Mathematical Foundations of Answer Set Programming

PAOLO FERRARIS AND VLADIMIR LIFSCHITZ

1 Introduction

Answer set programming (ASP) is a form of declarative logic programming oriented towards difficult combinatorial search problems. ASP has been applied, for instance, to developing a decision support system for the Space Shuttle [Nogueira *et al.*, 2001] and to graph-theoretic problems arising in zoology and linguistics [Brooks *et al.*, 2005]. This paper is about the design of provably correct ASP programs and about the mathematical theory it is based on. Our description of ASP may be useful as a complement to the monograph [Baral, 2003] and to the manuals on the software systems SMODELS[1] and DLV[2].

Syntactically, ASP programs look like Prolog programs, but the computational mechanisms used in ASP are different: they are based on the ideas that have led to the creation of fast satisfiability solvers for propositional logic.

ASP has emerged from interaction between two lines of research—on the semantics of negation in logic programming [Gelfond and Lifschitz, 1988] and on applications of satisfiability solvers to search problems [Kautz and Selman, 1992]. It was identified as a new programming paradigm in [Lifschitz, 1999, Marek and Truszczyński, 1999, Niemelä, 1999].

The main definition of ASP, discussed in Section 2 below, tells us under what conditions a model of a propositional formula F (that is, a truth assignment satisfying F) is called "stable." The idea of ASP is to represent the search problem we are interested in as the problem of finding a stable model of a formula, and then find a solution using an *answer set solver*—a system for generating stable models, such as the systems SMODELS and DLV mentioned above. Information on these and other available answer set solvers can be obtained from the library of logic programming systems

[1]http://www.tcs.hut.fi/Software/smodels/lparse.ps .
[2]http://www.dbai.tuwien.ac.at/proj/dlv/man/ .

maintained at the University of Koblenz and Landau.[3] Such systems are not applicable, at least directly, to arbitrary propositional formulas; what they expect as input is a conjunction, or list, of formulas of several special types that are particularly useful from the programmer's perspective; these formulas are assumed to be written in "logic programming notation"—as rules, often similar to Prolog rules.

Properties of stable models are discussed here in Section 2, and designing provably correct ASP programs is the topic of Section 3. Proofs of theorems are relegated to Section 4. These sections are followed by Appendix A, which provides the necessary background information about propositional logic, and Appendix B, which reviews the original 1988 definition of a stable model and relates it to the definition used in the main part of this paper.

As we will see, two equivalent formulas do not necessarily have the same stable models. For instance, the only stable model of the implication

(1) $\neg p \rightarrow q$

is the truth assignment that makes p false and q true; the only stable model of its contrapositive

(2) $\neg q \rightarrow p$

makes p true and q false. (In ASP, it is customary to identify a truth assignment with the set of atoms that get the value $\mathtt{t}$. So we can say that the only stable model of (1) is $\{q\}$, and the only stable model of (2) is $\{p\}$.) The fact that there is an essential difference between (1) and (2) will not surprise a Prolog programmer: in the world of programs with negation as failure, placing q in the head of a rule and the negation of p in the body is very different than placing p in the head and the negation of q in the body. A logician, on the other hand, will note that formulas (1) and (2) are equivalent to each other classically, but not intuitionistically. We will see, in fact, that intuitionistically equivalent formulas always have the same stable models. Hence intuitionistically equivalent transformations play an important role in ASP.

A conjunction $F \wedge G$ may have a stable model that is not a stable model of F. For instance, the formula

(3) $(\neg p \rightarrow q) \wedge p$

has one stable model $\{p\}$, which is not a stable model of its first conjunctive term (1). Thus appending an additional conjunctive term to a formula may give it a new stable model. In this sense, the concept of a

[3] `http://www.uni-koblenz.de/ag-ki/LP/lp_systems.html` .

stable model is nonmonotonic. Early work on stable models was an outgrowth of research on formal nonmonotonic reasoning [McCarthy, 1980, McDermott and Doyle, 1980, Reiter, 1980], and, more specifically, of the study in [Gelfond, 1987] of the relationship between autoepistemic logic [Moore, 1985] and the semantics of negation in logic programming.

2 Stable Models

After defining the concept of a stable model in Section 2.1, we apply this definition to three special cases that are often encountered in the practice of answer set programming: Horn formulas, choice formulas and constraints (Sections 2.2–2.4). Section 2.5 is a brief introduction to the use of the answer set solver SMODELS. Then we discuss two mathematical ideas that play an important role in the design of provably correct ASP programs: strong equivalence (Section 2.6) and splitting (Section 2.7).

2.1 Definition and Examples

Recall that we identify truth-valued functions on the set of atoms with subsets of that set (Section A.1).

The *reduct* F^X of a formula F relative to a set X of atoms is the formula obtained from F by replacing each maximal subformula that is not satisfied by X with $\bot$ [Ferraris, 2005].[4] We say that X is a *stable model* (or an *answer set*) of F if X is minimal among the sets satisfying F^X. The minimality of X is understood here in the sense of set inclusion: no proper subset of X satisfies F^X.

Clearly, every set that is a stable model of F according to this definition is a model of F. Indeed, if X does not satisfy F then F^X is $\bot$.

According to the definition, we can verify that X is a stable model of F as follows:

(i) mark in F the maximal subformulas that are not satisfied by X;

(ii) replace each of these subformulas with $\bot$ (after that, equivalent transformations of classical propositional logic can be used to simplify the result);

(iii) check that the resulting formula is satisfied by X;

(iv) check that it is not satisfied by any proper subset of X.

For instance, to check that $\{q\}$ is an answer set of (1), we do the following:

[4] This is somewhat different from the definition of the reduct proposed in [Gelfond and Lifschitz, 1988]. The relationship between the two definitions is discussed in Appendix B.

(i) mark the only subformula of (1) that is not satisfied by $\{q\}$:

$$\neg\underline{p} \to q;$$

(ii) replace that subformula with $\bot$:

$$\neg\bot \to q;$$

simplify:

$$q;$$

(iii) check that the last formula is satisfied by $\{q\}$;

(iv) check that it is not satisfied by $\emptyset$.

The other two models of (1)

$$\{p\},\ \{p, q\}$$

are not stable. We check, for instance, that $\{p\}$ is not stable as follows. First we mark the subformulas of (1) that are not satisfied by $\{p\}$:

$$\underline{\neg p} \to \underline{q}.$$

After replacing these subformulas with $\bot$, we get $\bot \to \bot$, or $\top$. Clearly $\{p\}$ is not minimal among the sets satisfying this reduct: the empty set satisfies it as well.

Alternatively, we can conclude that $\{p\}$ and $\{p, q\}$ cannot be stable models of (1) from the general property of stable models stated below. An atom A is a *head atom* of a formula F if at least one occurrence of A in F is strictly positive (see Section A.1 for the definition). This terminology is related to the fact that in logic programming it is customary to write implications $F \to G$ as "rules" $G \leftarrow F$ and to call G the "head"of the rule and H its "body." Clearly, every head atom of a rule $G \leftarrow F$ occurs in its head G.

THEOREM 1 ([Ferraris, 2005]). *Any stable model of F is a subset of the set of head atoms of F.*

Since the only head atom of (1) is q, Theorem 1 shows that stable models of that formula can be found only among the subsets of $\{q\}$.

Many formulas have several stable models. For instance, the conjunction of (1) and (2)

(4) $\quad (\neg p \to q) \wedge (\neg q \to p)$

has two stable models $\{p\}$ and $\{q\}$, and so does the formula $p \vee q$. On the other hand, $\neg\neg p$ and $\neg p \rightarrow p$ are examples of satisfiable formulas that have no stable models.

It is easy to see that for any X

$$\bot^X = \bot;$$

$$A^X = \begin{cases} A, & \text{if } X \models A, \\ \bot, & \text{otherwise} \end{cases}$$

(A is an atom);

$$(F \odot G)^X = \begin{cases} F^X \odot G^X, & \text{if } X \models F \odot G, \\ \bot, & \text{otherwise} \end{cases}$$

($\odot$ is $\wedge$, $\vee$ or $\rightarrow$).

These equalities provide an alternative, recursive definition of the reduct. There is no clause for negation here, because we treat it as an abbreviation (Section A.1). It is easy to check that

$$(\neg F)^X = \begin{cases} \bot, & \text{if } X \models F, \\ \top, & \text{otherwise.} \end{cases}$$

2.2 Horn Formulas

A *Horn formula* is a conjunction of several (0 or more) implications of the form $F \rightarrow A$, where F is a conjunction of several (0 or more) atoms, and A is an atom. The theorem below shows that any Horn formula has exactly one stable model.

For any Horn formula F, the intersection of all models of F is a model of F also; it is called the *minimal model* of F.

THEOREM 2. *For any Horn formula F, the minimal model of F is the only stable model of F.*

For instance, the formula

(5) $\quad p \wedge (p \rightarrow q) \wedge (q \wedge r \rightarrow s)$

has one stable model—its minimal model $\{p, q\}$. The only model of the empty conjunction $\top$ is the empty set.

2.3 Choice Formulas

From formulas with a unique stable model discussed above we turn now to an example of formulas that have exponentially many stable models.

For any finite set Z of atoms, by Z^c we denote the formula

$$\bigwedge_{A \in Z} (A \vee \neg A).$$

PROPOSITION 3. *For any finite set Z of atoms, a set X of atoms is a stable model of Z^c iff $X \subseteq Z$.*

Proof. For any subset X of Z, the reduct of Z^c relative to X is

$$\bigwedge_{A \in X} (A \vee \bot) \wedge \bigwedge_{A \in Z \setminus X} (\bot \vee \neg\bot),$$

which is equivalent to $\bigwedge_{A \in X} A$. This formula is satisfied by X, but is not satisfied by any proper subset of X. We have proved that if X is a subset of Z then X is a stable model of Z^c. The converse is immediate from Theorem 1. ■

For instance, $\{p, q\}^c$ is

$$(p \vee \neg p) \wedge (q \vee \neg q).$$

This formula has 4 answer sets—arbitrary subsets of $\{p, q\}$. Generally, if Z consists of n atoms then Z^c has 2^n stable models. To form one of them, we choose for every element of Z arbitrarily whether to include it in the model. We will call formulas of the form Z^c *choice formulas.* (The superscript ^{c} is used in this notation because it is the first letter of the word "choice.")

2.4 Constraints

The art of answer set programming is based on the possibility of representing the collection of sets that we are interested in as the collection of stable models of a formula. This is often achieved by conjoining a choice formula, which provides an approximation from above for the collection of sets that we want to describe, with formulas of a special syntactic form, called constraints, that eliminate the unsuitable stable models.

As discussed in the introduction, conjoining a formula F with another formula generally affects the collection of stable models of F nonmonotonically. But this does not happen if the second conjunctive term begins with negation. According to the proposition below, conjoining a formula F

with $\neg G$ has a simple effect on the set of stable models of F: it eliminates the stable models that do not satisfy the additional conjunctive term.

PROPOSITION 4. *A set of atoms is a stable model of $F \wedge \neg G$ iff it is a stable model of F that satisfies $\neg G$.*

Proof. *Case 1:* X satisfies $F \wedge \neg G$. Then X does not satisfy G, and $(F \wedge \neg G)^X$ is $F^X \wedge \neg\bot$, which is equivalent to F^X. Consequently X is minimal among the sets satisfying F^X iff it is minimal among the sets satisfying $(F \wedge \neg G)^X$. *Case 2:* X does not satisfy $F \wedge \neg G$. Then X cannot be a model of F that satisfies $\neg G$. ■

In the terminology of ASP, a *constraint* is simply a formula beginning with negation. To illustrate the special role of constraints as additional conjunctive terms, let us go back to example (3), where adding the conjunctive term p to formula (1) changed its collection of stable models nonmonotonically. If we conjoin (1) with the constraint $\neg p$ instead then we will get the formula

$$(\neg p \rightarrow q) \wedge \neg p.$$

Since the only stable model $\{q\}$ of (1) satisfies the constraint, the conjunction has $\{q\}$ as the only stable model as well. If we conjoin (1) with the constraint $\neg\neg p$ then we will get the formula

$$(\neg p \rightarrow q) \wedge \neg\neg p.$$

Since the only stable model $\{q\}$ of (1) does not satisfy this constraint, the conjunction has no stable models.

2.5 LPARSE and SMODELS

We briefly interrupt now the discussion of the theory of stable models to talk about the capabilities of one of the widely used answer set solvers, SMODELS.[5] Its frontend LPARSE serves also as the frontend of three other systems for computing stable models: GNT[6], ASSAT[7] and CMODELS[8]. This frontend requires that the input formula be represented in a special format, as a list (conjunction) of "rules," somewhat similar to Prolog rules. In this paper the reader will find many examples of representing formulas in the input language of LPARSE. A detailed description of that language can be found in the LPARSE manual, available online (see Footnote 1).

Our first example illustrates representing Horn formulas. Formula (5) would be represented in an LPARSE input file as follows:

[5] `http://www.tcs.hut.fi/Software/smodels/` .
[6] `http://www.tcs.hut.fi/Software/gnt/` .
[7] `http://assat.cs.ust.hk/` .
[8] `http://www.cs.utexas.edu/users/tag/cmodels/` .

```
p.
q :- p.
s :- q, r.
```

Note that each conjunctive term here is written as a separate rule, followed by a period. In each rule $A \leftarrow F$, the left arrow is written as `:-` and conjunction as a comma.

