Verification for ASP Denotational Semantics: a Case Study using the PVS Theorem Prover

F. Aguado[1], P. Ascariz[2], P. Cabalar[3], G. Pérez[4], C. Vidal[5]

[1,3,4,5]Department of Computer Science

University of Corunna, SPAIN.

{[1]aguado, [3]cabalar, [4]gperez, [5]concepcion.vidalm}@udc.es

[2]pfernandez@enaire.es

Abstract

In this paper we present an encoding of the (recently proposed) denotational semantics for Answer Set Programming (ASP) and its monotonic basis into the input language of the theorem prover PVS. Using some libraries and features from PVS, we have obtained semi-automated proofs for several fundamental properties of ASP. In this way, to the best of our knowledge, we provide the first known application of formal verification to ASP.

Keywords. Answer Set Programming, Theorem Proving, Formal Verification, PVS, Denotational Semantics

1 Introduction

Answer Set Programming (ASP)[4] constitutes nowadays one of the most successful paradigms of Knowledge Representation and Problem Solving in

Artificial Intelligence. The popularity of ASP is probably due to the availability of efficient solvers for hard computational problems, something that has allowed a boost in practical applications. But, together with this practical aspect, the success of ASP is also firmly supported by a constant evolution of its neat theoretical foundations, from its origins with the *stable models* semantics [5] for logic programs, until its full logical formalisation under *Equilibrium Logic* [7].

Equilibrium Logic is a non-monotonic formalism whose definition involves different types of models. For instance, *equilibrium models* of a theory Γ are defined by a kind of minimisation among models of Γ in the *Logic of Here-and-There* (HT) (an intermediate formalism between intuitionistic and classical logic). But at the same time, equilibrium models also happen to be a subset of the classical models of Γ. In this way, these three sets of models (classical, HT and equilibrium) frequently appear in papers about theoretical or fundamental properties of ASP.

Recently, a denotational semantics for ASP and Equilibrium Logic was proposed [2]. This denotational semantics allows characterising different sets of interpretations and models in a formal way, by just using several set operations. As a result, many meta-theorems proved in the literature in a textual or descriptive way become now formalisable in terms of standard set theory and partial order relations, opening the possibility of semi-automated proof checking and generation.

In this paper we explore that possibility: we provide the first known application of automated theorem proving for formal verification of ASP properties, to the best of our knowledge. In particular, we have implemented the basic definitions of the ASP denotational semantics in the language of the PVS (*Prototype Verification System*) theorem prover [6], using afterwards

some libraries and features of this prover to obtain certified proofs in an semi-automated way.

The rest of the paper is organised as follows. In the next section we begin recalling the basic features of PVS. After that, we start with the PVS formalisation of the ASP denotational semantics by introducing sets of partial interpretations. In Section 4, we describe the valuation of formulas in the logic of HT. Section 5 contains the PVS encoding of the denotational semantics and the main results of the paper. In Section 6 we provide some conclusions. Appendix A summarises the correspondence between mathematical symbols and PVS code and Appendix B contains an example of proof in PVS. The complete package of PVS files used for this work can be downloaded from [1].

2 Brief overview of PVS

PVS is a theorem prover developed and maintained by SRI [1] and is actively used in industrial applications, being also the "prover of choice" of NASA Langley Research Center, which actually maintains its own PVS library [2]. PVS provides an environment for constructing clear and precise specifications and for efficient mechanized verification. The distinguishing characteristics of PVS are its expressive specification language and its powerful theorem prover. The PVS specification language builds on classical typed higher-order logic with the usual base types, bool, nat, integer, real, among others, and the function type constructor (e.g., type $[A \to B]$ is the set of functions from set A to set B). Predicates are functions with range type bool. The type system of PVS also includes record types, dependent

[1] Public repository at `https://github.com/SRI-CSL/PVS`. Last commit 12/9/2016.
[2] `https://github.com/nasa/pvslib`

types, and abstract data types. Typechecking in this language requires the services of a theorem pover to discharge proof obligations corresponding to subtyping constraints.

PVS specifications are packaged as theories that can be parametric in types and constants. A collection, `prelude.pvs`, of theories and loadable libraries provide standard specifications and proved facts for a large number of theories. A theory can use the definitions and theorems of another theory by importing it.

The PVS environment has an automated theorem prover that provides a collection of powerful primitive inference procedures that are applied interactively under user guidance within a sequent calculus framework. The primitive inferences include propositional and quantifier rules, induction, rewriting, simplification using decision procedures for equality and linear arithmetic, data and predicate abstraction.

One of the main advantages of PVS with respect to other provers such as Coq, HOL, Isabelle, etc is that it allows the direct declaration of predicate subtypes. For instance:

```
bottom(S: set[I]): set[I] = {i: I | EXISTS (j: I):
    (member(j,S) AND R_ord2(i,j))}
```

All properties of the parent type are inherited by the subtype. A constraining predicate is provided to identify which elements are contained in the subset.

