argument or alternatively the function value to be k. The following (choice) rule encodes mappings via an n-ary function f:

$$\begin{aligned} &\{\mathit{assign}(f(X_1, \ldots, X_n), Y)\} \leftarrow \\ &\quad \mathit{dom}(X_1), \ldots, \mathit{dom}(X_n), \mathit{dom}(Y), \\ &\quad 1\{\mathit{arg}(X_1, k), \ldots, \mathit{arg}(X_n, k), \mathit{arg}(Y, k)\}. \end{aligned} \tag{10}$$

For instance, the rule encoding mappings via unary function sko is as follows:

$$\begin{aligned} &\{\mathit{assign}(\mathit{sko}(X), Y)\} \leftarrow \\ &\quad \mathit{dom}(X), \mathit{dom}(Y), \\ &\quad 1\{\mathit{arg}(X, k), \mathit{arg}(Y, k)\}. \end{aligned} \tag{11}$$

Observe that the cardinality constraint $1\{\mathit{arg}(X, k), \mathit{arg}(Y, k)\}$ necessitates at least one of argument X or value Y of function sko to be k, which in the same fashion as before makes the (relevant) instances of the rule particular to each incremental step.

To see how the previous rules are handled in iASP computations, we below show the instances of (7) and (11) generated in and accumulated over three incremental steps:

Step 1

$$\mathit{dom}(1). \quad \mathit{arg}(1, 1).$$
$$\{\mathit{assign}(\mathit{sko}(1), 1)\}.$$

Step 2

$$\mathit{dom}(2). \quad \mathit{arg}(2, 2).$$
$$\{\mathit{assign}(\mathit{sko}(1), 2)\}.$$
$$\{\mathit{assign}(\mathit{sko}(2), 1)\}.$$
$$\{\mathit{assign}(\mathit{sko}(2), 2)\}.$$

Step 3

$$dom(3). \quad arg(3,3).$$
$$\{assign(sko(1),3)\}.$$
$$\{assign(sko(2),3)\}.$$
$$\{assign(sko(3),1)\}.$$
$$\{assign(sko(3),2)\}.$$
$$\{assign(sko(3),3)\}.$$

Given that the body of (11) only relies on facts (over predicates dom and arg), its ground instances can be evaluated and then be reduced: if a ground body holds, the corresponding (choice) head is generated in a step; otherwise, the ground rule is trivially satisfied and needs not be considered any further. Hence, all rules shown above have an empty body after grounding. Notice, for example, that rule $\{assign(sko(1),1)\}.$ is generated in the first step, while it is not among the new ground rules in the second and third step.

Finally, a mapping of terms to domain elements must be unique and total. To this end, translation-based FMC approaches add uniqueness and totality axioms for each term to an instantiated theory. In iASP, such requirements can be encoded as follows:

$$\leftarrow assign(T,D), assign(T,k), D < k. \qquad (12)$$

$$\leftarrow cons(T), \{assign(T,D) : dom(D)\}0. \qquad (13)$$

$$\leftarrow func(T), \{assign(T,D) : dom(D)\}0. \qquad (14)$$

While the integrity constraint in (12) forces the mapping of each term to be unique, the ones in (13) and (14) stipulate each term to be mapped to some domain element. However, since the domain grows over incremental steps and new facts are added for predicate dom, ground instances of (13) and (14) are only valid in the step where they are generated. Hence, the integrity constraints in (13) and (14) belong to the *volatile* part of our iASP program.

3.2. Interpreting Clauses

To evaluate an input theory, we also need to interpret its predicates. To this end, we include a rule of the following form for every n-ary predicate p in our iASP program:

$$\begin{aligned}&\{p(X_1,\dots,X_n)\} \leftarrow \\ &\quad dom(X_1),\dots,dom(X_n), \\ &\quad 1\{arg(X_1,k),\dots,arg(X_n,k)\}. \end{aligned} \qquad (15)$$

As discussed above, requiring $1\{arg(X_1,k),\dots,$ $arg(X_n,k)\}$ to hold guarantees that (relevant) instances are particular to each incremental step. The only exception to this is $n = 0$ (a predicate p of arity zero), in which case the rule $\{p\}.$ belongs to the *static* part of our program. Also note that, unlike constants and functions, we do not reify predicates, as assigning a truth value can be expressed more naturally without it. For example, the following rules allow for interpreting the predicates p and q in (3):

$$\begin{aligned}\{p(X)\} &\leftarrow dom(X), 1\{arg(X,k)\}. \\ \{q(X,Y)\} &\leftarrow dom(X), dom(Y), \\ &\quad 1\{arg(X,k), arg(Y,k)\}. \end{aligned} \qquad (16)$$

Following [14], the basic idea of encoding a (flat) clause is to represent it by an integrity constraint containing the complements of the literals in the clause. However, clauses may contain equality literals of the form $(X = Y)$ or $(X \neq Y)$, where at least one of the terms X and Y is a variable, and so we also need to consider complements of such literals. W.l.o.g., we below assume that the left-hand side of every equality literal is a variable, while the right-hand side is either a variable or a non-variable term. In view of this convention, we define the encoding $\overline{L}$ of the complement of a (classical or equality) literal L as follows:

$$\overline{L} = \begin{cases} not\ p(X_1,\dots,X_n) \\ \text{if } L = p(X_1,\dots,X_n) \\ p(X_1,\dots,X_n) \\ \text{if } L = \neg p(X_1,\dots,X_n) \\ not\ assign(t,X) \\ \text{if } L = (X = t) \text{ for some non-variable term } t \\ assign(t,X) \\ \text{if } L = (X \neq t) \text{ for some non-variable term } t \\ X \neq Y \\ \text{if } L = (X = Y) \text{ for some variable } Y \\ X = Y \\ \text{if } L = (X \neq Y) \text{ for some variable } Y. \end{cases}$$

Observe that the first two cases refer to the interpretation of a predicate p, the third and the fourth to the mapping of non-variable terms to domain elements, and the last two to built-in comparison operators of grounders like *lparse* and *gringo*.

With the complements of literals at hand, we can now encode a flat clause containing literals $L_1,\dots,L_m$ and variables $X_1,\dots,X_n$ by an integrity constraint as follows:

$$\leftarrow \overline{L}_1, \dots, \overline{L}_m, dom(X_1), \dots, dom(X_n), \\ 1\{arg(X_1, k), \dots, arg(X_n, k)\}. \quad (17)$$

Note that we use the same technique as before to separate the (relevant) instances obtained at each incremental step. For our running example, the clauses in (3) give rise to the following integrity constraints:

$$\leftarrow not\ p(X), assign(a, X), dom(X), 1\{arg(X, k)\}.$$
$$\leftarrow q(X, X), dom(X), 1\{arg(X, k)\}.$$
$$\leftarrow p(X), not\ q(X, Y), assign(sko(X), Y), \\ dom(X), dom(Y), 1\{arg(X, k), arg(Y, k)\}. \quad (18)$$

While the first two integrity constraints each contribute a single instance at an incremental step, $(2 * k) - 1$ instances are obtained for the third one.

Although they are unlikely to occur in first-order theories, *propositional* clauses without variables and equality literals require a slightly different treatment. For a propositional clause containing (classical) literals $L_1, \dots, L_m$, instead of (17), we include the following simpler integrity constraint in the *static* part of our iASP program:

$$\leftarrow \overline{L}_1, \dots, \overline{L}_m. \quad (19)$$

3.3. Extracting Models

The rules that represent the mapping of terms to domain elements (described in Section 3.1) along with those representing satisfiability of flat clauses (described in Section 3.2) constitute our iASP program for FMC. To compute an answer set, the incremental variable k is increased by one at each step. This corresponds to the addition of a new domain element. If an answer set is found in a step i, it means that the input theory has a model over a domain of size i. In fact, from an answer set A of our iASP program, a model I of the input theory over domain $\{d \mid dom(d) \in A\}$ is extracted as follows:

$$c^I = d \text{ if } cons(c) \in A \text{ and } assign(c, d) \in A,$$
$$f^I = \{(d_1, \dots, d_n) \mapsto d \mid \\ func(f(d_1, \dots, d_n)) \in A, \\ assign(f(d_1, \dots, d_n), d) \in A\},$$
$$p^I = \{(d_1, \dots, d_n) \mid p(d_1, \dots, d_n) \in A\}.$$

For the iASP program encoding the theory in (3), composed of the rules in (5–7, 9, 11–14, 16, 18), the following answer set is obtained in the second incremental step:

$$\left\{ \begin{array}{l} dom(1), dom(2), arg(1,1), arg(2,2), \\ cons(a), assign(a, 1), \\ func(sko(1)), assign(sko(1), 2), \\ func(sko(2)), assign(sko(2), 2), \\ p(1), q(1, 2) \end{array} \right\}$$

The corresponding model over domain $\{1, 2\}$ is the one shown in (4).

3.4. Breaking Symmetries

In view of the fact that interpretations obtained by permuting domain elements are isomorphic, an input theory can have many symmetric models. For example, an alternative model to the one in (4) can easily be obtained by swapping domain elements 1 and 2. Such symmetries tend to degrade the performance of FMC systems. Hence, systems based on the constraint solving approach, such as *Sem* and *Falcon*, apply variants of a dynamic symmetry breaking technique called least number heuristic [11]. Translation-based systems, such as *Paradox* and *FM-Darwin*, staticly break symmetries by narrowing how terms can be mapped to domain elements.

Our approach to symmetry breaking is also a static one that aims at reducing the possibilities of mapping constants to domain elements. To this end, we use the technique described in [8,15], fixing an order of the constants in the input by uniquely assigning a rank in $[1, n]$, where n is the total number of constants, to each of them. Given such a ranking in terms of facts over predicate *order*, we can replace the rule in (6) with:

$$\{assign(T, k)\} \leftarrow cons(T), order(T, O), k \leq O.$$

For instance, if the order among three constants, c_1, c_2, and c_3, is given by facts $order(c_i, i)$. for $i \in \{1, 2, 3\}$, the following instances of the above rule are generated in and accumulated over three incremental steps:

Step 1

$$\{assign(c_1, 1)\}.$$
$$\{assign(c_2, 1)\}.$$
$$\{assign(c_3, 1)\}.$$

Step 2

$$\{assign(c_2, 2)\}.$$
$$\{assign(c_3, 2)\}.$$

Step 3

$$\{assign(c_3, 3)\}.$$

That is, while all three constants can be mapped to the first domain element, c_1 cannot be mapped to the second one, and only c_3 can be mapped to the third one. Note that, despite of these restrictions, the above rules still admit mappings like $c_1^I = 1, c_2^I = 1, c_3^I = 3$.