To instruct SMODELS to find the stable model of (5), we save the three lines shown above in a file, called, say, `file5`, and invoke LPARSE and SMODELS as follows:

```
% lparse file5 | smodels
```

The main part of the output generated in response to this command line is the program's stable model:

```
Answer: 1
Stable Model: q p
```

Negation in front of an atom is represented in the language of LPARSE as `not`. For instance, formula (4) would be written in a file, say, `file4`, as

```
q :- not p.
p :- not q.
```

The command line

```
% lparse file4 | smodels 0
```

instructs SMODELS to find the stable models of this formula. The zero at the end indicates that we want to compute all stable models; a positive number k in this position would tell SMODELS to terminate after computing k stable models. SMODELS will produce the following output:

```
Answer: 1
Stable Model: q
Answer: 2
Stable Model: p
```

To represent a choice formula $\{A_1, \ldots, A_n\}^c$ in the language of LPARSE, we simply drop the superscript ^{c}. A constraint $\neg F$, where F is a conjunction of literals, is written as `:-` F. For instance, the formula

$$\{p, q, r\}^c \wedge \neg\neg p \wedge \neg(q \wedge \neg r)$$

can be written in the syntax of LPARSE as[9]

[9]The choice construct was originally defined as an addition to the language of LPARSE in [Simons *et al.*, 2002], and it was treated there as a primitive, rather than abbreviation. The equivalence of our treatment of choice formulas to that definition follows from results of [Ferraris and Lifschitz, 2005] and [Ferraris, 2005].

```
{p,q,r}.
:- not p.
:- q, not r.
```

The search process employed in SMODELS is quite sophisticated, and it guarantees, in principle, that every stable model of the given input will be found. It can be viewed as a modification of the Davis-Putnam-Logemann-Loveland procedure for the propositional satisfiability problem (SAT) [Davis *et al.*, 1962]. We should note that finding a stable model of a formula is more difficult than SAT: the existence of a stable model is a Σ_2^P-complete property [Eiter and Gottlob, 1993]. But most uses of ASP involve formulas of special syntactic forms for which this property is known to be in class NP.[10]

Systems ASSAT and CMODELS operate in a different way: they reduce the problem of computing stable models of a given formula to an instance (or a series of instances) of SAT and then invoke SAT solvers to do search.

2.6 Strong Equivalence

As discussed in the introduction, two formulas have the same stable models if they are intuitionistically equivalent (see Section A.3 for the definition). The theorem below is stronger than this claim in several ways.

We say that a formula F is *strongly equivalent* to a formula G if any formula F' that contains an occurrence of F has the same stable models as the formula G' obtained from F' by replacing that occurrence with G. (As the term "strongly equivalent" suggests, this relation turns out to be stronger than equivalence in the sense of classical logic.) For instance, $p \to q$ has the same stable model as $p \to r$ (the empty set), but these two formulas are not strongly equivalent. Indeed, take F' to be $(p \to q) \wedge p$. Then G' is $(p \to r) \wedge p$. These two formulas have different stable models: $\{p, q\}$ and $\{p, r\}$ respectively. The theorem below shows, however, that intuitionistically equivalent formulas are strongly equivalent. This is not surprising in view of the replacement property of intuitionistic logic (Section A.3): if F is intuitionistically equivalent to G then F' is intuitionistically equivalent to G'.

The role of strong equivalence in the practice of answer set programming is determined by the fact that it allows us to simplify a part of a program without looking at the rest of it. For instance, we can observe that the formula $p \wedge (p \to q)$ is intuitionistically equivalent to $p \wedge q$. It follows that

[10]There are exceptions, however, and the answer set solvers GNT and CMODELS are among the systems that are not limited to "NP cases" of answer set programming. System DLV (`http://www.dbai.tuwien.ac.at/proj/dlv/`) is the earliest answer set solver of this kind.

in any program containing the rules

```
p.
q :- p.
```

replacing the second rule by

```
q.
```

will have no effect on the set of stable models.

If formulas F and G are not strongly equivalent to each other then this can be always demonstrated using a counterexample F' that is not much more complicated than F. We can always take F' to be a formula of the form $F \wedge H$, where H is a Horn formula (see Section 2.2 for the definition). One can say even more. A formula H is *unary* if it is a conjunction of several (0 or more) atoms and implications of the form $A_1 \rightarrow A_2$, where A_1 and A_2 are atoms. The theorem below shows that formulas F and G are strongly equivalent iff, for every unary H, $F \wedge H$ and $G \wedge H$ have the same stable models.

Furthermore, in the statement of the theorem intuitionistic logic is replaced with a stronger subsystem of classical logic, called "the logic of here-and-there." This logic is reviewed in Section A.2. Its role in the theory of stable models was first recognized in [Pearce, 1997], where it was used to define a nonmonotonic "equilibrium logic"; the definition of a stable model in Section 2 is equivalent to the semantics of equilibrium logic [Ferraris, 2005, Theorem 1].

Finally, the theorem below asserts not only that equivalence in the logic of here-and-there implies strong equivalence, but that the converse holds also. Thus the logic of here-and-there provides a complete characterization of strong equivalence. This fact, as well as the property of unary formulas pointed out above, was first established in [Lifschitz *et al.*, 2001].

THEOREM 5. *For any formulas F and G, the following conditions are equivalent:*

(i) *F is strongly equivalent to G,*

(ii) *for every unary formula H, $F \wedge H$ and $G \wedge H$ have the same stable models,*

(iii) *F is equivalent to G in the logic of here-and-there.*

Here are some examples of the use of the most important part of this theorem, the implication from (iii) to (i). As mentioned in Section 2.1, formulas $\neg\neg p$ and $\neg p \rightarrow p$ have no stable models. We can now say more:

since these formulas are intuitionistically equivalent (Section A.3), the result of replacing the subformula $\neg p \rightarrow p$ in any formula with $\neg\neg p$ does not change its stable models. In particular, any program containing the rule

```
p :- not p.
```

can be simplified by replacing that rule with

```
:- not p.
```

Another example: the formulas $p \vee \neg p$ and $\neg\neg p \rightarrow p$ are equivalent to each other in the logic of here-and-there (Section A.3); consequently, they are strongly equivalent. We know that the first of these formulas can be written in the input language of LPARSE as

```
{p}.
```

That language allows us to represent the double negation of an atom in the body of a rule as well: $\neg\neg A$ can be written as $\{\texttt{not}\ A\}\texttt{0}$. (This is a special case of "cardinality" notation discussed in Section 3.1 below.) In particular, we can write $\neg\neg p \rightarrow p$ as the rule

```
p :- {not p} 0.
```

One more example of the use of Theorem 5 is given by the proof of the following proposition:

PROPOSITION 6. *Let Z be the set of atoms occurring in a formula F. A subset X of Z satisfies F iff X is a stable model of $Z^c \wedge F$.*

Proof. From Propositions 3 and 4, a subset X of Z satisfies F iff X is a stable model of $Z^c \wedge \neg\neg F$. It remains to observe that $\neg\neg F \leftrightarrow F$ can be intuitionistically derived from Z^c, because Z^c is the conjunction of the excluded middle formulas $A \vee \neg A$ for all atoms A occurring in this equivalence. ■

Proposition 6 provides a reduction of the propositional satisfiability problem to ASP: to find a model of F, look for a stable model of the conjunction of F with the excluded middle formulas $A \vee \neg A$ for all atoms A occurring in F.

An alternative approach to proving the strong equivalence of propositional formulas, based on [Lifschitz *et al.*, 1999, Section 4] and [Turner, 2003], does not require the knowledge of intuitionistic logic or the logic of here-and-there. We can show that F is strongly equivalent to G by checking that, for every set X of atoms, the reducts F^X and G^X are equivalent to each other in classical logic:

THEOREM 7 ([Ferraris, 2005]). *For any formulas F and G, F is strongly equivalent to G iff, for every set X of atoms, F^X is equivalent to G^X in classical logic.*

For instance, the fact that $\neg p \rightarrow p$ is strongly equivalent to $\neg\neg p$ can be established by the following computation:

$$\begin{array}{lcccc} (\neg p \rightarrow p)^{\{p\}} & = & \bot \rightarrow p & \leftrightarrow & \top, \\ (\neg\neg p)^{\{p\}} & = & \neg\bot & = & \top; \\ & & & & \\ (\neg p \rightarrow p)^{\emptyset} & = & \bot, & & \\ (\neg\neg p)^{\emptyset} & = & \bot. & & \end{array}$$

2.7 Splitting

Theorem 2 describes stable models of Horn formulas; Proposition 3 describes stable models of choice formulas. Many ASP programs involve conjunctions of formulas of these two types. To design such programs, we need to understand the structure of their stable models.

Consider the following example:

(6) $\{p, q\}^c \wedge (p \rightarrow r) \wedge (q \wedge r \rightarrow s)$.

The first conjunctive term of this formula has 4 stable models:

(7) $\emptyset$, $\{p\}$, $\{q\}$, $\{p, q\}$.

The rest of the conjunction can be viewed as a "definition," characterizing r and s in terms of p and q. Appending this definition to the choice formula $\{p, q\}^c$ does not affect the total number of its stable models, but it can change each of the models (7) by adding to it some of the atoms r, s. In view of the implication $p \rightarrow r$, atom r is added to each model containing p. In view of the implication $q \wedge r \rightarrow s$, atom s is added to each model containing both q and r. Thus we can expect that (6) will have the following stable models:

(8) $\emptyset$, $\{p, r\}$, $\{q\}$, $\{p, q, r, s\}$.

The validity of this claim can be justified using the theorem below, which shows that in some cases we can compute the stable models of a conjunction by "splitting" it into conjunctive terms and computing first the stable models of one of these terms. Splitting was proposed in [Lifschitz and Turner, 1994] and generalized and simplified in [Erdoğan and Lifschitz, 2004] and [Ferraris, 2005].

THEOREM 8. *Let F and G be formulas such that F does not contain any head atoms of G. A set X of atoms is a stable model of $F \wedge G$ iff there* *exists a stable model* $\{A_1, \ldots, A_n\}$ *of* F *such that* X *is a stable model of* $A_1 \wedge \cdots \wedge A_n \wedge G$.

(For the definition of a head atom see Section 2.1.)

In application to (6), we take $\{p, q\}^c$ to be F and $(p \rightarrow r) \wedge (q \wedge r \rightarrow s)$ to be G. Since F does not contain any of the head atoms r, s of G, the stable models of $F \wedge G$ can be generated by taking each of the stable models (7) of F, conjoining its elements with G, and listing all stable models of each of the resulting formulas

$$\begin{array}{r} (p \rightarrow r) \wedge (q \wedge r \rightarrow s), \\ p \wedge (p \rightarrow r) \wedge (q \wedge r \rightarrow s), \\ q \wedge (p \rightarrow r) \wedge (q \wedge r \rightarrow s), \\ p \wedge q \wedge (p \rightarrow r) \wedge (q \wedge r \rightarrow s). \end{array}$$

Since these are Horn formulas, each of them has one stable model—its minimal model. As we have conjectured, these stable models are the sets (8) shown above.

Here is an example of the use of Theorem 8 in the case when F is not a choice formula. We want to find the stable models of the conjunction

(9) $(\neg p \rightarrow q) \wedge (q \rightarrow r)$.

The only stable model of the first conjunctive term is $\{q\}$ (Section 2.1). According to Theorem 8, it follows that (9) has the same stable models as

$$q \wedge (q \rightarrow r).$$

This is a Horn formula, and its minimal model $\{q, r\}$ is its only stable model.

Theorem 8 can be useful also when G is not a Horn formula. But in such cases $A_1 \wedge \cdots \wedge A_n \wedge G$ will not be a Horn formula either, and computing its stable models may require additional work. The following two propositions can often help at this stage.

Notation: F^A_G stands for the formula obtained from a formula F by substituting a formula G for all occurrences of an atom A.

PROPOSITION 9. *For any atom* A *that is not a head atom of* F, F *has the same stable models as* $F^A_\bot$.

Proof. Since A is not a head atom of F, A does not belong to any of the stable models of F (Theorem 1). Consequently, F has the same stable models as $F \wedge \neg A$ (Proposition 4). Similarly, $F^A_\bot$ has the same stable model as $F^A_\bot \wedge \neg A$. It remains to observe that $F \wedge \neg A$ and $F^A_\bot \wedge \neg A$ are intuitionistically equivalent to each other by the replacement property of intuitionistic logic (Section A.3). ■

Example: Using Proposition 9, we can find the stable model of $\neg p \to q$ without directly referring to the definition of a stable model as in Section 2.1. Since p is not a head atom of $\neg p \to q$, this formula has the same stable models as $\neg\bot \to q$, which is intuitionistically equivalent to the Horn formula q. Consequently, the only stable model of $\neg p \to q$ is q.

PROPOSITION 10. *For any atom A, a set X of atoms is a stable model of $F \wedge A$ iff there exists a stable model Y of $F^A_\top$ such that $X = Y \cup \{A\}$.*

Proof. By the replacement property of intuitionistic logic, $F \wedge A$ is intuitionistically equivalent to $F^A_\top \wedge A$, so that the two formulas have the same stable models. By Theorem 8, X is a stable model of $F^A_\top \wedge A$ iff there exists a stable model $\{A_1,\dots,A_n\}$ of $F^A_\top$ such that X is a stable model of $A_1 \wedge \dots \wedge A_n \wedge A$. The only stable model of this Horn formula is $\{A_1,\dots,A_n,A\}$, which can be written as $\{A_1,\dots,A_n\} \cup \{A\}$. ■

Example: Proposition 10 can be used to verify the claim that the only stable model of (3) is $\{p\}$. To find the stable models of $(\neg p \to q) \wedge p$, we need to add p to each stable model of $\neg\top \to q$. Since this formula is intuitionistically equivalent to $\top$, its only stable model is the empty set.