3 Partial interpretations

As a starting point, we will use the alternative characterisation of HT in terms of Gödel's three-valued logic G_3. In particular, in this section, we

concentrate on a set of definitions exclusively related to three-valued (or partial) interpretations and leave their use for valuation of formulas for the next section.

We start from a finite set of atoms Σ called the *propositional signature*. A *partial interpretation* is a mapping $v : \Sigma \to \{0, 1, 2\}$ assigning 0 (false), 2 (true) or 1 (undefined) to each atom p in the signature Σ. A partial interpretation v is said to be *classical (or total)* if $v(p) \neq 1$ for every atom p. We write $\mathcal{I}$ and $\mathcal{I}_c$ to stand for the set of all partial and total interpretations, respectively (fixing signature Σ). Note that $\mathcal{I}_c \subseteq \mathcal{I}$.

Before introducing the PVS encoding, it is worth to mention that the notation (mostly, the variable and function names) used in the PVS code throughout the paper is slightly different from the one we use in the text (which respects the original definitions in [2]). Table 1 shows the notation correspondence. Keeping that name mapping in mind, the PVS encoding of the above definitions is quite straightforward:

```
Sig_prop[T: TYPE+]: THEORY
BEGIN
ASSUMING
T_finite: ASSUMPTION is_finite_type[T]
ENDASSUMING

s3?(x:nat):bool = x <= 2
S3: TYPE = (s3?)
s2?(x:nat):bool = s3?(x) AND (x=0 OR x=2)
S2: TYPE = (s2?)
S3_cont_S2: JUDGEMENT S2 SUBTYPE_OF S3
I: TYPE+ = [T -> S3]
IC: TYPE = {i: I | FORALL(t: T): (i(t) = 0 OR i(t) = 2)}
I_cont_IC: JUDGEMENT IC SUBTYPE_OF I
```

Text Theory	**PVS**	**Denotes ...**	**Page**
Σ	`T`	the signature	5
p	`t`	an atom	5
$\{0,1,2\}$	`S3`	the set $\{0,1,2\}$	5
$\mathcal{I}$	`I`	the set of all partial interpretations	5
$\mathcal{I}_c$	`IC`	the set of all total interpretations	5
u, v	`i, j`	partial interpretations	5
v_t	`RT(i)`	the total interpretation associated to `i`	7
$v \leq u$	`R_ord(i,j: I)`	the order relation	7
$\overline{S}$	`comp(S)`	complementary	9
S_c	`classic(S)`	subset of classical interpretations	9
$S\downarrow$	`bottom(S)`	elements in S or below	9
$S\uparrow$	`top(S)`	elements in S or above	9
α, β	`s, t, t1, t2`	formulas	12
$\bot$	`bot`	falsum	12
$\alpha \vee \beta$	`op_or(t1,t2)`	disjunction	12
$\alpha \wedge \beta$	`op_and(t1,t2)`	conjunction	12
$\alpha \rightarrow \beta$	`op_imp(t1,t2)`	implication	12
$[\![\alpha]\!]$	`denotation(s)`	denotation of α	15

Table 1: Correspondence between notation used in the text and names used in the PVS code.

Given any partial interpretation $v \in \mathcal{I}$ we define a particular classical interpretation $v_t \in \mathcal{I}_c$ that, informally speaking, transforms 1's into 2's.

Formally:

$$v_t(p) \stackrel{\text{def}}{=} \begin{cases} 2 & \text{if } v(p) = 1 \\ v(p) & \text{otherwise} \end{cases}$$

Our PVS encoding represents v_t as the function `RT(i)`.

```
RT(i: I): IC =
    LAMBDA(t: T): IF i(t) = 0 THEN 0 ELSE 2 ENDIF
```

We will be interested in a particular ordering among partial interpretations that is defined as follows. Given two partial interpretations u, v, we say that $u \leq v$ when, for any atom $p \in \Sigma$, the following two conditions hold: $u(p) \leq v(p)$; and $u(p) = 0$ implies $v(p) = 0$. In other words, u and v must coincide in their 0's and an atom value in u must not be greater than its value in v. We encode this relation as `R_ord2` below:

```
R_ord2(i,j: I): bool =
  IF (FORALL (t: T): (i(t) <= j(t)) AND (i(t) = 0 IMPLIES j(t) = 0))
    THEN TRUE ELSE FALSE
  ENDIF
```

This relation can be equivalently characterised in terms of v_t as described below:

```
R_ord(i,j: I): bool =
    IF ((FORALL (t: T): (i(t) <= j(t))) AND RT(i) = RT(j)) THEN TRUE
    ELSE FALSE
    ENDIF
R_ord_same: LEMMA FORALL (i,j: I): (R_ord(i,j) IFF R_ord2(i,j))
```