To further disambiguate the mapping of constants to domain elements, the following rules can be added:

$$\begin{aligned}
\mathit{assigned}(T,k) &\leftarrow \mathit{order}(S,O), \mathit{order}(T,O{+}1),\\
& \mathit{assign}(S,k).\\
\mathit{assigned}(T,k) &\leftarrow \mathit{order}(S,O), \mathit{order}(T,O{+}1),\\
& \mathit{assigned}(S,k).\\
&\leftarrow \mathit{cons}(T), \mathit{assign}(T,k), 1 < k,\\
& \mathit{not}\ \mathit{assigned}(T,k{-}1).
\end{aligned}$$

The idea is to propagate the mapping of a constant to the ones that succeed it in the given order. Then, an integrity constraint is used to stipulate that there are no gaps in the sequence of domain elements to which constants are mapped. When reconsidering the constants c_1, c_2, c_3, the following (relevant) instances of the above rules are generated in and accumulated over three incremental steps:

Step 1

$$\begin{aligned}
\mathit{assigned}(c_2,1) &\leftarrow \mathit{assign}(c_1,1).\\
\mathit{assigned}(c_3,1) &\leftarrow \mathit{assign}(c_2,1).\\
\mathit{assigned}(c_3,1) &\leftarrow \mathit{assigned}(c_2,1).
\end{aligned}$$

Step 2

$$\begin{aligned}
&\mathit{assigned}(c_3,2) \leftarrow \mathit{assign}(c_2,2).\\
&\leftarrow \mathit{assign}(c_2,2), \mathit{not}\ \mathit{assigned}(c_2,1).\\
&\leftarrow \mathit{assign}(c_3,2), \mathit{not}\ \mathit{assigned}(c_3,1).
\end{aligned}$$

Step 3

$$\leftarrow \mathit{assign}(c_3,3), \mathit{not}\ \mathit{assigned}(c_3,2).$$

For the mapping $c_1^I = 1, c_2^I = 1, c_3^I = 3$, represented by the atoms $\mathit{assign}(c_1,1)$, $\mathit{assign}(c_2,1)$, and $\mathit{assign}(c_3,3)$, we derive two atoms over predicate $\mathit{assigned}$: $\mathit{assigned}(c_2,1)$ and $\mathit{assigned}(c_3,1)$. That is, the integrity constraint generated in Step 3 refutes the mapping at hand. On the other hand, all integrity constraints are satisfied when we switch from $c_3^I = 3$ to $c_3^I = 2$. In fact, the admissible mappings of constants to domain elements are exactly the ones referred to by the term "canonical form" in [8].

Finally, we note that our iASP encoding of the theory in (3) yields 10 answer sets in the second incremental step. If we apply the described symmetry breaking, it disallows mapping the single constant a to the second domain element, which prunes 5 of the 10 models. Although our simple technique can in general not break all symmetries related to the mapping of terms because it does not incorporate functions, the experiments in Section 5 demonstrate that it may nonetheless lead to significant performance gains. Unlike with constants, given a priori, additionally incorporating functions into our approach to symmetry breaking would require the extension of predicate order to newly composed functional terms in each incremental step. For the special case of unary functions, such an extension [8] is implemented in *Paradox*; with *FM-Darwin*, it has not turned out to be more effective than symmetry breaking for only constants [15].

3.5. Soundness and Completeness

Before stating our theorem, we first define the parameterized domain description formed for a set $\mathcal{T}$ of flat clauses. The signature $\langle \mathcal{F}_0, \mathcal{F}, \mathcal{P}_0, \mathcal{P}\rangle$ of $\mathcal{T}$ is built from a set $\mathcal{F}_0$ of *constants*, a set $\mathcal{F}$ of (non-zero arity) *functions*, a set $\mathcal{P}_0$ of *zero arity predicates*, and a set $\mathcal{P}$ of *non-zero arity predicates*. For $\mathcal{T}$, we then form the parameterized domain description (B, P, Q) in the following way:

$$B = \{\mathit{cons}(c).\mid c \in \mathcal{F}_0\} \cup \{\{p\}.\mid p \in \mathcal{P}_0\} \cup \Pi^{\mathcal{T}_0},$$

$$P = \left\{\begin{array}{l} \mathit{dom}(k).\quad \mathit{arg}(k,k).\\ \{\mathit{assign}(T,k)\} \leftarrow \mathit{cons}(T).\\ \leftarrow \mathit{assign}(T,D), \mathit{assign}(T,k), D < k. \end{array}\right\}$$
$$\cup\ \Pi^{\mathcal{F}} \cup \Pi^{\mathcal{P}} \cup \Pi^{\mathcal{T}},\ \text{and}$$

$$Q = \left\{\begin{array}{l} \leftarrow \mathit{cons}(T), \{\mathit{assign}(T,D) : \mathit{dom}(D)\}0.\\ \leftarrow \mathit{func}(T), \{\mathit{assign}(T,D) : \mathit{dom}(D)\}0. \end{array}\right\}$$

where $\Pi^{\mathcal{F}}$ contains rules of the form (8) and (10) for each function $f \in \mathcal{F}$, $\Pi^{\mathcal{P}}$ contains a rule of the form (15) for each predicate $p \in \mathcal{P}$, $\Pi^{\mathcal{T}}$ contains a rule of the form (17) for each non-propositional clause in $\mathcal{T}$, and $\Pi^{\mathcal{T}_0}$ contains a rule of the form (19) for each propositional clause in $\mathcal{T}$. With these concepts at hand, we are ready to formulate the soundness and completeness of our approach.

Theorem 1 (Soundness & Completeness)**.** *Let $\mathcal{T}$ be a set of flat clauses and (B, P, Q) the parameterized domain description for $\mathcal{T}$. Then, the logic program $R[i]$, as defined in (1), has an answer set for some positive integer i iff $\mathcal{T}$ has a model over a finite domain of size i.*

Note that the theorem still applies when including symmetry breaking, as described in the previous section, in view of the fact that it may eliminate some isomorphic models, but not all of them. A proof of Theorem 1 is provided in Appendix A.

4. System

We use *FM-Darwin* to read an input in TPTP format, a format for first-order theories widely used within the community of ATP, to clausify it if needed, and to flatten the clauses at hand. Additionally, *FM-Darwin* applies some input optimizations before flattening, such as renaming deep ground subterms to avoid the generation of flat clauses with many variables [15]. For obtaining flat clauses from an input theory specified in a file `tptp_input.p`, *FM-Darwin* is invoked as follows:

```
darwin -fd true -pfdp Exit
tptp_input.p
```

Having an input in terms of flat clauses, we can apply the transformations described in Section 3.1 and 3.2 to generate an iASP program. To this end, we implemented a compiler called *fmc2iasp*[2], written in Python. It outputs the rules that are specific to an input theory, while the theory-independent rules in (6), (7), and (12–14) are provided in a separate file. This separation allows us to test encoding variants without changing *fmc2iasp*, for instance, the symmetry breaking described in Section 3.4. Finally, we use *iClingo* to incrementally ground the obtained iASP program and to search for answer sets representing finite models of the input theory. Provided that `fmc.lp` is the file containing theory-independent rules, the following command-line call is used for FMC:

```
darwin -fd true -pfdp Exit
tptp_input.p | fmc2iasp.py | cat
fmc.lp - | iclingo
```

5. Experiments

We consider the following systems: *iClingo* (2.0.5), *Clingo* (2.0.5), *Paradox* (3.0), *FM-Darwin* (1.4.5), and *Mace4* (2009-11A). While *Paradox* and *FM-Darwin* are based on the translational approach to FMC, *Mace4* applies the constraint solving approach. For *iClingo* and *Clingo*, we used command line switch `--heuristic=VSIDS`, as it improved search performance.[3] Our experiments have been performed on a 3.4GHz Intel Xeon machine running Linux, imposing 300 seconds as time and 2GB as memory limit.

FMC instances stem from the FNT division of CADE's 2009 and 2010 ATP competitions. The instances in this division are satisfiable and suitable for evaluating FMC systems, among which *Paradox* won the first place in both years' competitions. The considered problem domains are: computing theory (COM), common-sense reasoning (CSR), geography (GEG), geometry (GEO), graph theory (GRA), groups (GRP), homological algebra (HAL), knowledge representation (KRS), lattices (LAT), logic calculi (LCL), medicine (MED), management (MGT), miscellaneous (MSC), natural language processing (NLP), number theory (NUM), planning (PLA), processes (PRO), rings in algebra (RNG), software verification (SWV), syntactic (SYN).[4]

Table 1 and 2 show benchmark results for each of the problem domains. Column # displays how many instances of a problem domain belong to the test suite. For each system and problem domain, average run time in seconds is taken over the solved instances; their number is given in parentheses.[5] A dash in an entry means that a system could not solve any instance of the corresponding problem domain within the run time and memory limits. For each system, the last row shows its average run time over all solved instances and provides their number in parentheses. The evaluation criteria in CADE competitions are first number of solved instances and then average run time as tie breaker.

In Table 1, we see that *Mace4* and *FM-Darwin* solved 50 and 82 instances, respectively, out of the 99 instances in total.[6] *Paradox*, the winner of the FNT division in CADE's 2009 ATP competition, solved 92 instances in 6.05 seconds on average. While the version of our system not using symmetry breaking (described in Section 3.4), denoted by *iClingo (2)*, solved two instances less, the one with symmetry breaking, denoted by *iClingo (1)*, also solved 92 instances. As it spent only 2.29 seconds on average, according to the CADE

[2]`http://potassco.sourceforge.net/`

[3]Note that *Minisat*, used internally by *Paradox*, also applies VSIDS as decision heuristic [16].

[4]`http://www.cs.miami.edu/~tptp/`

[5]Run time results of our system include the time for preprocessing by *FM-Darwin*; it is negligible compared to the model finding time.

[6]The FNT division included 100 instances in the 2009 competition. We dropped one instance that had an error in the encoding because the corrected version is unsatisfiable.