The following example shows how Propositions 9 and 10 can be used in combination with splitting. We want to find the stable models of

(10) $\{p,q\}^c \wedge (\neg p \to r)$.

By Theorem 8, this can be done by computing the stable models of each of the conjunctions

$$\begin{aligned} \neg p &\to r,\\ p \wedge (\neg p &\to r),\\ q \wedge (\neg p &\to r),\\ p \wedge q \wedge (\neg p &\to r). \end{aligned}$$

Proposition 9 shows that the only stable model of the first of these formulas is $\{r\}$. Proposition 10 shows that the only stable model of the second formula is $\{p\}$. Proposition 9 shows that the only stable model of the third formula is $\{q,r\}$. Proposition 10 shows that the only stable model of the last formula is $\{p,q\}$. Consequently, (10) has 4 stable models:

$$\{p\},\ \{r\},\ \{q,r\},\ \{p,q\}.$$

Using Theorem 8 twice, we can derive the following useful fact:

PROPOSITION 11. *Let F and G be formulas such that F does not contain head atoms of G, and G does not contain head atoms of F. A set of atoms is a stable model of $F \wedge G$ iff it can be represented as the union of a stable model of F and a stable model of G.*

Proof. By Theorem 8, X is a stable model of $F \wedge G$ iff there exists a stable model $\{A_1, \ldots, A_n\}$ of F such that X is a stable model of

(11) $G \wedge (A_1 \wedge \cdots \wedge A_n)$.

By Theorem 1, for any stable model $\{A_1, \ldots, A_n\}$ of F, atoms $A_1, \ldots, A_n$ are head atoms of F. Consequently they are different from the head atoms of G, so that the head atoms of G do not occur in the second conjunctive term of (11). By Theorem 8, X is a stable model of (11) iff there exists a stable model $\{B_1, \ldots, B_m\}$ of G such that X is a stable model of

$$B_1 \wedge \cdots \wedge B_m \wedge A_1 \wedge \cdots \wedge A_n,$$

that is to say, such that

$$X = \{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_m\}.$$

■

3 Programming

By an ASP program we understand a propositional formula that can be easily communicated to an answer set solver. We want to learn how to represent a given search problem as the problem of computing a stable model of such a formula.

After discussing in Sections 3.1 and 3.2 a few features of the language of LPARSE that have not been mentioned earlier, we give several examples of computational problems that can be solved using ASP (Sections 3.3–3.8). Then we talk about answer set programming with strong negation (Section 3.9) and about its application to representing actions (Section 3.10).

3.1 Cardinality Expressions

In answer set programming we often need formulas expressing conditions on cardinalities of sets. The following notation is useful. For any nonnegative integer l ("lower bound") and formulas $F_1, \ldots, F_n$,

(12) $l \leq \{F_1, \ldots, F_n\}$

stands for the disjunction

$$\bigvee_{I \subseteq \{1,\ldots,n\},\ |I|=l} \quad \bigwedge_{i \in I} F_i.$$

For instance,

$$2 \leq \{F_1, F_2, F_3\}$$

stands for

$$(F_1 \wedge F_2) \vee (F_1 \wedge F_3) \vee (F_2 \wedge F_3).$$

By

(13) $\{F_1, \dots, F_n\} \leq u$

where u is a nonnegative integer ("upper bound") we denote the formula

$$\neg(u + 1 \leq \{F_1, \dots, F_n\}).$$

Finally,

(14) $l \leq \{F_1, \dots, F_n\} \leq u$

stands for

$$(l \leq \{F_1, \dots, F_n\}) \wedge (\{F_1, \dots, F_n\} \leq u).$$

It is clear that any set of atoms

- satisfies (12) iff it satisfies at least l of the formulas $F_1, \dots, F_n$;
- satisfies (13) iff it satisfies at most u of the formulas $F_1, \dots, F_n$;
- satisfies (14) iff it satisfies at least l and at most u of the formulas $F_1, \dots, F_n$.

The input language of LPARSE allows us to use expressions (12)–(14) in the bodies of rules, with the symbol $\leq$ dropped, if all formulas $F_1, \dots, F_n$ are literals.[11] We saw an example in Section 2.3: the implication $\neg\neg p \rightarrow p$ can be represented in an LPARSE input file as

```
p :- {not p} 0.
```

because

$$\{\neg p\} \leq 0 \quad = \quad \neg(1 \leq \{\neg p\}) \quad = \quad \neg\neg p.$$

If $A_1, \dots, A_n$ are pairwise distinct atoms then we will write

$$\begin{array}{rcl} l \leq \{A_1, \dots, A_n\}^c & \text{for} & \{A_1, \dots, A_n\}^c \wedge (l \leq \{A_1, \dots, A_n\}), \\ \{A_1, \dots, A_n\}^c \leq u & \text{for} & \{A_1, \dots, A_n\}^c \wedge (\{A_1, \dots, A_n\} \leq u), \\ l \leq \{A_1, \dots, A_n\}^c \leq u & \text{for} & \{A_1, \dots, A_n\}^c \wedge (l \leq \{A_1, \dots, A_n\} \leq u). \end{array}$$

The following proposition explains why these are useful abbreviations.

PROPOSITION 12. *For any pairwise distinct atoms $A_1, \dots, A_n$, nonnegative integers l and u, and a set X of atoms,*

[11] What we said in Footnote 9 about the invention of choice formulas applies also to "cardinality formulas" (12)–(14) and to their combinations with choice formulas introduced below. These expressions were originally introduced in [Simons *et al.*, 2002] as primitives. The equivalence of our presentation to that definition follows from results of [Ferraris and Lifschitz, 2005] and [Ferraris, 2005].

(i) X *is a stable model of* $l \leq \{A_1,\ldots,A_n\}^c$ *iff* $X \subseteq \{A_1,\ldots,A_n\}$ *and* $l \leq |X|$;

(ii) X *is a stable model of* $\{A_1,\ldots,A_n\}^c \leq u$ *iff* $X \subseteq \{A_1,\ldots,A_n\}$ *and* $|X| \leq u$;

(iii) X *is a stable model of* $l \leq \{A_1,\ldots,A_n\}^c \leq u$ *iff* $X \subseteq \{A_1,\ldots,A_n\}$ *and* $l \leq |X| \leq u$.

Proof: Immediate from Proposition 6. ■

For instance, the stable models of

$$2 \leq \{p,q,r\}^c \leq 2$$

are

$$\{p,q\},\ \{p,r\},\ \{q,r\}.$$

Expressions of the forms

$$l \leq \{\cdots\}^c,\ \ \{\cdots\}^c \leq u,\ \ l \leq \{\cdots\}^c \leq u$$

can be used in LPARSE code in the head of a rule, with both $\leq$ and the superscript c dropped:

```
l {...},   {...} u,   l {...} u.
```

Note that LPARSE understands expressions of these types in different ways depending on whether they occur in the body or in the head of a rule: a choice formula is included in the second case, but not in the first. For instance, the LPARSE rules

```
r :- 1 {p,q}.
1 {p,q} :- r.
```

stand for

$$1 \leq \{p,q\} \rightarrow r$$

and

$$r \rightarrow 1 \leq \{p,q\}^c$$

respectively.

3.2 Variables in the Language of LPARSE

A group of rules that follow a pattern can be often described concisely in the input language of LPARSE using schematic variables. As in Prolog, variables must be capitalized. Here is an example:

```
p(1..4).
#domain p(I).
q(I) :- not q(I-1).
```

Assume that these 3 lines are saved as file `var`. The first line of `var` is an LPARSE abbreviation for a group of 4 rules:

```
p(1).  p(2).  p(3).  p(4).
```

It defines the auxiliary "domain" predicate[12] `p`, which is used in the second line to declare `I` to be a variable with the domain $\{1,\dots,4\}$. The last line of `var` is interpreted then as a schematic representation of 4 rules:

```
q(1) :- not q(0).
q(2) :- not q(1).
q(3) :- not q(2).
q(4) :- not q(3).
```

Generating these rules from the schematic expression at the end of file `var` is an example of "grounding," which is the main computational task performed by LPARSE.

To sum up, LPARSE interprets `var` as the conjunction of the formulas

$$(15)\quad \begin{array}{l} p(i), \\ \neg q(i-1) \rightarrow q(i) \end{array}$$

$(1 \leq i \leq 4)$. In response to the command

```
% lparse var | smodels 0
```

SMODELS will compute the only stable model of this conjunction:

```
Stable Model: q(1) q(3) p(1) p(2) p(3) p(4)
```

The auxiliary atoms `p(1)`,...,`p(4)` in the output can be suppressed by including the declaration

```
hide p(_).
```

[12] The general definition of a domain predicate in the LPARSE manual is somewhat complicated, and it has been changing from one version of the system to another.

("do not display atoms of the form `p(_)`") in file `var`. Alternatively, displaying information about domain predicates can be suppressed using the `-d` option of LPARSE, as follows:

```
% lparse -d none var | smodels 0
```

Besides `#domain` declarations, there is another mechanism for telling LPARSE how to ground schematic rules: domain predicates can be included directly in the bodies of these rules. For instance, file `var` can be rewritten in the following way:

```
p(1..4).
q(I) :- p(I), not q(I-1).
```

These two lines represent the conjunction of the formulas

$$(16)\quad \begin{array}{l} p(i), \\ p(i) \wedge \neg q(i-1) \rightarrow q(i) \end{array}$$

($1 \leq i \leq 4$). Since the conjunction of formulas (16) is intuitionistically equivalent to the conjunction of formulas (15), these two conjunctions have the same stable models.

In the language of LPARSE, variables can be used also to describe a list of literals that is formed according to a pattern. For instance, LPARSE understands

```
p(1..4).
2 {q(I) : p(I)} 3.
```

as shorthand for

```
p(1). p(2). p(3). p(4).
2 {q(1), q(2), q(3), q(4)} 3.
```

3.3 Graph Coloring

As we are turning to actual programming examples, we would like to facilitate representing propositional formulas by input files of answer set solvers. To this end, we will usually write formulas in "logic programming notation." A conjunction will be written as a list of its conjunctive terms; if a conjunctive term is an implication $F \rightarrow G$ then it will be written as $G \leftarrow F$; if a conjunctive term is a constraint $\neg F$ then it will be written as $\leftarrow F$. If the body F in such an expression is a conjunction then we will separate its conjunctive terms by commas; if the body or one of its conjunctive terms is a literal $\neg A$ then we will write it as *not* A.

An *n-coloring* of a graph G is a function f from its set of vertices to $\{1, \ldots, n\}$ such that $f(x) \neq f(y)$ for every pair of adjacent vertices x, y.

We would like to use ASP to find an n-coloring of a given graph or to determine that it does not exist. To this end, we will write a program whose answer sets are in a 1–1 correspondence with the n-colorings of G.

Let V be the set of vertices of the graph, and E the set of its edges. The program consists of the rules

(17) $1 \leq \{color(x,1),\dots,color(x,n)\}^c \leq 1 \qquad (x \in V),$

(18) $\leftarrow color(x,i), color(y,i) \qquad (\{x,y\} \in E;\ 1 \leq i \leq n).$

It has the desired property:

PROPOSITION 13. *A set X of atoms is a stable model of the conjunction of (17) and (18) iff X is*

(19) $\{color(x,f(x)) : x \in V\}$

for some n-coloring f of $\langle V,E\rangle$.

Proof. By Proposition 12(iii), each of the formulas (17) has n stable models $\{color(x,i)\}$ $(i=1,\dots,n)$. By Proposition 11, it follows that arbitrary stable models of the conjunction of these formulas are unions of such singletons, one per each $x \in V$. In other words, stable models of (17) can be characterized as sets of the form (19), where f is a function from V to $\{1,\dots,n\}$.

By Proposition 4, it follows that the stable models of the conjunction of (17) with the constraints (18) can be characterized as the sets of the form (19) that do not satisfy the bodies of the constraints. The last condition can be expressed by saying that the equalities $f(x)=i$ and $f(y)=i$ cannot hold simultaneously when $\{x,y\} \in E$, which means that $f(x) \neq f(y)$ whenever $\{x,y\} \in E$. ∎

Program (17), (18) can be encoded in the language of LPARSE as the following file `color`:

```
c(1..n).
1 {color(X,I) : c(I)} 1 :- v(X).
:- color(X,I), color(Y,I), e(X,Y), c(I).
```

The domain predicates `v` and `e`, characterizing the vertices and edges of G, are assumed to be defined in a separate file, called, say, `graph`. For instance, if G is the 3-dimensional cube then that file may look like this:

```
v(0..7).

e(0,1).  e(1,2).  e(2,3).  e(3,0).
e(4,5).  e(5,6).  e(6,7).  e(7,4).
e(0,4).  e(1,5).  e(2,6).  e(3,7).
```

(There is no need to include atoms with the opposite order of arguments, such as `e(1,0)`; it is only essential that the adjacency relation of G be the symmetric closure of `e`.) The command line uses the `-c` option of LPARSE to specify the value of the symbolic constant `n`, and it instructs LPARSE to concatenate the files `graph` and `color`:

```
% lparse -c n=2 -d none graph color | smodels
```

In response, SMODELS produces the set of atoms describing a 2-coloring of the cube:

```
Stable Model: color(0,1) color(1,2) color(2,1) color(3,2)
color(4,2) color(5,1) color(6,2) color(7,1)
```

As can be seen from the proof of Proposition 13, the first part (17) of our coloring program describes a "simple" superset of the set of n-colorings of G that we are trying to capture; the second half (18) consists of the constraints that weed out the "bad" elements of that superset. This "generate-and-test" organization is typical for simple ASP programs. But it would be a mistake to think that answer set solvers operate by generating the elements of the superset described in the generate part and verifying which of these elements satisfy the test conditions. As briefly discussed at the end of Section 2.5, the search algorithms implemented in these systems are based on very different ideas.