We can use PVS to certify that $\leq$ is, indeed, a partiar order relation:

```
R_reflexive: LEMMA FORALL (i: I): R_ord(i,i)
R_antisymmetric: LEMMA FORALL (i,j: I):
```

```
    (R_ord(i,j) AND R_ord(j,i)) IMPLIES i = j
R_transitive: LEMMA FORALL (i,j,k: I):
    (R_ord(i,j) AND R_ord(j,k)) IMPLIES R_ord(i,k)
R_partial_order: LEMMA partial_order?[I](R_ord)
```

Moreover, we can easily check the following properties relating $\leq$ and v_t:

$$\forall v \in \mathcal{I} \;\; v \leq v_t \qquad \forall v, u \in \mathcal{I} \;\; \text{if } v \leq u \text{ then } v_t = u_t$$
$$\forall v \in \mathcal{I}_c \;\; v_t = v \qquad \forall v, u \in \mathcal{I} \;\; \text{if } v \leq u \text{ and } u \in \mathcal{I}_c \text{ then } v_t = u$$

```
RT_ord: LEMMA FORALL (i: I): R_ord(i,RT(i))
RT_classic: LEMMA FORALL (i: IC): RT(i) = i
RT_ord_mon: LEMMA FORALL (i,j: I): R_ord(i,j) IMPLIES RT(i) = RT(j)
RT_ord_uniq: LEMMA FORALL (i: I, j: IC): R_ord(i,j) IMPLIES RT(i) = j
```

In [2], an important group of constructions that became crucial for defining the denotational semantics were the following operations. Given a set of interpretations $S \subseteq \mathcal{I}$ we define:

$$\begin{aligned}
\overline{S} &\stackrel{\text{def}}{=} \{u \in \mathcal{I} \mid u \notin S\} \\
S_c &\stackrel{\text{def}}{=} \{u \in \mathcal{I}_c \mid u \in S\} = \mathcal{I}_c \cap S \\
S\downarrow &\stackrel{\text{def}}{=} \{u \in \mathcal{I} \mid \text{there exists } v \in S, v \geq u\} \\
S\uparrow &\stackrel{\text{def}}{=} \{u \in \mathcal{I} \mid \text{there exists } v \in S, v \leq u\}
\end{aligned}$$

To avoid too many parentheses, we will assume that $\downarrow$, $\uparrow$ and subindex c have more priority than standard set operations $\cup$, $\cap$ and $\setminus$. The implementation of these operations in PVS is also quite straightforward:

```
i: VAR I
ic: VAR IC
comp(S: set[I]): set[I] = {i: I | NOT member(i,S)}
classic(S: set[I]): set[IC] = {ic | member(ic,S)}
```

```
bottom(S: set[I]): set[I] =
   {i: I | EXISTS (j: I): (member(j,S) AND R_ord2(i,j))}
top(S: set[I]): set[I] =
   {i: I | EXISTS (j: I): (member(j,S) AND R_ord2(j,i))}
```

Although for the forthcoming results in the paper we have used some lemmas from standard set theory included in the PVS library `prelude.pvs`, we have also required some additional specific properties for the set operators we have just introduced. These useful properties are specified below:

Proposition 1 *For any* $X, Y \subseteq \mathcal{I}$,

$$(X_c\downarrow)_c = X_c$$

If $X \subseteq Y$ *then* $X\downarrow \subseteq Y\downarrow$

If $X \subseteq Y$ *then* $X\uparrow \subseteq Y\uparrow$

$$\begin{aligned} (X \cup Y)\downarrow &= X\downarrow \cup Y\downarrow \\ (X \cup Y)\uparrow &= X\uparrow \cup Y\uparrow \\ (X \cap Y)\uparrow &\subseteq X\uparrow \cap Y\uparrow \\ (X \cap Y)\downarrow &\subseteq X\downarrow \cap Y\downarrow \end{aligned}$$

```
classic_bottom_classic: LEMMA FORALL (X: set[I]):
    classic(bottom(classic(X))) = classic(X)
bottom_subset: LEMMA FORALL (X, Y: set[I],i: I):
    subset?(X,Y) IMPLIES subset?(bottom(X),bottom(Y))
top_subset: LEMMA FORALL (X, Y: set[I],i: I):
    subset?(X,Y) IMPLIES subset?(top(X),top(Y))
union_bottom: LEMMA FORALL (X,Y: set[I]):
    bottom(union(X,Y)) = union(bottom(X),bottom(Y))
union_top: LEMMA FORALL (X,Y: set[I]):
    top(union(X,Y)) = union(top(X),top(Y))
intersection_bottom: LEMMA FORALL (X,Y: set[I]):
    subset? (bottom(intersection(X,Y)), intersection(bottom(X),bottom(Y)))
intersection_top: LEMMA FORALL (X,Y: set[I]):
    subset? (top(intersection(X,Y)), intersection(top(X),top(Y)))
```

With these new operators we can formally express that v_t is the only classical interpretation greater or equal than v in the following way:

Proposition 2 *For any $v \in \mathcal{I}$, it holds that $\{v\}\uparrow_c = \{v_t\}$*

```
classic_top_uni: LEMMA FORALL (i: I):
    classic(top(singleton(i))) = singleton(RT(i))
```