Benchmark	#	*iClingo (1)*	*iClingo (2)*	*Clingo*	*Paradox*	*FM-Darwin*	*Mace4*
CSR	1	2.87 (1)	2.30 (1)	6.26 (1)	—	20.56 (1)	—
GEG	1	—	—	—	230.36 (1)	—	—
GEO	12	0.08 (12)	0.09 (12)	0.11 (12)	0.08 (12)	0.09 (12)	0.04 (12)
GRA	2	3.44 (1)	—	12.78 (1)	0.49 (1)	—	—
GRP	1	4.25 (1)	216.96 (1)	6.31 (1)	0.63 (1)	—	0.28 (1)
HAL	2	2.52 (2)	2.46 (2)	2.94 (2)	0.67 (2)	11.84 (1)	—
KRS	6	0.14 (6)	0.16 (6)	0.27 (6)	0.11 (6)	30.87 (6)	0.03 (4)
LAT	5	0.10 (5)	0.11 (5)	0.13 (5)	0.12 (5)	0.08 (5)	0.04 (5)
LCL	17	8.62 (17)	9.50 (17)	10.86 (17)	3.70 (17)	1.65 (17)	5.10 (8)
MGT	4	0.08 (4)	0.09 (4)	0.10 (4)	0.06 (4)	0.12 (4)	1.09 (4)
MSC	3	4.70 (2)	0.23 (1)	12.58 (2)	122.56 (2)	0.19 (1)	—
NLP	9	1.66 (9)	2.03 (9)	3.17 (9)	0.24 (9)	0.26 (8)	22.07 (1)
NUM	1	0.19 (1)	0.24 (1)	0.28 (1)	0.27 (1)	0.11 (1)	201.51 (1)
PRO	9	1.09 (9)	9.03 (9)	2.02 (9)	0.34 (9)	0.77 (9)	31.53 (7)
SWV	8	0.15 (4)	0.14 (4)	0.20 (4)	0.13 (4)	44.84 (5)	0.04 (2)
SYN	18	0.59 (18)	0.57 (18)	0.72 (18)	0.40 (18)	3.84 (12)	0.68 (5)
Total	99	2.29 (92)	5.55 (90)	3.32 (92)	6.05 (92)	6.43 (82)	9.88 (50)

Table 1
Benchmark results for problems in the FNT division of CADE's 2009 ATP competition.

Benchmark	#	*iClingo (1)*	*iClingo (2)*	*Clingo*	*Paradox*	*FM-Darwin*	*Mace4*
COM	2	0.21 (2)	0.28 (2)	0.39 (2)	0.23 (2)	0.09 (2)	0.05 (2)
GEO	1	0.06 (1)	0.11 (1)	0.07 (1)	0.05 (1)	0.07 (1)	0.03 (1)
GRA	2	101.08 (1)	—	207.62 (1)	8.21 (2)	15.56 (1)	207.37 (1)
GRP	1	0.08 (1)	0.12 (1)	0.12 (1)	0.03 (1)	0.04 (1)	0.03 (1)
HAL	1	2.48 (1)	2.37 (1)	2.73 (1)	0.33 (1)	—	—
KRS	1	0.09 (1)	0.11 (1)	0.12 (1)	0.05 (1)	0.05 (1)	0.03 (1)
LCL	52	5.43 (42)	5.25 (41)	4.57 (40)	3.81 (49)	6.22 (42)	8.08 (19)
MED	1	0.11 (1)	0.11 (1)	0.15 (1)	0.08 (1)	0.06 (1)	0.02 (1)
MSC	1	—	—	—	—	—	—
NLP	52	41.59 (21)	38.29 (22)	35.49 (22)	0.21 (32)	15.85 (49)	3.71 (2)
NUM	9	0.17 (9)	0.19 (9)	0.24 (9)	0.19 (9)	0.10 (9)	22.61 (8)
PLA	4	70.41 (4)	0.27 (3)	0.52 (3)	0.20 (4)	0.23 (4)	0.26 (4)
RNG	2	0.19 (2)	0.26 (2)	0.29 (2)	0.23 (2)	0.20 (2)	287.29 (1)
SWV	2	—	—	—	—	—	—
SYN	12	7.13 (12)	7.18 (12)	7.96 (12)	5.46 (12)	68.76 (9)	—
Total	143	16.07 (98)	11.98 (96)	13.28 (96)	2.38 (117)	13.73 (122)	20.43 (41)

Table 2
Benchmark results for problems in the FNT division of CADE's 2010 ATP competition, restricted to instances not already used in 2009.

criteria, our system slightly outperformed *Paradox*. For assessing the advantages due to incremental grounding and solving, we also ran *Clingo*, performing iterative deepening search by successively grounding and solving our iASP encoding for fixed domains of increasing size. The average run time achieved with *Clingo*, 3.32 seconds, is substantially greater than the one of *iClingo (1)*; the gap becomes more apparent the more domain elements (not shown in Table 1) are needed.

Although there are 200 instances in the FNT division of CADE's 2010 ATP competition, we only show 143 of them in Table 2. (The 57 remaining instances were already used in 2009.) *Mace4* solved 41 of these 143 instances. *Paradox*, the winner of the FNT division also in 2010, solved 117 instances in 2.38 seconds on average. When considering all 200 instances used in 2010, *Paradox* solved 167 of them in 2.55 seconds on average, and *FM-Darwin* also solved 167 instances, but in 12.99 seconds on average. In fact, *FM-Darwin* solved the most instances in Table 2, 122 out of 143. Like with the results shown in Table 1, the version of our system with symmetry breaking, *iClingo (1)*, solved two more instances than the one without symmetry breaking, *iClingo (2)*. Furthermore, the advantages due to incremental grounding and solving are more apparent in Table 2, viewing that *Clingo* solved two instances less than *iClingo (1)*, 96 compared to 98. We however observe that *iClingo* cannot keep step with *Paradox* and *FM-Darwin* on the instances in Table 2; possible reasons are elaborated on next.

A general problem with the translational approach is that flattening may increase the number of variables in a clause, which can deteriorate grounding performance. We can observe this when comparing the results of *FM-Darwin* on the NLP domain in Table 2 with those of *iClingo* and *Paradox*: *FM-Darwin* solved 17 instances more than *Paradox* and 28 more than *iClingo (1)*. Although *FM-Darwin* also pursues a translational approach, it represents subproblems by function-free first-order clauses and uses *Darwin*, not relying on grounding [15], to solve them. An instance of the SWV group (instance SWV484+2) in Table 1 provides an extreme example for the infeasibility of grounding: it includes predicates of arity 34 and is solved only by *FM-Darwin*. Furthermore, in comparison to *iClingo*, sort inference [8,15] promotes *Paradox* and *FM-Darwin* on the instances of NLP in Table 2. This shows that there is still potential to improve our iASP approach to FMC. On the other hand, for the instances of CSR and MSC in Table 1, we speculate that clausification and further preprocessing steps of *Paradox* may be the cause for its deteriorated performance.

6. Discussion

We presented an efficient yet transparent approach to computing finite models of first-order theories by means of ASP. Our approach takes advantage of an incremental extension of ASP that allows us to consecutively search for models with given domain size by incrementing the corresponding parameter in the iASP encoding. The declarative nature of our approach makes it easily modifiable and leaves room for further improvements. Moreover, our approach is rather competitive and has even a slight edge on the winner of the FNT division of CADE's 2009 ATP competition on the respective benchmark collection. Finally, our approach complements the work in [17], where FMC systems were used for computing the answer sets of tight[7] logic programs in order to circumvent grounding.

In [18], a special class of first-order formulas, called Effectively Propositional (EPR) formulas, was addressed via ASP; application domains like planning and bounded model checking have been encoded by EPR formulas and were successfully tackled by means of FMC [19,20]. EPR formulas do not contain functions of non-zero arity in their clause forms. Although our approach takes more general input than this, it can currently not decide EPR formulas. To this end, we had either to extract a bound on the incremental parameter to make the system halt or to provide an alternative dedicated encoding for EPR formulas. Such extensions are interesting topics for future research.

Acknowledgments. This work was supported by the German Science Foundation (DFG) under grants SCHA 550/8-1/2. We are grateful to the anonymous referees of previous workshop and conference versions as well as of this paper for their helpful comments.

References

[1] Caferra, R., Leitsch, A., Peltier, N.: Automated Model Building. Kluwer Academic (2004)

[2] Bibel, W.: Automated Theorem Proving. Vieweg (1987)

[3] Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University (2003)

[4] Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an incremental ASP solver. In Garcia de la Banda, M., Pontelli, E., eds.: Proceedings of the 24th International Conference on Logic Programming (ICLP'08), Springer (2008) 190–205

[5] Zhang, J., Huang, Z.: Reducing symmetries to generate easier SAT instances. Electronic Notes in Theoretical Computer Science **125**(3) (2005) 149–164

[7]Tight programs are free of recursion through positive literals (cf. [3]).

[6] Tammet, T.: Finite model building: Improvements and comparisons. In Baumgartner, P., Fermüller, C., eds.: Proceedings of the Workshop on Model Computation — Principles, Algorithms, Applications (MODEL'03), (2003)

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

[8] Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model finding. In Baumgartner, P., Fermüller, C., eds.: Proceedings of the Workshop on Model Computation — Principles, Algorithms, Applications (MODEL'03), (2003)

[9] Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability. IOS (2009)

[10] Zhang, J., Zhang, H.: SEM: A system for enumerating models. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95), Morgan Kaufmann (1995) 298–303

[11] Zhang, J.: Constructing finite algebras with FALCON. Journal of Automated Reasoning **17**(1) (1996) 1–22

[12] Syrjänen, T.: `Lparse` 1.0 user's manual. http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz

[13] Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`. http://potassco.sourceforge.net

[14] Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234

[15] Baumgartner, P., Fuchs, A., de Nivelle, H., Tinelli, C.: Computing finite models by reduction to function-free clause logic. Journal of Applied Logic **7**(1) (2009) 58–74

[16] Eén, N., Sörensson, N.: An extensible SAT-solver. In Giunchiglia, E., Tacchella, A., eds.: Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03), Springer (2003) 502–518

[17] Sabuncu, O., Alpaslan, F.: Computing answer sets using model generation theorem provers. In Costantini, S., Watson, R., eds.: Proceedings of the 4th International Workshop on Answer Set Programming (ASP'07), (2007) 225–240

[18] Lierler, Y., Lifschitz, V.: Logic programs vs. first-order formulas in textual inference. http://z.cs.utexas.edu/users/ai-lab/publications_recent.php

[19] Navarro Pérez, J., Voronkov, A.: Planning with effectively propositional logic. In Podelski, A., Voronkov, A., Wilhelm, R., eds.: Volume in Memoriam of Harald Ganzinger, Springer (2008)

[20] Navarro Pérez, J., Voronkov, A.: Encodings of bounded LTL model checking in effectively propositional logic. In Pfenning, F., ed.: Proceedings of the 21st International Conference on Automated Deduction (CADE'07), Springer (2007) 346–361

Appendix

A. Proof of Theorem 1

In order to prove Theorem 1, Section A.1 first provides a formal account of answer set semantics for the logic programs under consideration. In Section A.2 and A.3, we formulate and prove lemmas on the soundness and the completeness, respectively, of our iASP encoding of FMC. Finally, our soundness and completeness result, Theorem 1, and its validity in the context of symmetry breaking, as described in Section 3.4, are elaborated on in Section A.4.