3.4 Cliques

A *clique* in a graph G is a set of pairwise adjacent vertices of G. We would like to use ASP to find a clique of a cardinality $\geq n$ in a given graph or to determine that it does not exist. To this end, we will write a program whose answer sets are in a 1–1 correspondence with cliques of cardinalities $\geq n$.

As before, V is the set of vertices of the graph, and E the set of its edges. The program consists of the rules

(20) $n \leq \{in(x) : x \in V\}^c$,

(21) $\leftarrow in(x), in(y) \qquad\qquad (x, y \in V;\ x \neq y;\ \{x, y\} \notin E)$.

PROPOSITION 14. *A set X of atoms is a stable model of the conjunction of (20) and (21) iff X is*

(22) $\{in(x) : x \in C\}$

for some clique C in G such that $|C| \geq n$.

Proof. By Proposition 12(i), the stable models of (20) can be characterized as sets of the form (22), where C is a set of vertices of a cardinality $\geq n$. By Proposition 4, it follows that the stable models of the conjunction of (20) with the constraints (21) can be characterized as the sets of the form (22) that do not satisfy the bodies of the constraints. The last condition can be expressed by saying that the conditions $x \in C$ and $y \in C$ cannot hold simultaneously for two different non-adjacent vertices x, y, which means that C is a clique. ∎

Here is an LPARSE encoding of (20), (21):

```
n {in(X) : v(X)}.
:- in(X), in(Y), v(X;Y), X!=Y, not e(X,Y), not e(Y,X).
```

The domain predicates `v` and `e` are assumed to characterize the vertices and edges of G, as in Section 3.3. In the body of the second rule, `v(X;Y)` is an LPARSE abbreviation for `v(X),v(Y)`, and `!=` represents $\neq$. The pair of conditions `not e(X,Y), not e(Y,X)` expresses that `X` and `Y` are non-adjacent.

3.5 Schur Numbers

A set S of integers is called *sum-free* if there are no numbers x, y in S such that $x+y$ is in S. For instance, $\{1, 3, 5\}$ is sum-free, and $\{2, 3, 5\}$ and $\{2, 4\}$ are not. We would like to use ASP to find, for given k and n, a partition of the interval $\{1, \ldots, n\}$ into at most k sum-free sets or to determine that such a partition does not exist. (The largest n such that $\{1, \ldots, n\}$ can be partitioned into k sum-free set is called the k-th *Schur number* and denoted by $S(k)$.)

In the following program the atoms $s_i(x)$ $(1 \leq i \leq k,\ 1 \leq x \leq n)$ are used to express that x belongs to the i-th set S_i in a partition of $\{1, \ldots, n\}$ into sum-free sets $S_1, \ldots, S_k$:

(23) $1 \leq \{s_1(x), \ldots, s_k(x)\}^c \leq 1 \qquad (1 \leq x \leq n),$

(24) $\leftarrow s_i(x), s_i(y), s_i(x+y) \qquad (1 \leq i \leq k;\ x, y \geq 1;\ x+y \leq n).$

The proposition below expresses the correctness of this program. Note that the conditions on the sets S_i in the statement of the proposition allow these sets to be empty, so that the list $S_1, \ldots, S_k$ represents a partition into *at most* k sets, not exactly k.

PROPOSITION 15. *A set X of atoms is a stable model of the conjunction of (23) and (24) iff X is*

(25) $\{s_i(x) \,:\, 1 \leq i \leq k;\ x \in S_i\}$

for sum-free pairwise disjoint sets $S_1, \ldots, S_k$ such that

(26) $S_1 \cup \cdots \cup S_k = \{1, \ldots, n\}$.

Proof. By Proposition 12(iii), each of the formulas (23) has k stable models $\{s_i(x)\}$ $(i = 1, \ldots, k)$. By Proposition 11, it follows that arbitrary stable models of the conjunction of these formulas are unions of such singletons, one per each $x \in \{1, \ldots, n\}$. In other words, the stable models of (23) can be characterized as sets of the form (25), where the sets S_i are pairwise disjoint and satisfy (26).

By Proposition 4, it follows that the stable models of the conjunction of (23) with the constraints (24) can be characterized as sets of the form (25), where S_i are pairwise disjoint, satisfy (26), and do not satisfy the bodies of the constraints. The last condition can be expressed by saying that each S_i is sum-free. ■

Here is how rules (23), (24) can be written in the language of LPARSE:

```
subset(1..k).
number(1..n).
#domain number(X;Y).

1 {s(I,X) : subset(I)} 1.
:- s(I,X), s(I,Y), s(I,X+Y), subset(I), X+Y<=n.
```

(In the last rule, `<=` is the LPARSE symbol for $\leq$.) In response to the command

```
% lparse -c k=3 -c n=13 -d none schur | smodels
```

SMODELS produces the output

```
Stable Model: s(3,1) s(1,2) s(1,3) s(3,4) s(2,5) s(2,6) s(2,7)
s(2,8) s(2,9) s(3,10) s(1,11) s(1,12) s(3,13)
```

which represents a partition of the interval $\{1, \ldots, 13\}$ into 3 sum-free sets:

$$\{2, 3, 11, 12\} \cup \{5, 6, 7, 8, 9\} \cup \{1, 4, 10, 13\}.$$

If we replace 13 by 14 in the command line then SMODELS will report that the program has no stable models; thus $S(3) = 13$.

3.6 Tiling

We would like to use ASP to find a way to cover an 8×8 chessboard by twenty-one 3×1 tiles and one 1×1 tile.

The idea of the solution below is due to Ashish Gupta (personal communication). The problem can be reformulated as follows: place twenty-one 3×1 tiles on an 8×8 chessboard without overlaps. If a tile is placed on the chessboard horizontally then we will describe its position by the atom $h(x, y)$ ($0 \leq x \leq 5$, $0 \leq y \leq 7$), where x, y are the coordinates of the tile's southwest corner. If a tile is placed on the chessboard vertically then we will describe its position by the atom $v(x, y)$ ($0 \leq x \leq 7$, $0 \leq y \leq 5$); x and y have the same meaning. Call these 96 atoms $A_1, \dots, A_{96}$. The stable models of the rule

(27) $21 \leq \{A_1, \dots, A_{96}\}^c \leq 21$

correspond to all possible ways to place 21 tiles on the chessboard. To this "generate" part we now add the constraints testing an arrangement for overlaps. Overlaps between two horizontal tiles are eliminated by the rules

(28) $\leftarrow h(x, y), h(x + i, y) \qquad (0 \leq x, y \leq 7;\ i = 1, 2).$

For overlaps between the vertical tiles, we include

(29) $\leftarrow v(x, y), v(x, y + i) \qquad (0 \leq x, y \leq 7;\ i = 1, 2).$

Finally, we eliminate overlaps between a horizontal tile and a vertical tile:

(30) $\leftarrow h(x, y), v(x + i, y - j) \qquad (0 \leq x, y \leq 7;\ 0 \leq i, j \leq 2).$

The stable models of program (27)–(30) correspond to the solutions to the tiling problem we are interested in.

The program above can be represented in the language of LPARSE as follows:

```
number(0..7).
#domain number(X;Y;I;J).

hpos(X,Y) :- X<=5.
vpos(X,Y) :- Y<=5.

21 {h(XX,YY) : hpos(XX,YY), v(XX,YY) : vpos(XX,YY)} 21.

:- h(X,Y), h(X+I,Y), 0<I, I<=2.
:- v(X,Y), v(X,Y+I), 0<I, I<=2.
:- h(X,Y), v(X+I,Y-J), I<=2, J<=2.
```

The domain predicates hpos and vpos represent the possible positions of horizontal and vertical tiles. In the output of SMODELS we read:

```
Stable Model: h(5,1) h(5,0) h(3,7) h(3,6) h(3,5) h(3,4) h(3,3)
h(3,2) h(2,1) h(2,0) h(0,7) h(0,6) v(7,5) v(7,2) v(6,5) v(6,2)
v(2,3) v(1,3) v(1,0) v(0,3) v(0,0)
```

3.7 Hamiltonian Cycles

Each of the programs in Sections 3.3–3.6 is a conjunction of choice formulas and constraints. In the next example we will have a chance to use Horn formulas as well.

A *Hamiltonian cycle* in a directed graph G is a closed path that passes through each vertex of G exactly once. We would like to use ASP to find a Hamiltonian cycle in a given directed graph or to determine that it does not exist.

The program below uses the atoms $in(x,y)$ for all edges $\langle x,y\rangle$ of G to express that $\langle x,y\rangle$ belongs to the path. The generate part of the program consists of the choice rules

(31) $\{in(x,y)\}^c \qquad (\langle x,y\rangle \in E)$

(E stands for the set of edges of G). We need to conjoin them with constraints that eliminate all subsets of E other than Hamiltonian cycles.

Two useful constraints are

(32) $\leftarrow 2 \le \{in(x,y) : y \in A_x\} \qquad (x \in V)$,

where A_x stands for $\{y : \langle x,y\rangle \in E\}$, and

(33) $\leftarrow 2 \le \{in(x,y) : x \in B_y\} \qquad (y \in V)$,

where B_y stands for $\{x : \langle x,y\rangle \in E\}$. They ensure that two *in*-edges neither start nor end at the same vertex, so that the set of *in*-edges is a path or a union of disjoint paths. In addition, we want to require that every vertex of G be reachable by a sequence of *in*-edges from some fixed vertex x_0. We will do this using the auxiliary atoms $r(x)$ ("x is reachable from x_0") for all vertices x of G. The following two rules provide a "recursive definition" of r:

(34) $r(x) \leftarrow in(x_0,x) \qquad (x \in V)$,

(35) $r(y) \leftarrow r(x), in(x,y) \qquad (\langle x,y\rangle \in E)$

(V stands for the set of vertices of G). Now we are ready to impose the reachability constraints:

(36) $\leftarrow not\ r(x) \qquad (x \in V)$.

Besides the generate part (31) and the test part (32), (33), (36), this program contains the rules (34) and (35), which define the auxiliary atoms used in one of the test rules. This "generate-define-test" structure is typical for more advanced ASP programs. As in the example above, the definitions of auxiliary atoms are often recursive.

The following proposition expresses the correctness of program (31)–(36). In its statement, the *essential part* of a set X of atoms is the set of atoms in X that have the form $in(x, y)$.

PROPOSITION 16. *A set X of atoms is the essential part of a stable model of (31)–(36) iff X has the form*

(37) $\{in(x,y)) : \langle x,y\rangle \in H\}$

where H is the set of edges of a Hamiltonian cycle in G. Furthermore, different stable models of this program have different essential parts.

The last sentence shows that if we "hide" the atoms of the form $r(x)$ in the list of stable models of this program produced by an answer set solver then the output will contain each Hamiltonian cycle of G exactly once.

For any set $H \subseteq E$, by R_H we denote the set of atoms $r(x)$ for all vertices x to which there is a path of nonzero length from x_0 over edges in H.

LEMMA 17. *A set X of atoms is a stable model of the conjunction of formulas (31), (34) and (35) iff X is*

(38) $\{in(x,y)) : \langle x,y\rangle \in H\} \cup R_H$

for some subset H of E.

Proof. Denote the conjunction of formulas (31) by F, and the conjunction of formulas (34), (35) by G. By Theorem 8, X is a stable model of $F \wedge G$ iff there exists a stable model $\{A_1, \dots, A_n\}$ of F such that X is a stable model of $A_1 \wedge \cdots \wedge A_n \wedge G$. By Proposition 3, it follows that the stable models of $F \wedge G$ can be characterized as the stable models of formulas of the form

$$(39) \quad \bigwedge_{\langle x,y\rangle \in H} in(x,y) \ \wedge\ G$$

for arbitrary subsets H of E. Formula (39) is a Horn formula, and its minimal model is its only stable model (Theorem 2). It remains to observe that the minimal model of (39) is (38). ■

Proof of Proposition 16. A set $H \subseteq E$ is the set of edges of a Hamiltonian cycle in G iff it satisfies the following conditions:

(i) H does not contain two different edges leaving the same vertex.

(ii) H does not contain two different edges ending at the same vertex.

(iii) For every vertex x of G, there exists a path of nonzero length from x_0 to x over edges in H.

By Lemma 17 and Proposition 4, a set X of atoms is a stable model of program (31)–(36) iff X has the form (38), where $H \subseteq E$, and does not satisfy the bodies of the constraints (32), (33), (36). It is clear that

- (i) holds iff (38) does not satisfy the bodies of constraints (32);
- (ii) holds iff (38) does not satisfy the bodies of constraints (33);
- (iii) holds iff (38) does not satisfy the bodies of constraints (36).

Consequently X is a stable model of (31)–(36) iff X has the form (38) for a subset H of E satisfying conditions (i)–(iii). Both parts of the statement of Proposition 16 now follow, because the essential part of (38) is (37). ■

The discussion of the Hamiltonian cycles example above is based on [Erdoğan and Lifschitz, 2004, Section 5].