Proposition 3 *For any interpretation v, $v \in (S_c)\downarrow \iff v_t \in S$.*

```
classic_total: LEMMA FORALL (S: set[I],i: I):
    (member(i,bottom(classic(S)))) IFF (member(RT(i),S))
```

A particularly interesting type of sets of interpretations are those S satisfying that $v_t \in S$ for any $v \in S$. When this happens, we say that S is *total-closed* or *classically closed*.

```
S: VAR set[I]
total_closed?(S): bool =  FORALL (i:I):
    (member(i,S)) IMPLIES (member(RT(i),S))
```

We can capture this property with any the following equivalent conditions:

Proposition 4 *The following assertions are equivalent:*

(i) S is total-closed

(ii) $S \subseteq S_c\downarrow$

(iii) $S\uparrow_c = S_c$

```
total_closed_1: LEMMA FORALL (S: set[I]):
    total_closed?(S) IMPLIES subset?(S, bottom(classic(S)))
total_closed_2: LEMMA FORALL (S: set[I]):
    subset?(S, bottom(classic(S))) IMPLIES classic(top(S))=classic(S)
```

```
total_closed_3: LEMMA FORALL (S: set[I]):
    classic(top(S))=classic(S) IMPLIES  total_closed?(S)
```

Proposition 5 *If S is a total-closed set of interpretations, then $(\overline{S})_c\downarrow \subseteq \overline{(S_c\downarrow)}$.*

```
bottom_classic_com_1: LEMMA FORALL (S: set[I],i: I):
    subset?(bottom(classic(comp(S))),comp(bottom(classic(S))))
```

Corollary 6 *If S is a total-closed set of interpretations, then $(\overline{S})_c\downarrow \subseteq \overline{S}$*

```
bottom_classic_com_2: LEMMA FORALL (S: set[I],i: I):
    total_closed?(S) IMPLIES subset?(bottom(classic(comp(S))),comp(S))
```

4 G_3 valuation of formulas

In this section we describe the valuation of formulas in Gödel's G_3 logic (equivalent to HT). We begin defining a *formula* α with the grammar:

$$\alpha ::= \bot \mid p \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \mid \alpha_1 \rightarrow \alpha_2$$

where α_1 and α_2 are formulae in their turn and $p \in \Sigma$ is any atom. We denote by $\neg\alpha \stackrel{\text{def}}{=} \alpha \rightarrow \bot$ and $\top \stackrel{\text{def}}{=} \neg\bot$. By $\mathcal{L}_\Sigma$ we denote the language of all well-formed formulae for signature Σ or just $\mathcal{L}$ when the signature is clear from the context.

The encoding of formulas in PVS uses the abstract data-type construct:

```
Sig_form[T: TYPE+]: DATATYPE
BEGIN

bot: bot?
```

```
atom(t: T): atom?
op_and(t1: Sig_form, t2: Sig_form): op_and?
op_or(t1: Sig_form, t2: Sig_form): op_or?
op_imp(t1: Sig_form, t2: Sig_form): op_imp?

END Sig_form
```

The `DATATYPE` construction in PVS provides a powerful tool for defining an abstract data type (ADT). To do so, we provide a set of constructors, accessors and recognizers. In this case, the constructors are `bot`, `atom`, etc, whereas the accessors are `t`, `t1` and `t2`. The recognizers are `bot?`, `atom?`, `op_and?`, etc. When the type checker is applied to an ADT three new theories are automatically created in a file `name_adt.pvs`. These theories provide the required axioms and induction principles to guarantee that the ADT conforms an algebra, defined by the constructors, that we can use as a starting point.

Given a partial interpretation $v \in \mathcal{I}$ we define a corresponding *valuation of formulas*, a function also named v (by abuse of notation) of type $v : \mathcal{L} \to \{0, 1, 2\}$ and defined as:

$$\begin{aligned}
v(\bot) &\stackrel{\text{def}}{=} 0 \\
v(\alpha \to \beta) &\stackrel{\text{def}}{=} \begin{cases} 2 & \text{if } v(\alpha) \leq v(\beta) \\ v(\beta) & \text{otherwise} \end{cases} \\
v(\alpha \wedge \beta) &\stackrel{\text{def}}{=} \min(v(\alpha), v(\beta)) \\
v(\alpha \vee \beta) &\stackrel{\text{def}}{=} \max(v(\alpha), v(\beta))
\end{aligned}$$

We say that v *satisfies* α when $v(\alpha) = 2$. We say that v is a *model* of a theory Γ iff v satisfies all the formulas in Γ.