A.1. Answer Set Semantics

The semantics of a (non-ground) logic program R is given by the answer sets of the ground instantiation of R. A program R written in the language supported by grounders *lparse* [12] and *gringo* [13] can contain built-in predicates $(=, \neq, \leq, <, \dots)$ as well as cardinality constraints $(l\{p(\vec{x}) : q(\vec{y}), \dots\}u)$. In order to define the ground instantiation of R, we thus consider a signature $\langle \mathbb{F}, \mathbb{P}, \mathbb{I} \rangle$ such that $\mathbb{P} \cap \mathbb{I} = \emptyset$ where

- $\mathbb{F}$ is a set of function symbols with an associated arity (possibly zero),
- $\mathbb{P}$ is a set of predicate symbols with an associated arity (possibly zero) that includes $\bot/0$ and $\top/0$, and
- $\mathbb{I}$ is a set of built-in predicates with an associated arity (possibly zero) that includes $=/2$, $\neq/2$, $</2$, and $\leq/2$.

The Herbrand universe $\mathbb{U}$ consists of all terms that can be constructed in the standard way from the function symbols in $\mathbb{F}$. The Herbrand base $\mathbb{B}$ consists of all atoms constructed from predicate symbols $p/n \in \mathbb{P}$ and n terms from $\mathbb{U}$. (Note that $\mathbb{B}$ does not include atoms over built-in predicates from $\mathbb{I}$.)

Let us introduce some notations for a rule r of the form

$$a_0 \leftarrow a_1, \dots, a_m, \mathit{not}\ a_{m+1}, \dots, \mathit{not}\ a_n. \qquad (20)$$

By $\mathit{head}(r) = a_0$ and $\mathit{body}(r) = \{a_1, \dots, a_m, \mathit{not}\ a_{m+1}, \dots, \mathit{not}\ a_n\}$, we denote the *head* and the *body* of r, respectively. A fact $a_0.$ is understood as a shorthand for the rule $a_0 \leftarrow \top.$, so that $\mathit{body}(a_0.) = \{\top\}$. Likewise, an integrity constraint r of the form $\leftarrow a_1, \dots, a_m, \mathit{not}\ a_{m+1}, \dots, \mathit{not}\ a_n.$ is identified with the rule $\bot \leftarrow \mathit{body}(r).$, so that $\mathit{head}(r) = \{\bot\}$.

For an atom a over a predicate from $\mathbb{P} \cup \mathbb{I}$, we denote the set of all first-order variables occurring in a by $\mathit{var}(a)$. We define $\mathit{var}(c) = \emptyset$ for a cardinality constraint c. (Also note that $\mathit{var}(\top) = \mathit{var}(\bot) = \emptyset$.) For a rule r of the form (20), the set of *global* variables occurring in r is $\mathit{var}(r) = \bigcup_{0 \leq i \leq n} \mathit{var}(a_i)$. A substitution $\theta : var(r) \to \mathbb{U}$ maps each global variable of r to some term in $\mathbb{U}$. We denote the rule obtained by applying θ to r by $r\theta$.

To deal with non-ground cardinality constraints in a program R, let $facts(R) = \{\top\} \cup \{head(r) \mid r \in R, body(r) = \{\top\}\}$. For a (non-ground) cardinality constraint c of the form $l\{a_1 : b_1, \dots, a_n : b_n\}u$, we define $local(c) = l\big(\{a_1\theta \mid \theta : var(b_1) \to \mathbb{U}, b_1\theta \in facts(R)\} \cup \dots \cup \{a_n\theta \mid \theta : var(b_n) \to \mathbb{U}, b_n\theta \in facts(R)\}\big)u$. Here, $var(b_1), \dots, var(b_n)$ are *local* variables of c; $a_1\theta, b_1\theta, \dots, a_n\theta, b_n\theta$ refer to the atoms obtained by applying a substitution θ to $a_1, b_1, \dots, a_n, b_n$. The parts "$l$," "$u$," and "$: b_i$" can optionally be omitted in c, in which case they are identified with "0," "∞," and "$: \top$," respectively.

Since built-in predicates from $\mathbb{I}$ are evaluated during grounding, for an atom a over a predicate from $\mathbb{I}$, we define $local(a) = \top$ if a holds according to its standard interpretation, and $local(a) = \bot$ otherwise. Given an arbitrary yet fixed strict total order $\prec$ on $\mathbb{U}$ that agrees with the standard order $<$ on integers for the common elements in $\mathbb{U} \cap \mathbb{Z}$, the standard interpretation of $=/2$, $\neq/2$, $</2$, and $\leq/2$ is (syntactic) equality, (syntactic) inequality, containment in $\prec$, and equality or containment in $\prec$, respectively.

By letting $local(a) = a$ for an atom a over a predicate from $\mathbb{P}$ and $local(r) = local(a_0) \leftarrow local(a_1), \dots, local(a_m), not\ local(a_{m+1}), \dots, not\ local(a_n).$ for a rule r of the form (20), we are now ready to define the *ground instantiation* of a program R as follows:

$$ground(R) = \{local(r\theta) \mid r \in R, \theta : var(r) \to \mathbb{U}\}.$$

That is, $ground(R)$ consists of the rules obtained by applying all possible substitutions $\theta : var(r) \to \mathbb{U}$ for global variables in rules r of R and by afterwards performing local evaluations on cardinality constraints and atoms over built-in predicates, respectively, in $r\theta$.

It remains to define the answer sets of $ground(R)$. In the following, we provide a simplified version of the definition in [14], which is sufficient for the logic programs considered here. Our simplifications rely on the fact that cardinality constraints occur only positively and contain only positive literals, while negative (body) literals $not\ a$ for atoms a hold w.r.t. all subsets of a Herbrand interpretation not including a.

For a subset A of the Herbrand base $\mathbb{B}$, we define the satisfaction relation on (positive) rule elements by $A \models a$ if $a \in A$, and $A \models l\{a_1, \dots, a_n\}u$ if $l \leq |\{a_1, \dots, a_n\} \cap A| \leq u$. (Note that the head of a choice rule is trivially satisfied since its lower and upper bound are 0 and ∞, respectively.) The body of a rule r of the form (20) is satisfied by A, written $A \models body(r)$, if $A \models a_1, \dots, A \models a_m$ and $A \not\models a_{m+1}, \dots, A \not\models a_n$; r is satisfied by A, denoted by $A \models r$, if $A \models body(r)$ implies $A \models a_0$. We call A a *model* of a program R if $\top \in A$, $\bot \notin A$, and $A \models r$ for every $r \in R$. The *reduct* of R relative to A is defined by

$$\begin{aligned}R^A = \{a \leftarrow body(r). \mid & r \in ground(R), \\ & a \in atom(head(r)) \cap A, A \models body(r)\}\end{aligned}$$

where $atom(a) = \{a\}$ if $a \in \mathbb{B}$, and $atom(c) = \{a_1, \dots, a_n\}$ if $c = l\{a_1, \dots, a_n\}u$. Finally, A is an *answer set* of R if A is a model of $ground(R)$ such that no proper subset of A is a model of R^A.

Before we proceed to prove soundness and completeness, as formulated in Theorem 1, let us link the language of a set $\mathcal{T}$ of flat clauses to the one of the parameterized domain description (B, P, Q) for $\mathcal{T}$ and the logic program $R[i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$, as defined in (1). For the signature $\langle \mathcal{F}_0, \mathcal{F}, \mathcal{P}_0, \mathcal{P} \rangle$ of $\mathcal{T}$, w.l.o.g., we assume that $k/0 \notin \mathcal{F}_0$, $\mathcal{P}_0 \cap \{\top/0, \bot/0\} = \emptyset$, and $\mathcal{P} \cap \{cons/1, func/1, dom/1, arg/2, assign/2, =/2, \neq/2, </2, \leq/2\} = \emptyset$. The signature $\langle \mathbb{F}, \mathbb{P}, \mathbb{I} \rangle$ of $R[i]$ is then given by:

$$\mathbb{F} = \mathcal{F}_0 \cup \mathcal{F} \cup \{1/0, \dots, i/0\}, \tag{21}$$

$$\begin{aligned}\mathbb{P} = \mathcal{P}_0 \cup \mathcal{P} \cup \{&\top/0, \bot/0, cons/1, func/1, \\ &dom/1, arg/2, assign/2\},\end{aligned} \tag{22}$$

$$\mathbb{I} = \{=/2, \neq/2, </2, \leq/2\}. \tag{23}$$

In the following, we assume $ground(R[i])$ to be defined relative to the signature $\langle \mathbb{F}, \mathbb{P}, \mathbb{I} \rangle$.