Here is a representation of program (31)–(36) in the language of LPARSE, assuming that x_0 is `0`:

```
{in(X,Y)} :- e(X,Y).

:- 2 {in(X,Y) : e(X,Y)}, v(X).
:- 2 {in(X,Y) : e(X,Y)}, v(Y).

r(X) :- in(0,X), v(X).
r(Y) :- r(X), in(X,Y), e(X,Y).

:- not r(X), v(X).

hide r(_).
```

3.8 The Blocks World

The blocks world consists of several blocks $1, \ldots, n$, placed on the table so that they form a tower or several towers. For instance, if $n = 2$ then the blocks world can be in 3 states:

```
  1         2
  2         1         1   2
-------   -------   -------
```

If $n = 3$ then 13 states are possible: 6 configurations in which the blocks form one tower; 6 configurations in which 2 blocks form a tower and the third is on the table; one configuration in which all blocks are on the table.

Blocks can be moved around, and in Section 3.10 we show how ASP can be used to find a sequence of actions that takes the blocks world from a given initial state to a given goal state (or, more generally, to a state satisfying a given goal condition). As a preliminary step, in this section we write an ASP program that represents the set of all possible configurations of n blocks.

Positions of blocks are described in this program by the atoms $on(x, y)$, where $x \in \{1, \dots, n\}$, $y \in \{1, \dots, n, table\}$, $x \neq y$. The first rule of the program is the choice rule allowing us to choose arbitrarily, for each block x, a unique location:

(40) $1 \leq \{on(x, y) : y \in \{1, \dots, n, table\} \setminus \{x\}\}^c \leq 1$

$(1 \leq x \leq n)$. Furthermore, we do not allow two blocks to be on top of the same block:

(41) $\leftarrow 2 \leq \{on(x, y) : x \in \{1, \dots, n\} \setminus \{y\}\}$

$(1 \leq y \leq n)$. These constraints are not sufficient, however, for eliminating all "bad" stable models of (40), because they allow subsets of blocks to form circular configurations "floating in space," such as $on(1, 2)$ and $on(2, 1)$.

The absence of such configurations can be expressed using an auxiliary recursively defined predicate, similar to the predicate r used in Section 3.7 to describe Hamiltonian cycles. The atoms $s(x)$, where $1 \leq x \leq n$, will express that x is supported by the table, that is to say, belongs to a tower of blocks that rests on the table. They are defined by the rules

(42) $s(x) \leftarrow on(x, table) \qquad (1 \leq x \leq n),$

(43) $s(x) \leftarrow s(y), on(x, y) \qquad (1 \leq x, y \leq n; x \neq y).$

The absence of blocks floating in space is expressed by the constraints

(44) $\leftarrow not\ s(x) \qquad (1 \leq x \leq n).$

In the language of LPARSE:

```
block(1..n).

1 {on(X,Y) : block(Y) : X!=Y, on(X,table)} 1 :- block(X).

:- 2 {on(X,Y) : block(X) : X!=Y}, block(Y).

s(X) :- on(X,table), block(X).
s(X) :- s(Y), on(X,Y), block(X;Y), X!=Y.

:- not s(X), block(X).

hide s(_).
```

3.9 Strong Negation

Some applications of ASP, including those related to actions and planning, are facilitated by the use of a second kind of negation, called "strong" (or "classical," or "true"), proposed in [Gelfond and Lifschitz, 1991].

Recall that propositional formulas are formed from atoms and the 0-place connective $\bot$ using the binary connectives $\wedge$, $\vee$ and $\rightarrow$ (Section A.1). Assume that we distinguish between atoms of two kinds, *positive* and *negative*, and that each negative atom is an expression of the form $\sim A$, where A is a positive atom. The symbol $\sim$ is called *strong negation*.

Note that syntactically strong negation is not really a connective, according to this definition: it is allowed to occur in front of positive atoms only. For example, expressions $\sim\sim p$ and $\sim(p \wedge q)$ are not formulas.[13]

A set of atoms is *coherent* if it does not contain "complementary" pairs of atoms A, $\sim A$.

Consider, for instance, the program

$$
\begin{array}{ll}
 & \{p\}^c, \\
(45) & q, \\
 & \sim q \leftarrow \neg p.
\end{array}
$$

It contains two positive atoms p, q and one negative atom $\sim q$. It is easy to check using the method of Section 2.7 that the stable models of this program are $\{p, q\}$ and $\{q, \sim q\}$. The first of them is coherent, and the second is not.

The problem of computing the coherent stable models of a formula can be easily reduced to the problem of computing arbitrarily stable models:

PROPOSITION 18. *A set X of atoms is a coherent stable model of a formula F iff X is a stable model of the formula*

[13] Alternatively, strong negation can be treated as an additional connective, in the spirit of [Nelson, 1949].

(46) $F \wedge \bigwedge_A \neg(A \wedge \sim A)$,

where the big conjunction extends over all positive atoms A such that both A and $\sim A$ are head atoms of F.

Proof. By Proposition 4, X is a stable model of (46) iff X is a stable model of F which does not have subsets of the form $\{A, \sim A\}$ such that A, $\sim A$ are head atoms of F. By Theorem 1, this condition on X is equivalent to saying that X is a coherent stable model of F. ■

In the input language of LPARSE, strong negation is written as `-` . When the input program contains strong negation, LPARSE should be called with the option `--true-negation` . The answer set solvers that accept input programs with strong negation, such as SMODELS, generate coherent answer sets only. For instance, if we save the rules

```
{p}.
q.
-q :- not p.
```

as `file45` and give the command

```
% lparse --true-negation file45 | smodels 0
```

then the output will contain only one model:

```
Stable Model: p q
```

Strong negation allows us to distinguish between the assertions "A is false" and "A is not known to be true" in ASP programs. The former is expressed by the presence of the negative atom $\sim A$ in a coherent stable model; the latter, by the absence of the positive atom A, which is obviously a weaker condition. The rule

(47) $\sim A \leftarrow \textit{not}\ A$

("A is false if there is no evidence to the contrary") is an ASP representation of the closed world assumption [Reiter, 1978] for the positive atom A. The following proposition describes the effect of adding this rule on the stable models of a program.

PROPOSITION 19. *Let F be a formula and A a positive atom such that $\sim A$ does not occur in F. For any set X of atoms, X is a coherent stable model of*

(48) $F \wedge (\neg A \rightarrow \sim A)$

iff

(i) X *is a stable model of* F *and* $A \in X$, *or*

(ii) $X = Y \cup \{\sim A\}$, *where* Y *is a stable model of* F *such that* $A \notin Y$.

Case (ii) is the case when there is no evidence that A is true, and the closed world assumption leads us to the conclusion that A is false.

Proof. By Theorem 8, X is a stable model of (48) iff X is a stable model of a formula of the form

(49) $A_1 \wedge \cdots \wedge A_n \wedge (\neg A \rightarrow \sim A)$,

where $\{A_1, \ldots, A_n\}$ is a stable model of F. *Case 1:* A equals one of the atoms A_i. Then (49) is intuitionistically equivalent to $A_1 \wedge \cdots \wedge A_n$, and X is a stable model of (49) iff $X = \{A_1, \ldots, A_n\}$. *Case 2:* A is different from all atoms A_i. Then A is not a head atom of (49). By Proposition 9, it follows that X is a stable model of (49) iff X is a stable model of the formula

$$A_1 \wedge \cdots \wedge A_n \wedge (\neg\bot \rightarrow \sim A),$$

which is intuitionistically equivalent to

$$A_1 \wedge \cdots \wedge A_n \wedge \sim A.$$

So X is a stable model of (48) iff $X = Y \cup \{\sim A\}$, where Y stands for $\{A_1, \ldots, A_n\}$. ∎

The rule

(50) $A \leftarrow \mathit{not}\ \sim A$

expresses the inverse closed world assumption: A is true if there is no evidence to the contrary.

3.10 Planning

We would like to use ASP to find a sequence of actions that takes the blocks world (Section 3.8) from a given initial state to a state satisfying a given goal condition. To be more precise, we will be looking for a sequence of *sets* of actions, because some actions can be executed concurrently. There are n^2 possible actions, where n is the number of blocks: any block $x \in \{1, \ldots, n\}$ can be moved to any location $l \in \{1, \ldots, n, \mathit{table}\}$ different from x.

We assume that

- a block can be moved only when there are no blocks on top of it, and
- at most k actions can be executed concurrently

(think of a robot with k grippers that can only grasp a block from above). We also assume that

- a block x can be moved onto a block y only if y is not being moved at the same time

(the robot's ability to coordinate the movements of the grippers is not good enough for that).

A *history* is a finite sequence

$$s_0, e_0, s_1, e_1, \ldots, e_{m-1}, s_m$$

where $s_0, s_1, \ldots, s_m$ are states of the blocks world, and each e_i $(0 \leq i < m)$ is a set of actions (an "event"), which, when executed concurrently in state s_i, lead to state s_{i+1}. We will write a program whose stable models represent the histories with a given initial state s_0 and a given length m such that their final state s_m satisfies a given goal condition.

Histories will be described by

- the atoms $on(x, l, i)$ $(x \in \{1, \ldots, n\}$, $l \in \{1, \ldots, n, table\}$, $x \neq l$, $i \in \{0, \ldots, m\})$, expressing that x is on l in state s_i, and
- the atoms $move(x, l, i)$ $(x \in \{1, \ldots, n\}$, $l \in \{1, \ldots, n, table\}$, $x \neq l$, $i \in \{0, \ldots, m-1\})$, expressing that x is moved onto l as part of event e_i.

One of the rules of the program (rule (55) below) uses these atoms to describe the effect of moving a block: if x is moved onto l as part of event e_i then x is on l in state s_{i+1}.

The program contains strong negation (Section 3.9), which is applied to the atoms $on(x, l, i)$. The usefulness of strong negation in ASP programs describing effects of actions is related to the frame problem [Shanahan, 1997]—the problem of describing what does *not* change when actions are executed. If $x_1, x_2, \ldots$ are the blocks that are moved in the course of event e_i then rule (55) tells us where these blocks are going to be afterwards. But what about locations of the blocks *other* than $x_1, x_2, \ldots$? An adequate formalization should allow us to conclude that the locations of all the other blocks will not change; in state s_{i+1} each of them will stay where it was in state s_i.

An elegant way to ensure this is to postulate the default that Leibniz stated in his *Introduction to a Secret Encyclopedia* and that is now called the *commonsense law of inertia:* "Everything is presumed to remain in the state in which it is" [Leibniz, 1995, p. 9]. In particular, the location of a block after event e_i is presumed, in the absence of evidence to the contrary, to remain the same as it was before the event. Blocks $x_1, x_2, \dots$ are exceptions: since they are moved, rule (55) provides evidence that their locations may not remain the same. We have seen in Section 3.9 that strong negation helps us formalize another default—the closed world assumption; here strong negation will be used to solve the frame problem.

The generate part of the program expresses that each event e_i can be composed of up to k actions, chosen arbitrarily:

(51) $\{move(x,l,i) \,:\, 1 \le x \le n,\ l \in \{1,\dots,n,table\},\ x \neq l\}^c \le k$

$(1 \le i < m)$. This rule is followed by constraints expressing that a block can be moved only if it is clear

(52) $\leftarrow move(x,l,i), on(y,x,i)$

$(1 \le x,y \le n,\ l \in \{1,\dots,n,table\},\ x \neq l,\ x \neq y,\ 0 \le i < m)$ and if the destination is not a block that is being moved also:

(53) $\leftarrow move(x,y,i), move(y,l,i)$

$(1 \le x,y \le n, l \in \{1,\dots,n,table\}, x \neq y, y \neq l, 0 \le i < m)$.

The next part of the program defines the locations of blocks in state s_i in terms of their initial locations and the events $e_0,\dots,e_{i-1}$. It begins with the rules

(54) $on(x, init(x), 0)$

$(1 \le x \le n)$, where $init(x)$ stands for the initial location of x, and

(55) $on(x,l,i+1) \leftarrow move(x,l,i)$

$(1 \le x \le n, l \in \{1,\dots,n,table\}, x \neq l, 0 \le i < m)$. The next rule expresses the uniqueness of the location of a block using strong negation:

(56) $\sim on(x,l,i) \leftarrow on(x,l',i)$

$(1 \le x \le n,\ l,l' \in \{1,\dots,n,table\},\ x \neq l,\ x \neq l',\ l \neq l',\ 0 \le i \le m)$. The last rule in this group expresses the commonsense law of inertia for the blocks world:

(57) $on(x,l,i+1) \leftarrow on(x,l,i), not \sim on(x,l,i+1)$

($1 \le x \le n$, $l \in \{1,\dots,n,\mathit{table}\}$, $x \ne l$, $0 \le i < m$). It says that

> x is on l in state s_{i+1} (the head of the rule) if
>
> x is on l in state s_i (the first term of the body) and
>
> the rules of the program provide no evidence to the contrary (the second term of the body).

Note that except for the presence of the term $on(x,l,i)$ in the body, (57) has the same syntactic form as (50).

Finally, we need to include constraints expressing that $s_1,\dots,s_m$ are valid states of the blocks world, and that s_m satisfies the goal condition G:

(58) $\leftarrow 2 \le \{on(x,y) \,:\, x \in \{1,\dots,n\} \setminus \{y\}\}$

($0 \le y \le n$, $0 \le i < m$);

(59) $\leftarrow \mathit{not}\ G$.

Rule (58) says that two blocks cannot be on top of the same block; this is a counterpart of rule (41). Counterparts of the other properties of valid states discussed in Section 3.8 are not needed in the new framework. Indeed, the existence of the location of every block and the absence of circular configurations are assumed to hold in the initial state described by the function *init*, and these properties are preserved when blocks are moved; the uniqueness of the location of a block is expressed by (56).