The translation of G_3 valuation of formulas into PVS is as follows:

```
v_form(s: Sig_form,i: I): RECURSIVE S3 =
 CASES s OF
   bot: 0,
   atom(t): i(t),
   op_and(t1,t2): min(v_form(t1,i),v_form(t2,i)),
   op_or(t1,t2): max(v_form(t1,i),v_form(t2,i)),
   op_imp(t1,t2): IF v_form(t1,i) <= v_form(t2,i) THEN 2
                  ELSE v_form(t2,i) ENDIF
 ENDCASES
 MEASURE s BY <<
```

In PVS, all functions must be total. The `MEASURE` part describes a bound function for the recursive definition that must decrease in each recursive call. In this way, the type-checker can provide a related *termination* lemma that will lead to a set of Type Checking Conditions (TCC's) that in most cases must be proved manually by the specifier. In our particular case, the `MEASURE` function corresponds to `<<` that stands for the syntactic tree of term `s` (the formula) so that structural induction will be used.

We describe next three propositions that have been particularly useful as intermediate steps inside larger proofs.

Proposition 7 *For any $v \in \mathcal{I}$ and any $\alpha, \beta \in \mathcal{L}$,*

- $v(\alpha \wedge \beta) = 2 \Longleftrightarrow v(\alpha) = 2$ *and* $v(\beta) = 2$
- $v(\alpha \vee \beta) = 2 \Longleftrightarrow v(\alpha) = 2$ *or* $v(\beta) = 2$

```
caractv_form_and2: Lemma FORALL (t1,t2: Sig_form,i: I):
    v_form(op_and(t1,t2),i) = 2 IFF
    v_form(t1,i)=2 AND v_form(t2,i)=2
caractv_form_or2: Lemma FORALL (t1,t2: Sig_form,i: I):
```

```
    v_form(op_or(t1,t2),i) = 2 IFF
    v_form(t1,i)=2 OR v_form(t2,i)=2
```

Proposition 8 *For any $v \in \mathcal{I}$ and any $\alpha, \beta \in \mathcal{L}$,*

- $v(\alpha \to \beta) = 2 \Longleftrightarrow v(\alpha) = 0 \;\; or \;\; v(\beta) = 2 \;\; or \;\; v(\alpha) = 1 = v(\beta)$
- $v(\alpha \to \beta) = 0 \Longleftrightarrow v(\alpha) \neq 0 \;\; and \;\; v(\beta) = 0$
- $v(\alpha \to \beta) = 1 \Longleftrightarrow v(\alpha) = 2 \;\; and \;\; v(\beta) = 1$

```
caractv_form_imp2: Lemma FORALL (t1,t2: Sig_form,i: I):
    v_form(op_imp(t1,t2),i) = 2 IFF
    v_form(t1,i)=0 OR v_form(t2,i)=2 OR (v_form(t1,i)=1 AND v_form(t2,i)=1)
caractv_form_imp0: Lemma FORALL (t1,t2: Sig_form,i: I):
    v_form(op_imp(t1,t2),i) = 0 IFF
    not(v_form(t1,i)=0) AND v_form(t2,i)=0
caractv_form_imp1: Lemma FORALL (t1,t2: Sig_form,i: I):
    v_form(op_imp(t1,t2),i) = 1 IFF
    v_form(t1,i)=2 AND v_form(t2,i)=1
```

The following result has been proved in PVS by structural induction. The proof of this result is one of longest and most used among those obtained in this work.

Proposition 9 *For any $v \in \mathcal{I}$ and any formula $\alpha \in \mathcal{L}$,*

- $v(\alpha) = 0 \Longleftrightarrow v_t(\alpha) = 0$
- $v(\alpha) = 2 \Longrightarrow v_t(\alpha) = 2$
- $v(\alpha) \geq 1 \Longleftrightarrow v_t(\alpha) = 2$

```
caract_val_zero: Lemma FORALL (t: Sig_form,i: I):
    v_form(t,i)=0 IFF v_form(t,RT(i))=0
```

```
val_two: Lemma FORALL (t: Sig_form,i: I):
    v_form(t,i)=2 IMPLIES v_form(t,RT(i))=2
caract_val_nozero: Lemma FORALL (t: Sig_form,i: I):
    v_form(t,i)>=1 IFF v_form(t,RT(i))=2
```

5 Denotational Semantics

At this point, we can already provide an alternative semantics for propositional formulas in terms of sets of interpretations rather than using the G_3 valuation function. The basic idea in a denotational semantics [8] is that, for any formula α, its *denotation* $[\![\,\alpha\,]\!]$ is a set of interpretations representing all the *models* of α. In our case, we recursively define $[\![\,\alpha\,]\!]$ as follows:

$$\begin{aligned}
[\![\,\bot\,]\!] &\stackrel{\text{def}}{=} \emptyset \\
[\![\,p\,]\!] &\stackrel{\text{def}}{=} \{v \in \mathcal{I} \,:\, v(p) = 2\} \\
[\![\,\alpha \to \beta\,]\!] &\stackrel{\text{def}}{=} \left(\overline{[\![\,\alpha\,]\!]} \cup [\![\,\beta\,]\!]\right) \cap \left(\overline{[\![\,\alpha\,]\!]} \cup [\![\,\beta\,]\!]\right)_c\downarrow \\
[\![\,\alpha \wedge \beta\,]\!] &\stackrel{\text{def}}{=} [\![\,\alpha\,]\!] \cap [\![\,\beta\,]\!] \\
[\![\,\alpha \vee \beta\,]\!] &\stackrel{\text{def}}{=} [\![\,\alpha\,]\!] \cup [\![\,\beta\,]\!]
\end{aligned}$$