For example, consider the (non-ground) parameterized domain description (B, P, Q) for the theory in (3):

$$B = \{\, \mathit{cons}(a).\, \}$$

$$P = \left\{ \begin{array}{l}
\mathit{dom}(k). \qquad\qquad \mathit{arg}(k,k). \\
\mathit{func}(\mathit{sko}(X)) \leftarrow \\
\qquad \mathit{dom}(X), 1\{\mathit{arg}(X,k)\}. \\
\{\mathit{assign}(T,k)\} \leftarrow \mathit{cons}(T). \\
\{\mathit{assign}(\mathit{sko}(X),Y)\} \leftarrow \\
\qquad \mathit{dom}(X), \mathit{dom}(Y), \\
\qquad 1\{\mathit{arg}(X,k), \mathit{arg}(Y,k)\}. \\
\leftarrow \mathit{assign}(T,D), \mathit{assign}(T,k), D < k. \\
\quad \{p(X)\} \leftarrow \mathit{dom}(X), 1\{\mathit{arg}(X,k)\}. \\
\{q(X,Y)\} \leftarrow \mathit{dom}(X), \mathit{dom}(Y), \\
\qquad\qquad 1\{\mathit{arg}(X,k), \mathit{arg}(Y,k)\}. \\
\leftarrow \mathit{not}\ p(X), \mathit{assign}(a,X), \mathit{dom}(X), \\
\quad 1\{\mathit{arg}(X,k)\}. \\
\leftarrow q(X,X), \mathit{dom}(X), 1\{\mathit{arg}(X,k)\}. \\
\leftarrow p(X), \mathit{not}\ q(X,Y), \\
\quad \mathit{assign}(\mathit{sko}(X),Y), \mathit{dom}(X), \\
\quad \mathit{dom}(Y), 1\{\mathit{arg}(X,k), \mathit{arg}(Y,k)\}.
\end{array} \right\}$$

$$Q = \left\{ \begin{array}{l}
\leftarrow \mathit{cons}(T), \{\mathit{assign}(T,D) : \mathit{dom}(D)\}0. \\
\leftarrow \mathit{func}(T), \{\mathit{assign}(T,D) : \mathit{dom}(D)\}0.
\end{array} \right\}$$

The ground instantiation of $R[2]$ is equivalent to:[8]

$$\left\{ \begin{array}{l}
\mathit{cons}(a). \\
\mathit{dom}(1). \quad \mathit{arg}(1,1). \qquad \mathit{dom}(2). \quad \mathit{arg}(2,2). \\
\mathit{func}(\mathit{sko}(1)) \leftarrow \mathit{dom}(1), 1\{\mathit{arg}(1,1)\}. \\
\mathit{func}(\mathit{sko}(2)) \leftarrow \mathit{dom}(2), 1\{\mathit{arg}(2,2)\}. \\
\qquad \{\mathit{assign}(a,1)\} \leftarrow \mathit{cons}(a). \\
\qquad \{\mathit{assign}(a,2)\} \leftarrow \mathit{cons}(a). \\
\{\mathit{assign}(\mathit{sko}(1),1)\} \leftarrow \mathit{dom}(1), \mathit{dom}(1), \\
\qquad\qquad\qquad 1\{\mathit{arg}(1,1), \mathit{arg}(1,1)\}. \\
\{\mathit{assign}(\mathit{sko}(1),2)\} \leftarrow \mathit{dom}(1), \mathit{dom}(2), \\
\qquad\qquad\qquad 1\{\mathit{arg}(1,2), \mathit{arg}(2,2)\}. \\
\{\mathit{assign}(\mathit{sko}(2),1)\} \leftarrow \mathit{dom}(2), \mathit{dom}(1), \\
\qquad\qquad\qquad 1\{\mathit{arg}(2,2), \mathit{arg}(1,2)\}. \\
\{\mathit{assign}(\mathit{sko}(2),2)\} \leftarrow \mathit{dom}(2), \mathit{dom}(2), \\
\qquad\qquad\qquad 1\{\mathit{arg}(2,2), \mathit{arg}(2,2)\}. \\
\leftarrow \mathit{assign}(a,1), \mathit{assign}(a,1), \bot. \\
\leftarrow \mathit{assign}(a,1), \mathit{assign}(a,2), \top. \\
\leftarrow \mathit{assign}(a,2), \mathit{assign}(a,2), \bot. \\
\leftarrow \mathit{assign}(\mathit{sko}(1),1), \mathit{assign}(\mathit{sko}(1),1), \bot. \\
\leftarrow \mathit{assign}(\mathit{sko}(1),1), \mathit{assign}(\mathit{sko}(1),2), \top. \\
\leftarrow \mathit{assign}(\mathit{sko}(1),2), \mathit{assign}(\mathit{sko}(1),2), \bot. \\
\leftarrow \mathit{assign}(\mathit{sko}(2),1), \mathit{assign}(\mathit{sko}(2),2), \top. \\
\leftarrow \mathit{assign}(\mathit{sko}(2),2), \mathit{assign}(\mathit{sko}(2),2), \bot. \\
\quad \{p(1)\} \leftarrow \mathit{dom}(1), 1\{\mathit{arg}(1,1)\}. \\
\quad \{p(2)\} \leftarrow \mathit{dom}(2), 1\{\mathit{arg}(2,2)\}. \\
\{q(1,1)\} \leftarrow \mathit{dom}(1), \mathit{dom}(1), \\
\qquad\qquad 1\{\mathit{arg}(1,1), \mathit{arg}(1,1)\}. \\
\{q(1,2)\} \leftarrow \mathit{dom}(1), \mathit{dom}(2), \\
\qquad\qquad 1\{\mathit{arg}(1,2), \mathit{arg}(2,2)\}. \\
\{q(2,1)\} \leftarrow \mathit{dom}(2), \mathit{dom}(1), \\
\qquad\qquad 1\{\mathit{arg}(2,2), \mathit{arg}(1,2)\}. \\
\{q(2,2)\} \leftarrow \mathit{dom}(2), \mathit{dom}(2), \\
\qquad\qquad 1\{\mathit{arg}(2,2), \mathit{arg}(2,2)\}. \\
\leftarrow \mathit{not}\ p(1), \mathit{assign}(a,1), \mathit{dom}(1), 1\{\mathit{arg}(1,1)\}. \\
\leftarrow \mathit{not}\ p(2), \mathit{assign}(a,2), \mathit{dom}(2), 1\{\mathit{arg}(2,2)\}. \\
\leftarrow q(1,1), \mathit{dom}(1), 1\{\mathit{arg}(1,1)\}. \\
\leftarrow q(2,2), \mathit{dom}(2), 1\{\mathit{arg}(2,2)\}. \\
\leftarrow p(1), \mathit{not}\ q(1,1), \mathit{assign}(\mathit{sko}(1),1), \\
\quad \mathit{dom}(1), \mathit{dom}(1), 1\{\mathit{arg}(1,1), \mathit{arg}(1,1)\}. \\
\leftarrow p(1), \mathit{not}\ q(1,2), \mathit{assign}(\mathit{sko}(1),2), \\
\quad \mathit{dom}(1), \mathit{dom}(2), 1\{\mathit{arg}(1,2), \mathit{arg}(2,2)\}. \\
\leftarrow p(2), \mathit{not}\ q(2,1), \mathit{assign}(\mathit{sko}(2),1), \\
\quad \mathit{dom}(2), \mathit{dom}(1), 1\{\mathit{arg}(2,2), \mathit{arg}(1,2)\}. \\
\leftarrow p(2), \mathit{not}\ q(2,2), \mathit{assign}(\mathit{sko}(2),2), \\
\quad \mathit{dom}(2), \mathit{dom}(2), 1\{\mathit{arg}(2,2), \mathit{arg}(2,2)\}. \\
\leftarrow \mathit{cons}(a), \{\mathit{assign}(a,1), \mathit{assign}(a,2)\}0. \\
\leftarrow \mathit{func}(\mathit{sko}(1)), \\
\quad \{\mathit{assign}(\mathit{sko}(1),1), \mathit{assign}(\mathit{sko}(1),2)\}0. \\
\leftarrow \mathit{func}(\mathit{sko}(2)), \\
\quad \{\mathit{assign}(\mathit{sko}(2),1), \mathit{assign}(\mathit{sko}(2),2)\}0.
\end{array} \right\}$$

[8] We omit part of the rules of $\mathit{ground}(R[2])$ containing atoms in the body that do not occur in the head of any rule in $\mathit{ground}(R[2])$. The bodies of such rules are not satisfied by any answer set of $R[2]$. We also omit (redundant) ground rules from $P[k/1]$ that have equivalent counterparts in $P[k/2]$ (and are not produced upon grounding $P[k]$ incrementally w.r.t. an evolving Herbrand universe).

One can check that

$$A = \left\{ \begin{array}{l} \mathit{dom}(1), \mathit{dom}(2), \mathit{arg}(1,1), \mathit{arg}(2,2), \\ \mathit{cons}(a), \mathit{assign}(a,1), \\ \mathit{func}(\mathit{sko}(1)), \mathit{assign}(\mathit{sko}(1),2), \\ \mathit{func}(\mathit{sko}(2)), \mathit{assign}(\mathit{sko}(2),2), \\ p(1), q(1,2) \end{array} \right\}$$

(additionally including the syntactic atom $\top$) is a model of $\mathit{ground}(R[2])$. In particular, the bodies of integrity constraints are not satisfied by A. In fact, the reduct $(R[2])^A$ contains the following rules:

$$\left\{ \begin{array}{l} \mathit{cons}(a). \\ \mathit{dom}(1). \quad \mathit{arg}(1,1). \qquad \mathit{dom}(2). \quad \mathit{arg}(2,2). \\ \mathit{func}(\mathit{sko}(1)) \leftarrow \mathit{dom}(1), 1\{\mathit{arg}(1,1)\}. \\ \mathit{func}(\mathit{sko}(2)) \leftarrow \mathit{dom}(2), 1\{\mathit{arg}(2,2)\}. \\ \qquad \mathit{assign}(a,1) \leftarrow \mathit{cons}(a). \\ \mathit{assign}(\mathit{sko}(1),2) \leftarrow \mathit{dom}(1), \mathit{dom}(2), \\ \qquad\qquad 1\{\mathit{arg}(1,2), \mathit{arg}(2,2)\}. \\ \mathit{assign}(\mathit{sko}(2),2) \leftarrow \mathit{dom}(2), \mathit{dom}(2), \\ \qquad\qquad 1\{\mathit{arg}(2,2), \mathit{arg}(2,2)\}. \\ \quad p(1) \leftarrow \mathit{dom}(1), 1\{\mathit{arg}(1,1)\}. \\ q(1,2) \leftarrow \mathit{dom}(1), \mathit{dom}(2), \\ \qquad 1\{\mathit{arg}(1,2), \mathit{arg}(2,2)\}. \end{array} \right\}$$

Note that the bodies of all rules in $(R[2])^A$ are satisfied by A and that choice rules are turned into strict rules (for head atoms in A). Due to this, it is not difficult to verify that no proper subset of A is a model of $(R[2])^A$, so that A is an answer set of $R[2]$.

Recall that A is the answer set shown in Section 3.3, which corresponds to the model in (4) over domain $\{1,2\}$. In the following, we show that the correspondence between finite models of a theory and answer sets of its associated iASP program holds in general.