Here is program (51)–(59) in the language of LPARSE:

```
step(0..m).
block(1..n).
location(1..n;table).

#domain step(I).
#domain block(X;Y;Z).
#domain location(L;L1).

{move(XX,LL,I) : block(XX) : location(LL) : XX!=LL} k :- I<m.

:- move(X,L,I), on(Y,X,I), X!=L, X!=Y, I<m.
:- move(X,Y,I), move(Y,L,I), X!=Y, Y!=L, I<m.

on(X,L,0) :- init(X,L).
on(X,L,I+1) :- move(X,L,I), X!=L, I<m.
-on(X,L,I) :- on(X,L1,I), X!=L, X!=L1, L!=L1.
on(X,L,I+1) :- on(X,L,I), not -on(X,L,I+1), X!=L, I<m.
```

```
:- 2 {on(XX,Y,I) : block(XX) : XX!=Y}.
:- not goal.

hide.
show move(_,_,_).
```

The last two lines instruct SMODELS to display no atoms except for the actions `move(...)`. The initial state and the goal condition are assumed to be defined in a separate file, for instance:

```
init(1,2).  init(2,table).  init(3,4).
init(4,table).  init(5,6).  init(6,table).

goal :- on(2,1,m), on(3,2,m), on(6,5,m), on(5,4,m).
```

The idea of the solution to the frame problem given by rule (57) goes back to [Reiter, 1980, Section 1.1.4], but implementing that idea was not straightforward [Hanks and McDermott, 1987], and it was achieved years later [Turner, 1997]. The fact that planning can be reduced to finding a stable model was noted in [Subrahmanian and Zaniolo, 1995], and first experiments on generating plans using SMODELS were reported in [Dimopoulos *et al.*, 1997]. The discussion of blocks world planning in this section is based on [Lifschitz, 2002, Section 5].

4 Proofs of Theorems

4.1 Proof of Theorem 1

LEMMA 20. *If $X \models F$ and a set Y contains all head atoms of F then $X \cap Y \models F^X$.*

Proof. The proof is by structural induction on F. Assume that $X \models F$. Clearly F is not $\bot$. *Case 1:* F is an atom A. Since $X \models F$, F^X is A and $A \in X$. Since A is a head atom, we can further conclude that $A \in X \cap Y$. *Case 2:* F is $G \wedge H$. Since $X \models F$, we know that F^X is $G^X \wedge H^X$, $X \models G$ and $X \models H$. Since all head atoms of G and H belong to Y, from the induction hypothesis we conclude that $X \cap Y \models G^X$ and $X \cap Y \models H^X$. Consequently $X \cap Y \models F^X$. *Case 3:* F is $G \vee H$. Similar to Case 2. *Case 4:* F is $G \to H$. Since $X \models F$, F^X is $G^X \to H^X$. *Case 4.1:* $X \models G$. Then $X \models H$. Since all head atoms of H belong to Y, from the induction hypothesis we conclude that $X \cap Y \models H^X$. Consequently $X \cap Y \models F^X$. *Case 4.2:* $X \not\models G$. Then G^X is $\bot$, so that F^X is a tautology. ■

THEOREM 1. *Any stable model of F is a subset of the set of head atoms of F.*

Proof. Let X be a stable model of F, and Y the set of head atoms of F. By Lemma 20, $X \cap Y \models F^X$. Since X is minimal among the sets satisfying F^X, it follows that $X \cap Y = X$, and consequently $X \subseteq Y$. ■

4.2 Proof of Theorem 2

LEMMA 21. *For any Horn formula F and any two sets X and Y of atoms, if $X \subseteq Y$ and $Y \models F$ then $X \models F$ iff $X \models F^Y$.*

Proof. Assume first that F is a single implication

(60) $A_1 \wedge \cdots \wedge A_n \rightarrow A$.

Case 1: $A_1, \ldots, A_n$ belong to Y. Under the assumption $Y \models F$ the consequent A of F belongs to Y also, so that $F^Y = F$. *Case 2:* for some i, $A_i \notin Y$. Under the assumption $X \subseteq Y$, $A_i \notin X$, so that X satisfies F. On the other hand, F^Y is the tautology $\bot \rightarrow A^Y$, so that X satisfies F^Y as well.

If F is a conjunction $F_1 \wedge \cdots \wedge F_m$ of several implications of the form (60) then X satisfies F iff X satisfies each F_j. Under the assumption $Y \models F$, F^Y is $F_1^Y \wedge \cdots \wedge F_m^Y$; consequently X satisfies F^Y iff X satisfies each of the conjunctive terms F_j^Y. The assertion of the lemma follows from the special case proved above. ■

THEOREM 2. *For any Horn formula F, the minimal model of F is the only stable model of F.*

Proof. Let M be the minimal model of a Horn formula F. Lemma 21, applied to M as Y, shows that F^M is satisfied by M but is not satisfied by any proper subset of M. Consequently M is a stable model of F. Now take any stable model Y of F. By the choice of M, $M \subseteq Y$. Lemma 21, applied to M as X, shows that $M \models F^Y$. By the definition of a stable model, Y is minimal among the sets satisfying F^Y. Consequently $Y \subseteq M$. We have proved that $Y = M$. ■

4.3 Proof of Theorems 5 and 7

LEMMA 22. *For any formula F and any set X of atoms, $X \models F^X$ iff $X \models F$.*

Proof. Reduct F^X is obtained from F by replacing some subformulas that are not satisfied by X with $\bot$. ■

LEMMA 23. *For any two formulas F and G and any set X of atoms,*

(a) $(F \wedge G)^X$ is equivalent to $F^X \wedge G^X$ in classical logic, and

(b) $(F \vee G)^X$ *is equivalent to* $F^X \vee G^X$ *in classical logic.*

Proof. Part (a): consider two cases, depending on whether X satisfies $F \wedge G$. If it does then the two formulas are equal to each other; if not then each of them is equivalent to $\bot$. For part (b), the proof is similar. ■

LEMMA 24. *For any formula* F *and any two sets* X *and* Y *of atoms,* $X \models F^Y$ *iff* $\langle X \cap Y, Y \rangle \models F$.

Proof. The proof is by structural induction on F. If F is $\bot$ then the assertion of the lemma is trivial. If F is an atom A,

$$\begin{array}{lll} X \models A^Y & \text{iff} & A \in Y \text{ and } A \in X \\ & \text{iff} & A \in X \cap Y \\ & \text{iff} & \langle X \cap Y, Y \rangle \models A. \end{array}$$

If F is $G \wedge H$ then, using Lemma 23(a),

$$\begin{array}{lll} X \models (G \wedge H)^Y & \text{iff} & X \models G^Y \wedge H^Y \\ & \text{iff} & X \models G^Y \text{ and } X \models H^Y \\ & \text{iff} & \langle X \cap Y, Y \rangle \models G \text{ and } \langle X \cap Y, Y \rangle \models H \\ & \text{iff} & \langle X \cap Y, Y \rangle \models G \wedge H. \end{array}$$

If F is $G \vee H$ then the reasoning is similar, using Lemma 23(b). Finally, if F is $G \rightarrow H$,

$$\begin{array}{lll} X \models (G \rightarrow H)^Y & \text{iff} & Y \models G \rightarrow H \text{ and } X \models G^Y \rightarrow H^Y \\ & \text{iff} & Y \models G \rightarrow H \text{ and} \\ & & \qquad X \not\models G^Y \text{ or } X \models H^Y \\ & \text{iff} & Y \models G \rightarrow H \text{ and} \\ & & \qquad \langle X \cap Y, Y \rangle \not\models G \text{ or } \langle X \cap Y, Y \rangle \models H \\ & \text{iff} & \langle X \cap Y, Y \rangle \models G \rightarrow H. \end{array}$$

■

LEMMA 25. *Let* F, G, F', G' *be formulas such that* G' *is obtained from* F' *by replacing some (zero or more) occurrences of* F *with* G. *For any set* X *of atoms, if* F^X *is equivalent to* G^X *then* $(F')^X$ *is equivalent to* $(G')^X$.

Proof. Assume that F^X is equivalent to G^X. By Lemma 22, it follows that

(61) $X \models F \leftrightarrow G$.

We will prove that $(F')^X$ is equivalent to $(G')^X$ by structural induction on F'. This assertion is trivial when F' equals F and also when the number of occurrences of F in F' that are being replaced is 0; in particular, the cases when F' is $\bot$ or an atom are trivial. Assume that F' has the form $F'_1 \odot F'_2$, and G' is $G'_1 \odot G'_2$, where G'_i is obtained from F'_i by replacing some occurrences of F with G. *Case 1:* $X \not\models F'$. In view of (61), $X \not\models G'$, so that $(F')^X = \bot$ and $(G')^X = \bot$. *Case 2:* $X \models F'$. In view of (61), $X \models G'$, so that $(F')^X = (F'_1)^X \odot (F'_2)^X$ and $(G')^X = (G'_1)^X \odot (G'_2)^X$, and the claim follows by the induction hypothesis. ∎

COMBINED STATEMENT OF THEOREMS 5 AND 7. *For any formulas F and G, the following conditions are equivalent:*

(i) F is strongly equivalent to G,

(ii) for every unary formula H, $F \wedge H$ and $G \wedge H$ have the same stable models,

(iii) F is equivalent to G in the logic of here-and-there,

(iv) for any set X of atoms, F^X is equivalent to G^X in classical logic.

Proof. From (i) to (ii): obvious.

From (ii) to (iii): assume that F is not equivalent to G in the logic of here-and-there, and let $\langle X, Y\rangle$ be an HT-interpretation that satisfies, say, F but not G. Then $X \subseteq Y$ and, by Lemma 24, $X \models F^Y$, $X \not\models G^Y$. Since $X \models F^Y$, F^Y is not $\bot$, which implies that $Y \models F$. By Lemma 22, it follows that $Y \models F^Y$. *Case 1:* $Y \not\models G^Y$. By Lemma 22, $Y \not\models G$, so that Y is not a stable model of $G \wedge H$ for any H. But if we take H to be $\bigwedge_{A\in Y} A$ then Y is a stable model of $F \wedge H$. Indeed, by Lemma 23(a), $(F \wedge H)^Y$ is equivalent to $F^Y \wedge H^Y$, which is the same as $F^Y \wedge H$; both conjunctive terms of this formula are satisfied by Y, but the second term is not satisfied by any proper subset of Y. *Case 2:* $Y \models G^Y$. Since $X \not\models G^Y$, X is different from Y; consequently X is a proper subset of Y. Let H be the unary formula

$$\bigwedge_{A\in X} A \wedge \bigwedge_{A,A'\in Y\setminus X} (A \to A').$$

Set Y is not a stable model of $F \wedge H$. Indeed, just as in Case 1, $(F \wedge H)^Y$ is equivalent to $F^Y \wedge H$; X is a proper subset of Y that satisfies both conjunctive terms. We will show, on the other hand, that Y is a stable model of $G \wedge H$, which contradicts condition (ii). In view of Lemma 23(a),

$(G \wedge H)^Y$ is equivalent to $G^Y \wedge H$. Clearly Y satisfies both conjunctive terms; the only proper subset of Y that satisfies H is X, and X does not satisfy G^Y.

From (iii) to (iv): if F and G are satisfied by the same HT-interpretations then, by Lemma 24, for any set Y of atoms, F^Y and G^Y are satisfied by the same sets of atoms.

From (iv) to (i): immediate from Lemma 25. ■

4.4 Proof of Theorem 8

LEMMA 26. *If X is a stable model of F then F^X is equivalent to $\bigwedge_{A \in X} A$.*

Proof. Since all atoms occurring in these two formulas belong to X, it is sufficient to show that the formulas are satisfied by the same subsets of X. By the definition of a stable model, the only subset of X satisfying F^X is X. ■

LEMMA 27. *Let S be a set of atoms that contains all atoms occurring in a formula F but does not contain any head atoms of a formula G. For any set X of atoms, if X is a stable model of $F \wedge G$ then $X \cap S$ is a stable model of F.*

Proof. Since X is a stable model of $F \wedge G$, $X \models F$, so that $X \cap S \models F$, and, by Lemma 22, $X \cap S \models F^{X \cap S}$. It remains to show that no proper subset Y of $X \cap S$ satisfies $F^{X \cap S}$. Let S' be the set of head atoms of G, and let Z be $X \cap (S' \cup Y)$. Set Z has the following properties:

(i) $Z \cap S = Y$;

(ii) $Z \subset X$;

(iii) $Z \models G^X$.

To prove (i), note that since S' is disjoint from S, and Y is a subset of $X \cap S$,

$$Z \cap S = X \cap (S' \cup Y) \cap S = X \cap Y \cap S = (X \cap S) \cap Y = Y.$$

To prove (ii), note that set Z is clearly a subset of X. It cannot be equal to X, because otherwise we would have, by (i),

$$Y = Z \cap S = X \cap S;$$

this is impossible, because Y is a proper subset of $X \cap S$. Property (iii) follows from Lemma 20, because $X \models G$, and $S' \cup Y$ contains all head atoms of G.