This recursive definition is captured in the PVS code below:

```
denotation(s: Sig_form): RECURSIVE set[I] =
    CASES s OF
      bot: emptyset,
      atom(t): {i: I | i(t) = 2},
      op_and(t1,t2): intersection(denotation(t1),denotation(t2)),
      op_or(t1,t2): union(denotation(t1),denotation(t2)),
      op_imp(t1,t2): intersection
        (union(comp(denotation(t1)),denotation(t2)),
        bottom(classic(union(comp(denotation(t1)),denotation(t2)))))
    ENDCASES
```

```
        MEASURE s BY <<
```

An important result is that the denotational of a formula α is, indeed, an equivalent reformulation of the set G_3 models of α. This is stated by the theorem below.

Theorem 10 *Let $v \in \mathcal{I}$ be a partial interpretation and $\alpha \in \mathcal{L}$ a formula. Then:*

$$v(\alpha) = 2 \text{ in } G_3 \text{ iff } v \in [\![\alpha]\!]$$

whose enunciate in PVS is as follows:

```
denot_charac: LEMMA FORALL (s: Sig_form, i: I):
    v_form(s,i)=2 IFF member(i,denotation(s))
```

From this, we can also conclude that:

Corollary 11 *For any $\alpha \in \mathcal{L}$, $[\![\alpha]\!]$ is total-closed.*

```
denotation_include: LEMMA FORALL (s: Sig_form,i: I):
    member(i,denotation(s)) IMPLIES member(RT(i),denotation(s))
denotation_total_closed: LEMMA FORALL (s: Sig_form,i:I):
    total_closed?(denotation(s))
```

Theorem 12 *Let $v \in \mathcal{I}$ be a partial interpretation and $\alpha \in \mathcal{L}$ a formula. Then:*

$$v(\alpha) \neq 0 \text{ in } G_3 \text{ iff } v_t \in [\![\alpha]\!]$$

```
denot_charac_0: LEMMA FORALL (s: Sig_form, i: I):
    NOT v_form(s,i) 0 IFF member(RT(i),denotation(s))
```

The next result is a kind of deduction theorem for the denotational semantics. Note that if a formula α has denotation $\mathcal{I}$, it means that α is a tautology.

Theorem 13 *For any pair of formulae* α, β*:* $[\![\alpha]\!] \subseteq [\![\beta]\!]$ *iff* $[\![\alpha \rightarrow \beta]\!] = \mathcal{I}$*. Moreover,* $[\![\alpha]\!] = [\![\beta]\!]$ *iff* $[\![\alpha \leftrightarrow \beta]\!] = \mathcal{I}$*.*

```
denotation_inclusion: LEMMA FORALL (t1,t2: Sig_form):
    subset?(denotation(t1),denotation(t2)) IFF
    (FORALL (i:I): member (i,denotation(op_imp(t1,t2))))
denotation_equal: LEMMA FORALL (t1,t2: Sig_form):
    denotation(t1) = denotation(t2) IFF
    (FORALL (i:I):
     member (i,denotation(op_and(op_imp(t1,t2),op_imp(t2,t1)))))
```

The following result proves that implication verifies monotonicity for the consequent and anti-monotonicity for the antecedent.

Proposition 14 *For any* $\alpha, \beta, \gamma \in \mathcal{L}$

1. $[\![\alpha]\!] \subseteq [\![\beta]\!]$ *implies* $[\![\gamma \rightarrow \alpha]\!] \subseteq [\![\gamma \rightarrow \beta]\!]$
2. $[\![\alpha]\!] \subseteq [\![\beta]\!]$ *implies* $[\![\beta \rightarrow \gamma]\!] \subseteq [\![\alpha \rightarrow \gamma]\!]$

The encoding in PVS is also quite natural, as shown next:

```
mon_imp_left: LEMMA FORALL (t1,t2,s: Sig_form,i: I):
    subset? (denotation(t1),denotation(t2)) IMPLIES
    subset? (denotation(op_imp(s,t1)),denotation(op_imp(s,t2)))
mon_imp_right: LEMMA FORALL (t1,t2,s: Sig_form,i: I):
    subset? (denotation(t1),denotation(t2)) IMPLIES
    subset? (denotation(op_imp(t2,s)),denotation(op_imp(t1,s)))
```

One interesting result obtained in [2] is that implication can be alternative characterised as a *union* of three different sets of models:

Proposition 15 *For any* $\alpha, \beta \in \mathcal{L}$ *it follows that:*

$$[\![\alpha \rightarrow \beta]\!] = \overline{[\![\alpha]\!]}_c \downarrow \cup \left(\overline{[\![\alpha]\!]} \cap [\![\beta]\!]_c \downarrow \right) \cup [\![\beta]\!]$$

```
denotation_imp: LEMMA FORALL (t1,t2: Sig_form,i: I):
denotation(op_imp(t1,t2)):
union(union(
    bottom(classic(comp(denotation(t1)))),
    intersection(comp(denotation(t1)),bottom(classic(denotation(t2)))),
    denotation(t2))
```

This has some interesting consequences such as, for instance:

Corollary 16 *For any $\alpha, \beta \in \mathcal{L}$, it follows that:*

1. $[\![\beta]\!] \subseteq [\![\alpha \to \beta]\!]$
2. $[\![\alpha \to \beta]\!] \subseteq \overline{[\![\alpha]\!]} \cup [\![\beta]\!]$
3. $[\![\alpha \to \beta]\!] \cap [\![\alpha]\!] \subseteq [\![\beta]\!]$

```
denotation_imp_sub_r: LEMMA FORALL (t1,t2: Sig_form,i: I):
    subset?(denotation(t2),denotation(op_imp(t1,t2)))
denotation_imp_inc_1: LEMMA FORALL (t1,t2: Sig_form,i: I):
    subset?(denotation(op_imp(t1,t2)),
            union(comp(denotation(t1)),denotation(t2)))
denotation_imp_inc_2: LEMMA FORALL (t1,t2: Sig_form,i: I):
    subset?(intersection(denotation(op_imp(t1,t2)),denotation(t1)),
            denotation(t2))
```

The following result proves that disjunction in G_3 can be represented in terms of the other operators:

Theorem 17 *For any Σ, the system $\mathcal{L}_\Sigma\{\bot, \wedge, \to\}$ is complete because given any pair of formulas α, β for Σ, it holds that:*

$$[\![\alpha \vee \beta]\!] = [\![(\alpha \to \beta) \to \beta]\!] \cap [\![(\beta \to \alpha) \to \alpha]\!].$$

```
denotation_or: LEMMA FORALL (t1,t2: Sig_form,i: I):
    denotation(op_or(t1,t2)) =
    intersection(denotation(op_imp(op_imp(t1,t2),t2)),
                 denotation(op_imp(op_imp(t2,t1),t1)))
```

Finally, we include a characterisation of models for negation:

Proposition 18 *For any formula α:*

1. $[\![\neg\alpha]\!] = \overline{[\![\alpha]\!]}_c\downarrow$
2. *For any partial interpretation v, $v \in [\![\neg\alpha]\!]$ iff $v_t \in \overline{[\![\alpha]\!]}$*

```
denotation_not_charact: LEMMA FORALL (s: Sig_form, i: I):
    denotation(op_imp(s,bot)) = bottom(classic(comp(denotation(s))))
denotation_not: LEMMA FORALL (s: Sig_form, i: I):
    member(i,denotation(op_imp(s,bot))) IFF
    member(RT(i),comp(denotation(s)))
```

6 Conclusions

In this paper we have provided a specification of the denotational semantics for Answer Set Programming (ASP) in the language of the theorem prover PVS. As a result, we have been able to provide computer-checked proofs for several fundamental properties of ASP denotational semantics, constituting the first case of automated formal verification for this approach.

From the perspective of ASP, we have certified the correctness of some theoretical properties (metatheorems) that are usually proved by hand (and thus, more prune to errors) opening the possibility to explore new semantic relations with the guarantee of a formal system for checking their soundness. We have obtained a guided generation of proofs leading to more confidence

on correctness. Thanks to the use of a semi-automated prover, we have also detected some cases of hypotheses that had been introduced in some results in [2] and that PVS has revealed to be unnecessary. From the PVS perspective, the obtained theory is medium-sized: it currently comprises 113 proofs, but only a part of [2] has been certified and the rest is still under development. The most interesting result with respect to the theorem prover is perhaps that we have detected some properties from standard set theory that were missing in the PVS set library and became frequently useful in most proofs. For future work, we plan to include these properties in the set library and, furthermore, design new strategies [3] that help to automate some repetitive work we have also detected in the proof generation.

Acknowledgements We wish to thank César A. Muñoz for his support and expert guidance with non-trivial features of the PVS theorem prover. This research was partially supported by Spanish MEC project TIN2013-42149-P.

References

[1] Felicidad Aguado, Pablo Ascariz, Pedro Cabalar, Gilberto Pérez, and Concepción Vidal. PVS files for ASP denotational semantics, September 2015.
`https://github.com/nasa/pvslib/tree/master/ASP`.

[2] Felicidad Aguado, Pedro Cabalar, David Pearce, Gilberto Pérez, and Concepción Vidal. A denotational semantics for equilibrium logic. *Theory and Practice of Logic Programming*, 15(4-5):620–634, 2015.