A.2. *Proof of Soundness*

The following lemma establishes one direction of Theorem 1 by showing that an answer set of our iASP program yields a (finite) model of a first-order theory.

Lemma 1 (Soundness). *Let $\mathcal{T}$ be a set of flat clauses and (B,P,Q) the parameterized domain description for $\mathcal{T}$. If the logic program $R[i]$, as defined in (1), has an answer set for some positive integer i, then $\mathcal{T}$ has a model over a (finite) domain of size i.*

Proof. Assume that A is an answer set of $R[i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$. As (informally) described in Section 3.3, we extract the domain $\{d \mid \mathit{dom}(d) \in A\}$ along with the following concepts from A:

$$c^I = d \text{ if } \mathit{cons}(c) \in A \text{ and } \mathit{assign}(c,d) \in A,$$

$$\begin{aligned} f/n^I = \{(d_1,\dots,d_n) \mapsto d \mid \\ \mathit{func}(f(d_1,\dots,d_n)) \in A, \\ \mathit{assign}(f(d_1,\dots,d_n),d) \in A\}, \end{aligned}$$

$$p^I = \begin{cases} \top & \text{if } p/0 \in \mathcal{P}_0 \cap A \\ \bot & \text{if } p/0 \in \mathcal{P}_0 \setminus A, \end{cases}$$

$$\begin{aligned} p/n^I = \{(d_1,\dots,d_n) \mid p/n \in \mathcal{P}, \\ p(d_1,\dots,d_n) \in A\}. \end{aligned}$$

Since A is a model of $\mathit{ground}(R[i])$, in view of the rules (7), $\mathit{dom}(k).$ and $\mathit{arg}(k,k).$, in $P[k]$, A contains $\mathit{dom}(1), \mathit{arg}(1,1), \dots, \mathit{dom}(i), \mathit{arg}(i,i)$. Given that no other atoms over $\mathit{dom}/1$ and $\mathit{arg}/2$ occur in the head of any rule in $(R[i])^A$, since no proper subset of A is a model of $(R[i])^A$, we also have that $\mathit{dom}(1), \mathit{arg}(1,1), \dots, \mathit{dom}(i), \mathit{arg}(i,i)$ are all atoms over $\mathit{dom}/1$ and $\mathit{arg}/2$ in A. This shows that $\{d \mid \mathit{dom}(d) \in A\} = \{1,\dots,i\}$. It remains to check that I provides us with a model of $\mathcal{T}$ over domain $\{1,\dots,i\}$.

In an interpretation, every constant in $\mathcal{F}_0$ and every function in $\mathcal{F}$ (along with a tuple of arguments) must be mapped to a unique domain element. Hence, we need to show that the atoms of the form $\mathit{assign}(t,d)$ in A, where t is a constant or a functional term, define a total mapping to domain elements d.

To begin with, for each constant $c/0 \in \mathcal{F}_0$, B contains $\mathit{cons}(c)$. Since A is a model of $\mathit{ground}(R[i])$, this implies that $\mathit{cons}(c) \in A$. As no other atom over $\mathit{cons}/1$ occurs in the head of any rule in $(R[i])^A$, since no proper subset of A is a model of $(R[i])^A$, we also have that $\mathit{cons}(c) \in A$ implies $c/0 \in \mathcal{F}_0$. Furthermore, for each $c/0 \in \mathcal{F}_0$, the integrity constraint (13),

$$\leftarrow \mathit{cons}(T), \{\mathit{assign}(T,D) : \mathit{dom}(D)\}0.,$$

in $Q[k]$ has the following instance in $\mathit{ground}(R[i])$:

$$\leftarrow \mathit{cons}(c), \{\mathit{assign}(c,1), \dots, \mathit{assign}(c,i)\}0.$$

Since A is a model of $\mathit{ground}(R[i])$, this implies that $\{\mathit{assign}(c,1), \dots, \mathit{assign}(c,i)\} \cap A \neq \emptyset$.

Regarding functions $f/n \in \mathcal{F}$, $P[k]$ contains the rule (8),

$$\begin{aligned} &\mathit{func}(f(X_1,\dots,X_n)) \leftarrow \\ &\quad \mathit{dom}(X_1), \dots, \mathit{dom}(X_n), \\ &\quad 1\{\mathit{arg}(X_1,k), \dots, \mathit{arg}(X_n,k)\}. \end{aligned}$$

Since $dom(1), arg(1,1), \dots, dom(i), arg(i,i)$ belong to A, for each substitution of the variables $X_1, \dots, X_n$ with domain elements in $\{1, \dots, i\}$, an instance of the rule is contained in $(R[i])^A$, and there are no further ground instances of (8) in $(R[i])^A$. As a consequence, $func(f(\vec{x})) \in A$ iff $f/n \in \mathcal{F}$ and $\vec{x} \in \{1, \dots, i\}^n$. Similar to constants, the integrity constraint (14),

$$\leftarrow func(T), \{assign(T,D) : dom(D)\}0.,$$

in $Q[k]$ has the following instance in $ground(R[i])$:

$$\begin{aligned}\leftarrow\ & func(f(\vec{x})),\\ & \{assign(f(\vec{x}),1), \dots, assign(f(\vec{x}),i)\}0.\end{aligned}$$

Since A is a model of $ground(R[i])$, this implies that $\{assign(f(\vec{x}),1), \dots, assign(f(\vec{x}),i)\} \cap A \neq \emptyset$.

We have thus established that the atoms of the form $assign(t,d)$ in A are such that each constant or functional term t is mapped to some domain element d. It remains to show that the mapping is unique. To this end, we note that only the ground instances of the rules (6) and (10) in $P[k]$,

$$\{assign(T,k)\} \leftarrow cons(T).$$

$$\begin{aligned}\{assign(f(X_1, \dots, X_n), Y)\} \leftarrow\ & \\ & dom(X_1), \dots, dom(X_n), dom(Y),\\ & 1\{arg(X_1,k), \dots, arg(X_n,k), arg(Y,k)\}.,\end{aligned}$$

that contribute to $(R[i])^A$ contain atoms over $assign/2$ in the head. Since no proper subset of A is a model of $(R[i])^A$, this implies that every atom of the form $assign(t,d)$ in A contains a constant or functional term t and a domain element d as its arguments. In view of the integrity constraint (12),

$$\leftarrow assign(T,D), assign(T,k), D < k.,$$

in $P[k]$, since A is a model of $ground(R[i])$, for every constant or functional term t, we have that $|\{assign(t,1), \dots, assign(t,i)\} \cap A| \leq 1$. As shown above, $\{assign(t,1), \dots, assign(t,i)\} \cap A \neq \emptyset$ also holds. That is, c^I and f/n^I, as defined above, provide a total mapping of constants $c/0 \in \mathcal{F}_0$ and functional terms constructed from $f/n \in \mathcal{F}$ to domain elements.

To check that I is an interpretation over domain $\{1, \dots, i\}$, we note that $P[k]$ contains a rule (15),

$$\begin{aligned}\{p(X_1, \dots, X_n)\} \leftarrow\ & \\ & dom(X_1), \dots, dom(X_n),\\ & 1\{arg(X_1,k), \dots, arg(X_n,k)\}.,\end{aligned}$$

for each predicate $p/n \in \mathcal{P}$. For every instance $r \in ground(R[i])$, derived from (15) by replacing k with j for $1 \leq j \leq i$, if $A \models body(r)$, the fact that $dom(1)$, $\dots$, $dom(i)$ are all atoms over $dom/1$ in A implies that $X_1, \dots, X_n$ are substituted with domain elements. As other rules in $ground(R[i])$ do not contain any atom over p/n in the head, since no proper subset of A is a model of $(R[i])^A$, this implies that all atoms $p(\vec{x})$ over p/n in A are such that $\vec{x} \in \{1, \dots, i\}^n$. That is, I is indeed an interpretation over domain $\{1, \dots, i\}$.

It remains to show that I is a model of $\mathcal{T}$. To this end, suppose that I is not a model of $\mathcal{T}$. Then, there is some clause C in $\mathcal{T}$ that is not satisfied by I. Let $L_1, \dots, L_m$ be the literals and $X_1, \dots, X_n$ the variables of C. Since C is not satisfied by I, there is a variable assignment $\theta = \{X_1 \mapsto d_{X_1}, \dots, X_n \mapsto d_{X_n}\}$, where $d_{X_1}, \dots, d_{X_n}$ are contained in $\{1, \dots, i\}$, under which none of $L_1, \dots, L_m$ is satisfied by I. If $\theta \neq \emptyset$ (C is not propositional), let $j = \max\{d_{X_1}, \dots, d_{X_n}\}$, and note that the following instance of the integrity constraint (17) belongs to $ground(R[i])$:

$$\begin{aligned}\leftarrow \overline{L}_1\theta, \dots, \overline{L}_m\theta, dom(d_{X_1}), \dots, dom(d_{X_n}),\\ 1\{arg(d_{X_1},j), \dots, arg(d_{X_n},j)\}.\end{aligned}$$

Since $dom(1), \dots, dom(i)$ and $arg(j,j)$ belong to A, we have that $A \models dom(d_{X_1}), \dots, A \models dom(d_{X_n})$ and $A \models 1\{arg(d_{X_1},j), \dots, arg(d_{X_n},j)\}$. On the other hand, if $\theta = \emptyset$ (C is propositional), the following integrity constraint (19) belongs to $ground(R[i])$:

$$\leftarrow \overline{L}_1\theta, \dots, \overline{L}_m\theta.$$

Regardless of whether C is propositional or not, one of the following is the case for every $1 \leq j \leq m$:[9]

- If $L_j = p$ for some $p/0 \in \mathcal{P}_0$, then $\overline{L}_j\theta = not\ p$, $p^I = \bot$, and $A \not\models p$.
- If $L_j = \neg p$ for some $p/0 \in \mathcal{P}_0$, then $\overline{L}_j\theta = p$, $p^I = \top$, and $A \models p$.
- If $L_j = p(X_{1_j}, \dots, X_{n_j})$ for some $p/n_j \in \mathcal{P}$, then $\overline{L}_j\theta = not\ p(d_{X_{1_j}}, \dots, d_{X_{n_j}})$, $(d_{X_{1_j}}, \dots, d_{X_{n_j}}) \notin (p/n_j)^I$, and $A \not\models p(d_{X_{1_j}}, \dots, d_{X_{n_j}})$.
- If $L_j = \neg p(X_{1_j}, \dots, X_{n_j})$ for some $p/n_j \in \mathcal{P}$, then $\overline{L}_j\theta = p(d_{X_{1_j}}, \dots, d_{X_{n_j}})$, $(d_{X_{1_j}}, \dots, d_{X_{n_j}}) \in (p/n_j)^I$, and $A \models p(d_{X_{1_j}}, \dots, d_{X_{n_j}})$.
- If $L_j = (X = c)$ for some $c/0 \in \mathcal{F}_0$, then $\overline{L}_j\theta = not\ assign(c, d_X)$, $c^I \neq d_X$, and $A \not\models assign(c, d_X)$.