Since X is a stable model of $F \wedge G$, from property (ii) we can conclude that $Z \not\models (F \wedge G)^X$. Consequently, by Lemma 23(a) and property (iii), $Z \not\models F^X$. Since all atoms occurring in F belong to S, $F^X = F^{X \cap S}$, so that we can rewrite this formula as $Z \not\models F^{X \cap S}$. Since all atoms occurring in $F^{X \cap S}$ belong to S, it follows that $Z \cap S \not\models F^{X \cap S}$. By property (i), we conclude that $Y \not\models F^{X \cap S}$. ■

THEOREM 8. *Let F and G be formulas such that F does not contain any head atoms of G. A set X of atoms is a stable model of $F \wedge G$ iff there exists a stable model $\{A_1, \dots, A_n\}$ of F such that X is a stable model of*

(62) $A_1 \wedge \cdots \wedge A_n \wedge G.$

Proof. Take formulas F and G such that F does not contain any head atoms of G, and let S the set of atoms occurring in F. Observe first that if a set X of atoms is a stable model of a formula of the form (62), where $A_1, \dots, A_n \in S$, then $X \cap S = \{A_1, \dots, A_n\}$. Indeed, by Lemma 27 with $A_1 \wedge \cdots \wedge A_n$ as F, $X \cap S$ is a stable model of $A_1 \wedge \cdots \wedge A_n$, and the only stable model of this formula is $\{A_1, \dots, A_n\}$. Consequently, the assertion to be proved can be reformulated as follows: a set X of atoms is a stable model of $F \wedge G$ iff

(i) $X \cap S$ is a stable model of F, and

(ii) X is a stable model of $\bigwedge_{A \in X \cap S} A \ \wedge \ G$.

If $X \cap S$ is not a stable model of F then X is not a stable model of $F \wedge G$ by Lemma 27. Now suppose that $X \cap S$ is a stable model of F. Then, by Lemma 26, $F^{X \cap S}$ is equivalent to $\bigwedge_{A \in X \cap S} A$. Consequently, by Lemma 23(a),

$$(F \wedge G)^X \ \leftrightarrow \ F^X \wedge G^X \ = \ F^{X \cap S} \wedge G^X \ \leftrightarrow \ \bigwedge_{A \in X \cap S} A \ \wedge \ G^X$$

$$= \ \Big(\bigwedge_{A \in X \cap S} A\Big)^X \wedge G^X \ \leftrightarrow \ \Big(\bigwedge_{A \in X \cap S} A \wedge G\Big)^X.$$

We can conclude that X is a stable model of $F \wedge G$ iff X is a stable model of $\bigwedge_{A \in X \cap S} A \ \wedge \ G$. ■

5 Conclusion

Many publications in the area of answer set programming are directed towards practical applications, and the titles of several papers of this kind included in the bibliography[14] show the remarkable diversity of the areas

[14] [Soininen and Niemelä, 1998], [Erdem *et al.*, 2000], [Nogueira *et al.*, 2001], [Heljanko and Niemelä, 2003], [Baral *et al.*, 2004], [Brooks *et al.*, 2005], [Leone *et al.*, 2005], [Hermansson *et al.*, 2005].

of science and technology where ASP may be useful. Success in this work would have been impossible without efficient, reliable, carefully crafted answer set solvers.

The main topic of this paper, however, is theoretical. We have seen that ASP is based on interesting mathematics, including some ideas developed in the early days of modern logic. The senior author (VL) is particularly pleased to contribute a paper on mathematical foundations of answer set programming to a volume in honor of Dov Gabbay in view of the important role that intuitionistic logic plays in this theory. Intuitionistic logic is what both of us were interested in as beginning researchers many years ago, when we first learned about each other's work.

A Propositional Logic

A.1 Syntax and Semantics

(Propositional) formulas are formed from propositional atoms and the 0-place connective $\bot$ using the binary connectives $\wedge$, $\vee$ and $\rightarrow$. We use

$$\begin{array}{rll} \top & \text{as shorthand for} & \bot \rightarrow \bot, \\ \neg F & \text{as shorthand for} & F \rightarrow \bot, \\ F \leftrightarrow G & \text{as shorthand for} & (F \rightarrow G) \wedge (G \rightarrow F). \end{array}$$

Atoms and negated atoms are called *literals*.

The relation $X \models F$ between a set X of atoms and a formula F is defined recursively:

- for an atom A, $X \models A$ if $A \in X$;
- $X \not\models \bot$;
- $X \models F \wedge G$ if $X \models F$ and $X \models G$;
- $X \models F \vee G$ if $X \models F$ or $X \models G$;
- $X \models F \rightarrow G$ if $X \not\models F$ or $X \models G$.

If $X \models F$ then we say that X *satisfies* F, or is a *model* of F. A formula is a *tautology* if it is satisfied by every set of atoms. A formula F is *equivalent* to a formula G if $F \leftrightarrow G$ is a tautology (or, equivalently, if F and G have the same models).

An occurrence of an atom A in a formula F is *positive* if the number of implications containing that occurrence in the antecedent is even, and *negative* otherwise. For instance, both occurrences of p in the formula

(63) $((p \rightarrow q) \wedge r) \rightarrow p$

are positive, and q, r are negative. An occurrence of an atom A in a formula F is *strictly positive* if it does not belong to the antecedent of any implication in F. For instance, the second occurrence of p in (63) is strictly positive, and the first is not. Since $\neg F$ is shorthand for $F \rightarrow \bot$, no occurrence of an atom in a formula of the form $\neg F$ can be strictly positive.

A.2 Logic of Here-and-There

The logic of here-and-there is a 3-valued logic that was originally proposed by the inventor of intuitionistic logic Arend Heyting as a technical tool for the purpose of proving that intuitionistic logic is weaker than classical [Heyting, 1930]. (He remarks that the truth values in his truth tables "can be interpreted as follows: 0 denotes a correct proposition, 1 denotes a false proposition, and 2 denotes a proposition that cannot be false but whose correctness is not proved.") We will identify a function from the set of atoms to the extended set of truth values $\{0, 1, 2\}$ with the ordered pair consisting of the set X of atoms that are mapped to 0 and the set Y of atoms that are mapped to 0 or 2. (If an atom belongs to X then it is true "here"; if an atom belongs to Y then it is true "there".)

An *HT-interpretation* is an ordered pair $\langle X, Y\rangle$ of sets of atoms such that $X \subseteq Y$. The *satisfaction* relation $\models$ between an HT-interpretation $\langle X, Y\rangle$ and a formula F is defined recursively:

- for an atom A, $\langle X, Y\rangle \models A$ if $A \in X$;
- $\langle X, Y\rangle \not\models \bot$;
- $\langle X, Y\rangle \models F \wedge G$ if $\langle X, Y\rangle \models F$ and $\langle X, Y\rangle \models G$;
- $\langle X, Y\rangle \models F \vee G$ if $\langle X, Y\rangle \models F$ or $\langle X, Y\rangle \models G$;
- $\langle X, Y\rangle \models F \rightarrow G$ if
  - (i) $\langle X, Y\rangle \not\models F$ or $\langle X, Y\rangle \models G$, and
  - (ii) $Y \models F \rightarrow G$.

(The symbol $\models$ in the last line refers to the satisfaction relation of classical logic defined in Section A.1.)

A formula is *valid in the logic of here-and-there* if it is satisfied by every HT-interpretation. A formula F is *equivalent* to a formula G *in the logic of here-and-there* if $F \leftrightarrow G$ is valid in the logic of here-and-there (or, equivalently, if F and G are satisfied by the same HT-interpretations).

The following facts relate the satisfaction relation of the logic of here-and-there to the satisfaction relation of classical logic:

(64) $\langle X, X\rangle \models F$ iff $X \models F$.

(65) If $\langle X, Y\rangle \models F$ then $Y \models F$.

(66) $\langle X, Y\rangle \models \neg F$ iff $Y \models \neg F$.

From property (64) we see that a formula can be valid in the logic of here-and-there only if it is a tautology. It follows that two formulas can be equivalent to each other in the logic of here-and-there only if they are classically equivalent. To see where the two equivalence relations differ from each other, note that $\neg\neg p$ is not equivalent to p in the logic of here-and-there. Indeed, by (66), the HT-interpretation $\langle \emptyset, \{p\}\rangle$ satisfies $\neg\neg p$, but it clearly does not satisfy p.

A.3 Natural Deduction

In the natural deduction system for propositional logic, the derivable objects are *sequents*—expressions of the form $\Gamma \Rightarrow F$ ("F under the assumptions Γ"), where F is a formula and Γ is a finite set of formulas. Notationally, we will identify the set of assumption in a sequent with the list of its elements. For instance, we will write $\Gamma, F \Rightarrow G$ for $\Gamma \cup \{F\} \Rightarrow G$.

The axiom schemas are

(67) $F \Rightarrow F$

and

(68) $\Rightarrow F \vee \neg F$.

The latter is called *the law of excluded middle.* The inference rules are

$$(\wedge I)\ \frac{\Gamma \Rightarrow F \quad \Delta \Rightarrow G}{\Gamma, \Delta \Rightarrow F \wedge G} \qquad (\wedge E)\ \frac{\Gamma \Rightarrow F \wedge G}{\Gamma \Rightarrow F} \quad \frac{\Gamma \Rightarrow F \wedge G}{\Gamma \Rightarrow G}$$

$$(\vee I)\ \frac{\Gamma \Rightarrow F}{\Gamma \Rightarrow F \vee G} \quad \frac{\Gamma \Rightarrow G}{\Gamma \Rightarrow F \vee G} \qquad (\vee E)\ \frac{\Gamma \Rightarrow F \vee G \quad \Delta_1, F \Rightarrow H \quad \Delta_2, G \Rightarrow H}{\Gamma, \Delta_1, \Delta_2 \Rightarrow H}$$

$$(\rightarrow I)\ \frac{\Gamma, F \Rightarrow G}{\Gamma \Rightarrow F \rightarrow G} \qquad (\rightarrow E)\ \frac{\Gamma \Rightarrow F \quad \Delta \Rightarrow F \rightarrow G}{\Gamma, \Delta \Rightarrow G}$$

$$(C)\ \frac{\Gamma \Rightarrow \bot}{\Gamma \Rightarrow F}$$

$$(W)\ \frac{\Gamma \Rightarrow F}{\Gamma' \Rightarrow F}$$

if $\Gamma \subseteq \Gamma'$

Among the first six inference rules, the rules in the left column are *introduction rules*, and the rules in the right column are *elimination rules.* Rule (C) is the *contradiction rule*, and (W) is *weakening.*

Since we defined $\neg F$ as an abbreviation for $F \to \bot$ (Section A.1), "negation introduction"

$$\frac{\Gamma, F \Rightarrow \bot}{\Gamma \Rightarrow \neg F}$$

is a special case of $(\to I)$, and "negation elimination"

$$\frac{\Gamma \Rightarrow F \quad \Delta \Rightarrow \neg F}{\Gamma, \Delta \Rightarrow \bot}$$

is a special case of $(\to E)$. Similarly, the introduction and elimination rules for equivalence

$$\frac{\Gamma \Rightarrow F \to G \quad \Delta \Rightarrow G \to F}{\Gamma, \Delta \Rightarrow F \leftrightarrow G} \qquad \frac{\Gamma \Rightarrow F \leftrightarrow G}{\Gamma \Rightarrow F \to G} \qquad \frac{\Gamma \Rightarrow F \leftrightarrow G}{\Gamma \Rightarrow G \to F}$$

are special cases of $(\wedge I)$ and $(\wedge E)$.

To prove a formula F in this system means to prove the sequent $\Rightarrow F$. For instance, here is a proof of the equivalence

(69) $(\neg p \to p) \leftrightarrow \neg\neg p.$

1.	$\neg p \to p \Rightarrow \neg p \to p$	— axiom
2.	$\neg p \Rightarrow \neg p$	— axiom
3.	$\neg p, \neg p \to p \Rightarrow p$	— by $(\to E)$ from 2, 1
4.	$\neg p, \neg p \to p \Rightarrow \bot$	— by $(\to E)$ from 3, 2
5.	$\neg p \to p \Rightarrow \neg\neg p$	— by $(\to I)$ from 4
6.	$\Rightarrow (\neg p \to p) \to \neg\neg p$	— by $(\to I)$ from 5
7.	$\neg\neg p \Rightarrow \neg\neg p$	— axiom
8.	$\neg p, \neg\neg p \Rightarrow \bot$	— by $(\to E)$ from 2, 7
9.	$\neg p, \neg\neg p \Rightarrow p$	— by (C) from 8
10.	$\neg\neg p \Rightarrow \neg p \to p$	— by $(\to I)$ from 9
11.	$\Rightarrow \neg\neg p \to (\neg p \to p)$	— by $(\to I)$ from 10
12.	$\Rightarrow (\neg p \to p) \leftrightarrow \neg\neg p$	— by $(\wedge I)$ from 6, 11

The deductive system described above is sound and complete: a formula F is provable in this system iff F is a tautology.

A formula is *intuitionistically provable* if it can be proved in this deductive system without references to axiom schema (68). A formula F is *intuitionistically equivalent* to a formula G if $F \leftrightarrow G$ is intuitionistically provable.

For instance, the implication $\neg p \rightarrow p$ is intuitionistically equivalent to the formula $\neg\neg p$, because the proof of (69) above contains no references to the law of excluded middle. On the other hand, this implication is not intuitionistically equivalent to p: the equivalence obtained from (69) by dropping the double negation in the right-hand side cannot be proved without (68).

According to the replacement property of intuitionistic logic, if F is a subformula of a formula F', and G' is obtained from F' by replacing an occurrence of F with another formula G, then $F' \leftrightarrow G'$ is intuitionistically derivable from $F \leftrightarrow G$. For instance, from the fact that $\neg p \rightarrow p$ is intuitionistically equivalent to $\neg\neg p$ we can conclude that $(\neg p \rightarrow p) \wedge q$ is intuitionistically equivalent to $\neg\neg p \wedge q$.

Every intuitionistically provable formula is valid in the logic of here-and-there; if two formulas are intuitionistically equivalent then they are equivalent in the logic of here-and-there. Moreover, these assertions remain true if, instead of intuitionistic logic, we talk about the stronger deductive system, obtained from classical by replacing (68) with the axiom schema expressing *the weak law of excluded middle:*

(70) $\Rightarrow \neg F \vee \neg\neg F.$

We can use this fact, for example, to check that the formulas $p \vee \neg p$ and $\neg\neg p \rightarrow p$ are equivalent to each other in the logic of here-and-there, as follows:

1.	$p \vee \neg p \Rightarrow p \vee \neg p$	— axiom
2.	$p \Rightarrow p$	— axiom
3.	$\neg p \Rightarrow \neg p$	— axiom
4.	$\neg\neg p \Rightarrow \neg\neg p$	— axiom
5.	$\neg p, \neg\neg p \Rightarrow \bot$	— by $(\rightarrow E)$ from 3, 4
6.	$\neg p, \neg\neg p \Rightarrow p$	— by (C) from 5
7.	$p \vee \neg p, \neg\neg p \Rightarrow p$	— by $(\vee E)$ from 1, 2, 6
8.	$p \vee \neg p \Rightarrow \neg\neg p \rightarrow p$	— by $(\rightarrow I)$ from 7
9.	$\Rightarrow (p \vee \neg p) \rightarrow (\neg\neg p \rightarrow p)$	— by $(\rightarrow I)$ from 8
10.	$\neg\neg p \rightarrow p \Rightarrow \neg\neg p \rightarrow p$	— axiom
11.	$\Rightarrow \neg p \vee \neg\neg p$	— axiom
12.	$\neg p \Rightarrow p \vee \neg p$	— by $(\vee I)$ from 3
13.	$\neg\neg p, \neg\neg p \rightarrow p \Rightarrow p$	— by $(\rightarrow E)$ from 4, 10
14.	$\neg\neg p, \neg\neg p \rightarrow p \Rightarrow p \vee \neg p$	— by $(\vee I)$ from 13
15.	$\neg\neg p \rightarrow p \Rightarrow p \vee \neg p$	— by $(\vee E)$ from 11, 12, 14
16.	$\Rightarrow (\neg\neg p \rightarrow p) \rightarrow (p \vee \neg p)$	— by $(\rightarrow I)$ from 15
17.	$\Rightarrow (p \vee \neg p) \leftrightarrow (\neg\neg p \rightarrow p)$	— by $(\wedge I)$ from 9, 16

Here is an axiom schema that is even stronger than (70) and that can be used for establishing the validity of formulas in the logic of here-and-there as well:

(71) $\Rightarrow F \vee (F \rightarrow G) \vee \neg G.$

Nothing stronger would be acceptable: A propositional formula is valid in the logic of here-and-there iff it is provable in the deductive system obtained from intuitionistic logic by adding axiom schema (71). This theorem is due to Lex Hendriks [Lifschitz *et al.*, 2001, Section 2.2].

B Traditional Definition of a Stable Model

In [Gelfond and Lifschitz, 1988], a logic program is assumed to consist of rules of the form

(72) $A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$

where $n \geq m \geq 0$ and $A_0, \ldots, A_n$ are atoms; we will call such expressions *traditional rules.* A finite set of traditional rules with $m = n$, that is, rules of the form

(73) $A_0 \leftarrow A_1, \ldots, A_m$

is essentially a Horn formula in the sense of Section 2.2.

The *traditional reduct* of a traditional program Π relative to a set X of atoms is the set of rules (73) for all rules (72) in Π such that

$$A_{m+1}, \ldots, A_n \notin X.$$

According to the 1988 definition, the stable model of a traditional program Π is a set X of atoms with the following property: *X is the minimal model of the traditional reduct of Π relative to X.* This is equivalent to our definition of a stable model (Section 2.1) limited to traditional programs:

PROPOSITION 28. *For any traditional program Π, a set X of atoms is the minimal model of the traditional reduct of Π relative to X iff X is a stable model of Π.*

Proof. Let $\Pi^{\underline{X}}$ denote the traditional reduct of Π relative to X.

Case 1: $X \not\models \Pi$. Set X is not a stable model of Π. On the other hand, Π contains a rule (72) such that $A_1, \ldots, A_m \in X$ and $A_0, A_{m+1}, \ldots, A_n \notin X$. The corresponding rule (73) in $\Pi^{\underline{X}}$ is not satisfied by X, so that X is not the minimal model of $\Pi^{\underline{X}}$.

Case 2: $X \models \Pi$. We will show that Π^X and $\Pi^{\underline{X}}$ are satisfied by the same subsets of X. Since Π^X is the conjunction of the formulas R^X for all rules R

of Π, and $\Pi^{\underline{X}}$ is the union of the programs $\{R\}^{\underline{X}}$ for all rules R of Π, it is sufficient to verify this claim for the case when Π is a single rule (72). If X contains at least one of the atoms $A_{m+1}, \ldots, A_n$ then $\Pi^{\underline{X}}$ is empty and Π^X is the tautology $\bot \rightarrow A_0^X$. Otherwise $\Pi^{\underline{X}}$ is (73). If $A_1, \ldots, A_m \in X$ then $A_0 \in X$, because $X \models \Pi$; consequently Π^X is the result of replacing $A_{m+1}, \ldots, A_n$ in (72) with $\bot$, which is equivalent to (73). It remains to consider the case when $A_{m+1}, \ldots, A_n \notin X$ and at least one of the atoms $A_1, \ldots, A_m$, say A_1, does not belong to X. In this case Π^X is the tautology $\bot \rightarrow A_0^X$. On the other hand, $\Pi^{\underline{X}}$ is the rule (73) whose body contains A_1 and consequently is not satisfied by any subset of X. It follows that every subset of X satisfies $\Pi^{\underline{X}}$. ∎

Intuitively, a rule (73) can be viewed as a rule for generating atoms: we are allowed to generate its head A_0 as soon as all atoms $A_1, \ldots, A_m$ in the body have been generated. The minimal model of a set of rules of the form (73) is the set of all atoms that can be generated by this process, starting from the empty set. The traditional definition of a stable model can be thought of as an extension of this idea to rules containing negative literals in the body. A rule (72) allows us to generate A_0 as soon as we generated the atoms $A_1, \ldots, A_m$ *provided that none of the atoms* $A_{m+1}, \ldots, A_n$ *can be generated using the rules of the program.* There is a vicious circle in this sentence: to decide whether a rule of Π can be used to generate a new atom, we need to know which atoms can be generated using the rules of Π. The traditional definition of a stable model overcomes this difficulty using a "fixpoint construction." Take a set X that you suspect may be exactly the set of atoms that can be generated using the rules of Π. Under this assumption, Π has the same meaning as the traditional reduct of Π relative to X, which is a set of rules of the form (73). Consider the minimal model of the traditional reduct. If this model is exactly identical to the set X that we started with then X was a "good guess"; it is indeed a stable model of Π.

The definition of a stable model for traditional programs can be viewed as a possible definition of a "correct" answer to a query in Prolog. Let Π be a Prolog program without variables (or the set of ground rules obtained from a Prolog program with variables by replacing each rule with all its ground instances). If Π is a traditional program with a unique stable model then the correct answer to a ground query A is *yes* or *no* depending on whether A belongs to that model.

From this perspective, a program with several stable models is "bad"—it does not provide an unambiguous specification for the behavior of a Prolog system. Programs without answer sets are "bad" also. In answer set programming, on the other hand, programs without a unique answer set are quite useful: they correspond to computational problems with many solutions, or with no solutions.

The concept of a stable model is only one of several available definitions of the semantics of negation as failure. Two other definitions frequently referred to in the literature are based on program completion [Clark, 1978] and the well-founded model [Van Gelder *et al.*, 1991]. These three definitions are not completely equivalent to each other, but each of them provides an adequate description of the behavior of Prolog.

Acknowledgements

This paper grew out of a graduate seminar that one of the authors has taught at the University of Texas and his course included in the program of the 2004 European Summer School on Logic, Language and Information. We are grateful to the students who took those classes for their feedback. We would like to thank also Esra Erdem, Artur Garcez, Joohyung Lee, Grigori Mints, Jayadev Misra and Hudson Turner for comments on a draft of this paper. This work was partially supported by the National Science Foundation under Grant IIS-0412907.

BIBLIOGRAPHY

[Baral *et al.*, 2004] Chitta Baral, Karen Chancellor, Nam Tran, Nhan Tran, Anna Joy, and Michael Berens. A knowledge based approach for representing and reasoning about cell signaling networks. In *Proceedings of European Conference on Computational Biology (ECCB), Supplement on Bioinformatics*, pages 15–22, 2004.

[Baral, 2003] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[Brooks *et al.*, 2005] Daniel R. Brooks, Esra Erdem, James W. Minett, and Donald Ringe. Character-based cladistics and answer set programming. In *Proceedings of International Symposium on Practical Aspects of Declarative Languages (PADL)*, pages 37–51, 2005.

[Clark, 1978] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[Davis *et al.*, 1962] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of ACM*, 5(7):394–397, 1962.

[Dimopoulos *et al.*, 1997] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in non-monotonic logic programs. In Sam Steel and Rachid Alami, editors, *Proceedings of European Conference on Planning*, pages 169–181. Springer-Verlag, 1997.

[Eiter and Gottlob, 1993] Thomas Eiter and Georg Gottlob. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In Dale Miller, editor, *Proceedings of International Logic Programming Symposium (ILPS)*, pages 266–278, 1993.

[Erdem *et al.*, 2000] Esra Erdem, Vladimir Lifschitz, and Martin Wong. Wire routing and satisfiability planning. In *Proceedings of International Conference on Computational Logic*, pages 822–836, 2000.

[Erdoğan and Lifschitz, 2004] Selim T. Erdoğan and Vladimir Lifschitz. Definitions in answer set programming. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Proceedings* *of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 114–126, 2004.

[Ferraris and Lifschitz, 2005] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5:45–74, 2005.

[Ferraris, 2005] Paolo Ferraris. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 2005. To appear.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080, 1988.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Gelfond, 1987] Michael Gelfond. On stratified autoepistemic theories. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 207–211, 1987.

[Hanks and McDermott, 1987] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[Heljanko and Niemelä, 2003] Keijo Heljanko and Ilkka Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3:519–550, 2003.

[Hermansson *et al.*, 2005] Martin Hermansson, Andreas Uphoff, Reijo Käkelä, and Pentti Somerharju. Automated quantitative analysis of complex lipidomes by liquid chromatography/mass spectrometry. *Analytical Chemistry*, 77:2166–2175, 2005.

[Heyting, 1930] Arend Heyting. Die formalen Regeln der intuitionistischen Logik. *Sitzunsberichte der Preussischen Akademie von Wissenschaften. Physikalisch-mathematische Klasse*, pages 42–56, 1930.

[Kautz and Selman, 1992] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, pages 359–363, 1992.

[Leibniz, 1995] Gottfried Wilhelm Leibniz. *Philosophical Writings*. Everyman, 1995.

[Leone *et al.*, 2005] Nicola Leone, Thomas Eiter, Wolfgang Faber, Michael Fink, Georg Gottlob, Gianluigi Greco, Giovambattista Ianni, Edyta Kalka, Domenico Lembo, Maurizio Lenzerini, Vincenzino Lio, Bartosz Nowicki, Riccardo Rosati, Marco Ruzzi, Witold Staniszkis, and Giorgio Terracina. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 915–917. ACM, 2005. Demo paper.

[Lifschitz and Turner, 1994] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proceedings of International Conference on Logic Programming (ICLP)*, pages 23–37, 1994.

[Lifschitz *et al.*, 1999] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

[Lifschitz *et al.*, 2001] Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.

[Lifschitz, 1999] Vladimir Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.

[Lifschitz, 2002] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138:39–54, 2002.

[Marek and Truszczyński, 1999] Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

[McCarthy, 1980] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39,171–172, 1980. Reproduced in [McCarthy, 1990].

[McCarthy, 1990] John McCarthy. *Formalizing Common Sense: Papers by John McCarthy*. Ablex, Norwood, NJ, 1990.

[McDermott and Doyle, 1980] Drew McDermott and Jon Doyle. Nonmonotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.

[Moore, 1985] Robert Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.

[Nelson, 1949] David Nelson. Constructible falsity. *Journal of Symbolic Logic*, 14:16–26, 1949.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[Nogueira *et al.*, 2001] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An A-Prolog decision support system for the Space Shuttle. In *Proceedings of International Symposium on Practical Aspects of Declarative Languages (PADL)*, pages 169–183, 2001.

[Pearce, 1997] David Pearce. A new logical characterization of stable models and answer sets. In Jürgen Dix, Luis Pereira, and Teodor Przymusinski, editors, *Non-Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 1216)*, pages 57–70. Springer-Verlag, 1997.

[Reiter, 1978] Raymond Reiter. On closed world data bases. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 119–140. Plenum Press, New York, 1978.

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Shanahan, 1997] Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

[Simons *et al.*, 2002] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.

[Soininen and Niemelä, 1998] Timo Soininen and Ilkka Niemelä. Developing a declarative rule language for applications in product configuration. In Gopal Gupta, editor, *Proceedings of International Symposium on Practical Aspects of Declarative Languages (PADL)*, pages 305–319. Springer-Verlag, 1998.

[Subrahmanian and Zaniolo, 1995] V.S. Subrahmanian and Carlo Zaniolo. Relating stable models and AI planning domains. In *Proceedings of International Conference on Logic Programming (ICLP)*, 1995.

[Turner, 1997] Hudson Turner. Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, 31:245–298, 1997.

[Turner, 2003] Hudson Turner. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 3(4,5):609–622, 2003.

[Van Gelder *et al.*, 1991] Allen Van Gelder, Kenneth Ross, and John Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.