[9]Recall the encoding $\overline{L}$ of the complement of a (classical or equality) literal L from Section 3.2.

- If $L_j = (X \neq c)$ for some $c/0 \in \mathcal{F}_0$, then $\overline{L}_j\theta = \mathit{assign}(c, d_X)$, $c^I = d_X$, and $A \models \mathit{assign}(c, d_X)$.
- If $L_j = (X = f(X_{1_j}, \dots, X_{n_j}))$ for some $f/n_j \in \mathcal{F}$, then $\overline{L}_j\theta = \mathit{not}\ \mathit{assign}(f(d_{X_{1_j}}, \dots, d_{X_{n_j}}), d_X)$, $((d_{X_{1_j}}, \dots, d_{X_{n_j}}) \mapsto d_X) \notin (f/n_j)^I$, and $A \not\models \mathit{assign}(f(d_{X_{1_j}}, \dots, d_{X_{n_j}}), d_X)$.
- If $L_j = (X \neq f(X_{1_j}, \dots, X_{n_j}))$ for some $f/n_j \in \mathcal{F}$, then $\overline{L}_j\theta = \mathit{assign}(f(d_{X_{1_j}}, \dots, d_{X_{n_j}}), d_X)$, $((d_{X_{1_j}}, \dots, d_{X_{n_j}}) \mapsto d_X) \in (f/n_j)^I$, and $A \models \mathit{assign}(f(d_{X_{1_j}}, \dots, d_{X_{n_j}}), d_X)$.
- If $L_j = (X = Y)$ for variables X and Y, then $\overline{L}_j\theta = (d_X \neq d_Y)$, $d_X \neq d_Y$, and $\mathit{local}(d_X \neq d_Y) = \top$.
- If $L_j = (X \neq Y)$ for variables X and Y, then $\overline{L}_j\theta = (d_X = d_Y)$, $d_X = d_Y$, and $\mathit{local}(d_X = d_Y) = \top$.

The above cases show that $A \models a$ if $\overline{L}_j\theta = a$, $A \not\models a$ if $\overline{L}_j\theta = \mathit{not}\ a$, or $\mathit{local}(\overline{L}_j\theta) = \top$ holds. That is, $A \models \mathit{body}(r)$ holds for some integrity constraint $r \in \mathit{ground}(R[i])$, which contradicts the fact that A is a model of $\mathit{ground}(R[i])$. From this, we conclude that I must be a model of $\mathcal{T}$. $\square$

A.3. *Proof of Completeness*

Our second lemma shows that the other direction of Theorem 1, i.e., that a finite model of a first-order theory coincides with an answer set, holds as well.

Lemma 2 (Completeness). *Let $\mathcal{T}$ be a set of flat clauses and (B, P, Q) the parameterized domain description for $\mathcal{T}$. If $\mathcal{T}$ has a model over a finite domain of size i, then the logic program $R[i]$, as defined in (1), has an answer set.*

Proof. Assume that an interpretation I over finite domain $\{1, \dots, i\}$ is a model of $\mathcal{T}$. We will show that the following set A of atoms is an answer set of $R[i]$:

$$
\begin{aligned}
A = \{&\top\} \\
&\cup \{\mathit{dom}(j), \mathit{arg}(j,j) \mid 1 \leq j \leq i\} \\
&\cup \{\mathit{cons}(c) \mid c/0 \in \mathcal{F}_0\} \\
&\cup \{\mathit{assign}(c,d) \mid c/0 \in \mathcal{F}_0, c^I = d\} \\
&\cup \{\mathit{func}(f(\vec{x})) \mid f/n \in \mathcal{F}, \vec{x} \in \{1, \dots, i\}^n\} \\
&\cup \{\mathit{assign}(f(\vec{x}), d) \mid f/n \in \mathcal{F}, (\vec{x} \mapsto d) \in f/n^I\} \\
&\cup \{p \mid p/0 \in \mathcal{P}_0, p^I = \top\} \\
&\cup \{p(\vec{x}) \mid p/n \in \mathcal{P}, \vec{x} \in p/n^I\}.
\end{aligned}
$$

To begin with, we note the instances in $\mathit{ground}(R[i])$ of the rules (7), $\mathit{dom}(k).$ and $\mathit{arg}(k,k).$, in $P[k]$, obtained by replacing k with j for $1 \leq j \leq i$, are satisfied by A. The same applies to every rule $\mathit{cons}(c).$ in B and $\mathit{ground}(R[i])$. Since $\mathit{body}(r) = \{\top\}$ for such rules $r \in (R[i])^A$, every model of $(R[i])^A$ must contain $\mathit{dom}(1), \mathit{arg}(1,1), \dots, \mathit{dom}(i), \mathit{arg}(i,i)$ as well as $\mathit{cons}(c)$ for all $c/0 \in \mathcal{F}_0$. In particular, these atoms belong to each subset of A that is a model of $(R[i])^A$.

Regarding functions $f/n \in \mathcal{F}$, $P[k]$ contains the rule (8),

$$
\begin{aligned}
&\mathit{func}(f(X_1, \dots, X_n)) \leftarrow \\
&\quad \mathit{dom}(X_1), \dots, \mathit{dom}(X_n), \\
&\quad 1\{\mathit{arg}(X_1, k), \dots, \mathit{arg}(X_n, k)\}.
\end{aligned}
$$

By its definition, A satisfies all instances of this rule in $\mathit{ground}(R[i])$. (A includes $\mathit{func}(f(\vec{x}))$ for all $\vec{x} \in \{1, \dots, i\}^n$.) Moreover, since $\mathit{dom}(1), \mathit{arg}(1,1), \dots, \mathit{dom}(i), \mathit{arg}(i,i)$ belong to every model of $(R[i])^A$, every model of $(R[i])^A$ must contain $\mathit{func}(f(\vec{x}))$ for all $\vec{x} \in \{1, \dots, i\}^n$. In particular, these atoms belong to each subset of A that is a model of $(R[i])^A$.

Atoms over $\mathit{assign}/2$ occur in the heads of the rules (6) and (10) in $P[k]$,

$$
\{\mathit{assign}(T, k)\} \leftarrow \mathit{cons}(T).
$$

$$
\begin{aligned}
&\{\mathit{assign}(f(X_1, \dots, X_n), Y)\} \leftarrow \\
&\quad \mathit{dom}(X_1), \dots, \mathit{dom}(X_n), \mathit{dom}(Y), \\
&\quad 1\{\mathit{arg}(X_1, k), \dots, \mathit{arg}(X_n, k), \mathit{arg}(Y, k)\}.,
\end{aligned}
$$

whose instances $r \in \mathit{ground}(R[i])$ are satisfied by A because $A \models \mathit{head}(r)$ holds trivially. However, for every constant $c/0 \in \mathcal{F}_0$ and $c^I = d$, since $\mathit{cons}(c) \in A$ and $\mathit{assign}(c,d) \in A$, the following rule is contained in $(R[i])^A$:

$$
\mathit{assign}(c, d) \leftarrow \mathit{cons}(c).
$$

As shown above, $\mathit{cons}(c)$ belongs to every model of $(R[i])^A$, so that $\mathit{assign}(c,d)$ must be included as well. Likewise, for every function $f/n \in \mathcal{F}$, $((d_1, \dots, d_n) \mapsto d) \in f/n^I$, and $j = \max\{d_1, \dots, d_n, d\}$, the following rule is contained in $(R[i])^A$:

$$
\begin{aligned}
&\mathit{assign}(f(d_1, \dots, d_n), d) \leftarrow \\
&\quad \mathit{dom}(d_1), \dots, \mathit{dom}(d_n), \mathit{dom}(d), \\
&\quad 1\{\mathit{arg}(d_1, j), \dots, \mathit{arg}(d_n, j), \mathit{arg}(d, j)\}.
\end{aligned}
$$

Given that $\mathit{dom}(1), \mathit{arg}(1,1), \dots, \mathit{dom}(i), \mathit{arg}(i,i)$ belong to every model of $(R[i])^A$, we have that $\mathit{assign}(f(d_1, \dots, d_n), d)$ must be included as well. We have thus shown that all atoms over $\mathit{assign}/2$ in A also belong to each subset of A that is a model of $(R[i])^A$.

Regarding the predicates from $\mathcal{P}_0 \cup \mathcal{P}$, B contains a rule $\{p\}$. for each $p/0 \in \mathcal{P}_0$, and $P[k]$ contains a rule (15),

$$\begin{array}{l}\{p(X_1,\dots,X_n)\} \leftarrow \\ \quad dom(X_1),\dots,dom(X_n), \\ \quad 1\{arg(X_1,k),\dots,arg(X_n,k)\}.,\end{array}$$

for each $p/n \in \mathcal{P}$. Their instances $r \in ground(R[i])$ are satisfied by A because $A \models head(r)$ holds trivially. However, for every $p/0 \in \mathcal{P}_0$ such that $p^I = \top$, $(R[i])^A$ contains the rule p., so that every model of $(R[i])^A$ must include p. Furthermore, for every $p/n \in \mathcal{P}$, $(d_1,\dots,d_n) \in p/n^I$, and $j = \max\{d_1,\dots,d_n\}$, the following rule is contained in $(R[i])^A$:

$$\begin{array}{l}p(d_1,\dots,d_n) \leftarrow \\ \quad dom(d_1),\dots,dom(d_n), \\ \quad 1\{arg(d_1,j),\dots,arg(d_n,j)\}.\end{array}$$

Given that $dom(1), arg(1,1),\dots,dom(i), arg(i,i)$ belong to every model of $(R[i])^A$, we have that $p(d_1,\dots,d_n)$ must be included as well. We have thus shown that all atoms over predicates from $\mathcal{P}_0 \cup \mathcal{P}$ in A also belong to each subset of A that is a model of $(R[i])^A$. Since other atoms of A cannot be dropped either in a model of $(R[i])^A$ (shown above), we conclude that no proper subset of A is a model of $(R[i])^A$.

It remains to show that A satisfies all integrity constraints in $ground(R[i])$. To this end, we note that the mapping of constants $c/0 \in \mathcal{F}_0$ and functional terms constructed from $f/n \in \mathcal{F}$ to domain elements, given by I, is total and unique. Hence, A contains exactly one atom of the form $assign(c,d)$ for each $c/0 \in \mathcal{F}_0$, and exactly one atom of the form $assign(f(\vec{x}),d)$ for each $f/n \in \mathcal{F}$ and $\vec{x} \in \{1,\dots,i\}^n$, where $d \in \{1,\dots,i\}$. In view of uniqueness, A satisfies the instances in $ground(R[i])$ of the integrity constraint (12),

$$\leftarrow assign(T,D), assign(T,k), D < k.,$$

in $P[k]$. Moreover, in view of totality, the instances in $ground(R[i])$ of the integrity constraints (13) and (14),

$$\leftarrow cons(T), \{assign(T,D) : dom(D)\}0.$$

$$\leftarrow func(T), \{assign(T,D) : dom(D)\}0.,$$

in $Q[k]$ are satisfied by A.

Finally, given that I is a model of $\mathcal{T}$, for every clause, containing literals $L_1,\dots,L_m$ and variables $X_1,\dots,X_n$, in $\mathcal{T}$ and every variable assignment $\theta = \{X_1 \mapsto d_{X_1},\dots,X_n \mapsto d_{X_n}\}$, some L_j for $1 \leq j \leq m$ is satisfied by I under θ. Then, one of the following cases applies to L_j:

- If $L_j = p$ for some $p/0 \in \mathcal{P}_0$, then $\overline{L}_j\theta = not\ p$, $p^I = \top$, and $A \models p$.
- If $L_j = \neg p$ for some $p/0 \in \mathcal{P}_0$, then $\overline{L}_j\theta = p$, $p^I = \bot$, and $A \not\models p$.
- If $L_j = p(X_{1_j},\dots,X_{n_j})$ for some $p/n_j \in \mathcal{P}$, then $\overline{L}_j\theta = not\ p(d_{X_{1_j}},\dots,d_{X_{n_j}})$, $(d_{X_{1_j}},\dots,d_{X_{n_j}}) \in (p/n_j)^I$, and $A \models p(d_{X_{1_j}},\dots,d_{X_{n_j}})$.
- If $L_j = \neg p(X_{1_j},\dots,X_{n_j})$ for some $p/n_j \in \mathcal{P}$, then $\overline{L}_j\theta = p(d_{X_{1_j}},\dots,d_{X_{n_j}})$, $(d_{X_{1_j}},\dots,d_{X_{n_j}}) \notin (p/n_j)^I$, and $A \not\models p(d_{X_{1_j}},\dots,d_{X_{n_j}})$.
- If $L_j = (X = c)$ for some $c/0 \in \mathcal{F}_0$, then $\overline{L}_j\theta = not\ assign(c, d_X)$, $c^I = d_X$, and $A \models assign(c, d_X)$.
- If $L_j = (X \neq c)$ for some $c/0 \in \mathcal{F}_0$, then $\overline{L}_j\theta = assign(c, d_X)$, $c^I \neq d_X$, and $A \not\models assign(c, d_X)$.
- If $L_j = (X = f(X_{1_j},\dots,X_{n_j}))$ for some $f/n_j \in \mathcal{F}$, then $\overline{L}_j\theta = not\ assign(f(d_{X_{1_j}},\dots,d_{X_{n_j}}), d_X)$, $((d_{X_{1_j}},\dots,d_{X_{n_j}}) \mapsto d_X) \in (f/n_j)^I$, and $A \models assign(f(d_{X_{1_j}},\dots,d_{X_{n_j}}), d_X)$.
- If $L_j = (X \neq f(X_{1_j},\dots,X_{n_j}))$ for some $f/n_j \in \mathcal{F}$, then $\overline{L}_j\theta = assign(f(d_{X_{1_j}},\dots,d_{X_{n_j}}), d_X)$, $((d_{X_{1_j}},\dots,d_{X_{n_j}}) \mapsto d_X) \notin (f/n_j)^I$, and $A \not\models assign(f(d_{X_{1_j}},\dots,d_{X_{n_j}}), d_X)$.
- If $L_j = (X = Y)$ for variables X and Y, then $\overline{L}_j\theta = (d_X \neq d_Y)$, $d_X = d_Y$, and $local(d_X \neq d_Y) = \bot$.
- If $L_j = (X \neq Y)$ for variables X and Y, then $\overline{L}_j\theta = (d_X = d_Y)$, $d_X \neq d_Y$, and $local(d_X = d_Y) = \bot$.

The above cases show that $A \models a$ if $\overline{L}_j\theta = not\ a$, $A \not\models a$ if $\overline{L}_j\theta = a$, or $local(\overline{L}_j\theta) = \bot$ holds. That is, $A \not\models body(r)$ for every instance $r \in ground(R[i])$ of the integrity constraints (17) and (19),

$$\begin{array}{l}\leftarrow \overline{L}_1,\dots,\overline{L}_m, dom(X_1),\dots,dom(X_n), \\ \qquad 1\{arg(X_1,k),\dots,arg(X_n,k)\}.\end{array}$$

$$\leftarrow \overline{L}_1,\dots,\overline{L}_m.,$$

in $P[k]$ or B (depending on whether a clause is propositional or not), respectively. As we have now considered all rules in $R[i]$ and verified that A is a model of $ground(R[i])$, along with the fact that no proper subset of A is a model of $(R[i])^A$ (shown above), we conclude that A is an answer set of $R[i]$. $\square$

A.4. Soundness and Completeness

In view of Lemma 1, showing the soundness of our iASP encoding of FMC, and Lemma 2, showing its completeness, we immediately derive the desired soundness and completeness result.

Theorem 1 (Soundness & Completeness)**.** *Let $\mathcal{T}$ be a set of flat clauses and (B, P, Q) the parameterized domain description for $\mathcal{T}$. Then, the logic program $R[i]$, as defined in (1), has an answer set for some positive integer i iff $\mathcal{T}$ has a model over a finite domain of size i.*

Proof. The result is immediate by Lemma 1 and 2. □

Note that Theorem 1 still applies when using symmetry breaking, as described in Section 3.4. To see this, assume that the set of constants in a theory $\mathcal{T}$ is $\{c_1, \dots, c_n\}$, and let I be a model of $\mathcal{T}$ over finite domain $\{1, \dots, i\}$. Then, define a permutation π on domain elements by

$$\pi(d) = \begin{cases} |\{c_j^I \mid 1 \leq j < \min\{m \mid c_m^I = d\}\}| + 1 \\ \quad \text{if } d \in \{c_1^I, \dots, c_n^I\} \\ |\{c_j^I \in \{c_1^I, \dots, c_n^I\} \mid d < c_j^I\}| + d \\ \quad \text{if } d \in \{1, \dots, i\} \setminus \{c_1^I, \dots, c_n^I\}. \end{cases}$$

The permutation π rearranges domain elements such that the sequence $(\pi(c_1^I), \dots, \pi(c_n^I))$ starts with 1 and consecutively picks domain elements on their first use. For example, given the mapping $c_1^I = 3, c_2^I = 1, c_3^I = 3$ of constants c_1, c_2, c_3 and domain $\{1, 2, 3, 4\}$, we get $\pi(1) = 2, \pi(2) = 3, \pi(3) = 1, \pi(4) = 4$ and $(\pi(c_1^I), \pi(c_2^I), \pi(c_3^I)) = (1, 2, 1)$. This mapping satisfies the conditions encoded by the rules for symmetry breaking provided in Section 3.4. Along with the fact that a model of a theory $\mathcal{T}$ stays a model of $\mathcal{T}$ when permuting domain elements (via π), we conclude that Lemma 2 (Completeness) and Theorem 1 remain valid also in the presence of symmetry breaking rules.

B. *fmc2iasp* at Work

We below provide an example input in TPTP format (Section B.1). Theory-independent rules (Section B.2) as well as the ones generated by *fmc2iasp* (Section B.3) can then be used as inputs to compute finite models with *iClingo*.

B.1. Input Theory

The input theory (2), written in TPTP format, is as follows:

```
fof(1, axiom, p(a)).
fof(2, axiom, ! [X]: (~q(X,X)) ).
fof(3, axiom, ! [X]:
        (p(X) => (? [Y]: q(X,Y))) ).
```

B.2. Theory-Independent iASP Program

The theory-independent program part with symmetry breaking (cf. Section 3.4), in the input language of *iClingo*, is as follows:

```
#cumulative k.

dom(k).
arg(k,k).

{ assign(T,k) } :- cons(T),
        order(T,O), k <= O.

:- assign(T,D), assign(T,k), D < k.

assigned(T,k) :- order(S,O),
        order(T,O+1), assign(S,k).
assigned(T,k) :- order(S,O),
        order(T,O+1), assigned(S,k).

:- cons(T), assign(T,k),
   not assigned(T,k-1), 1 < k.

#volatile k.

:- cons(T), { assign(T,D):dom(D) } 0.
:- func(T), { assign(T,D):dom(D) } 0.
```

B.3. Theory-Dependent iASP Program

The rules generated by *fmc2iasp* for the flat clauses in (3) are as follows:

```
#cumulative k.

% functions
func(sko(X0)) :- dom(X0),
        1 { arg(X0,k) }.
{ assign(sko(X0),Y) } :- dom(X0;Y),
        1 { arg(X0;Y,k) }.
```

```
% predicates
{ p(X0) } :- dom(X0),
        1 { arg(X0,k) }.
{ q(X0,X1) } :- dom(X0;X1),
        1 { arg(X0;X1,k) }.

% flat clauses
:- not p(X0), assign(a,X0),
   dom(X0), 1 { arg(X0,k) }.
:- q(X0,X0),
   dom(X0), 1 { arg(X0,k) }.
:- p(X0), not q(X0,X1),
   assign(sko(X0),X1),
   dom(X0;X1), 1 { arg(X0;X1,k) }.

#base.

cons(a).
order(a,1).

#hide.
#show assign/2.
#show p/1.
#show q/2.
```

In order to compute a finite model of (2), we can use this program concatenated with the rules from Section B.2, as it is described in Section 4.