[3] M. Archer, B. D. Vito, and C. Muñoz. Developing user strategies in PVS: A tutorial. In *Proceedings of Design and Application of Strategies/Tactics in Higher Order Logics (STRATA'03)*, Hampton VA 23681-2199, USA, 2003. NASA LaRC. NASA/CP-2003-212448.

[4] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[5] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *Logic Programming: Proc. of the Fifth International Conference and Symposium (Volume 2)*, pages 1070–1080. MIT Press, Cambridge, MA, 1988.

[6] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[7] David Pearce. A new logical characterisation of stable models and answer sets. In *Non monotonic extensions of logic programming. Proc. NMELP'96. (LNAI 1216)*. Springer-Verlag, 1996.

[8] Dana Scott and Christopher Strachey. Toward a mathematical semantics for computer languages. Technical Report PRG-6, Oxford Programming Research Group Technical Monograph, 1971.

Appendix A. An example of PVS proof

As an example of proof development in PVS, we will partly show the proof of Theorem 10 we reproduce below.

Theorem 10. Let $v \in \mathcal{I}$ be a partial interpretation and $\alpha \in \mathcal{L}$ a formula. Then:

$$v(\alpha) = 2 \text{ in } G_3 \text{ iff } v \in [\![\alpha]\!]$$

```
denot_charac: LEMMA FORALL (s: Sig_form, i: I):
    v_form(s,i) = 2 IFF member(i,denotation(s))
```

The proof of this result is made by induction. The command to invoke an inductive proof is `induct`, that in this case generates 5 subgoals corresponding to the 5 constructors of type `Sig_fom` (formulas). The first subgoal corresponds to the $\bot$ constant:

```
Rule? (induct "s")
Inducting on s on formula 1,
this yields  5 subgoals:
denot_charac.1 :


  |-------
{1}   FORALL (i: I): v_form(bot, i) = 2 IFF member(i, denotation(bot))
```

To prove this subgoal, we apply skolemization, expand the definition of `v_form` and we finally simplify. Then, the system proceeds to show the second subgoal (atoms).

```
This completes the proof of denot_charac.1.


denot_charac.2 :


  |-------
```

```
{1}   FORALL (atom1_var: T):
        FORALL (i: I):
          v_form(atom(atom1_var), i) = 2 IFF
           member(i, denotation(atom(atom1_var)))
```

The proof is analogous to the previous case. For the subgoal corresponding to the $\wedge$ operator, the system provides the induction hypotheses:

```
This completes the proof of denot_charac.2.


denot_charac.3 :


  |-------
{1}   FORALL (op_and1_var: Sig_form[T], op_and2_var: Sig_form[T]):
        ((FORALL (i: I):
            v_form(op_and1_var, i) = 2 IFF
             member(i, denotation(op_and1_var)))
          AND
          (FORALL (i: I):
             v_form(op_and2_var, i) = 2 IFF
              member(i, denotation(op_and2_var))))
         IMPLIES
         (FORALL (i: I):
            v_form(op_and(op_and1_var, op_and2_var), i) = 2 IFF
             member(i, denotation(op_and(op_and1_var, op_and2_var))))
```

After applying skolemization and instantiation on interpretation `i`, we obtain the following, where `[-1]` and `{-2}` correspond to the induction hypothesis:

```
denot_charac.3 :


[-1]  v_form(op_and1_var, i) = 2 IFF member(i, denotation(op_and1_var))
{-2}  v_form(op_and2_var, i) = 2 IFF member(i, denotation(op_and2_var))
```

```
  |-------
[1]   v_form(op_and(op_and1_var, op_and2_var), i) = 2 IFF
       member(i, denotation(op_and(op_and1_var, op_and2_var)))
```

At this point, a propositional simplification is applied and the proof is divided in two branches, one for each direction of the biconditional, which turn out to be the subgoals 3.1 and 3.2:

```
Rule? (split)
Splitting conjunctions,
this yields  2 subgoals:
denot_charac.3.1 :

[-1]  v_form(op_and1_var, i) = 2 IFF member(i, denotation(op_and1_var))
[-2]  v_form(op_and2_var, i) = 2 IFF member(i, denotation(op_and2_var))
  |-------
{1}   v_form(op_and(op_and1_var, op_and2_var), i) = 2 IMPLIES
       member(i, denotation(op_and(op_and1_var, op_and2_var)))
```

Once some propositional simplifications are applied, we expand the definitions `member` and `denotation` and we apply Lemma `caractv_form_and2` (Proposition 7) to obtain:

```
denot_charac.3.1 :

{-1}  FORALL (t1, t2: Sig_form[T], i: I[T]):
        v_form(op_and(t1, t2), i) = 2 IFF
         v_form(t1, i) = 2 AND v_form(t2, i) = 2
[-2]  v_form(op_and(op_and1_var, op_and2_var), i) = 2
[-3]  v_form(op_and1_var, i) = 2 IMPLIES denotation(op_and1_var)(i)
[-4]  denotation(op_and1_var)(i) IMPLIES v_form(op_and1_var, i) = 2
[-5]  v_form(op_and2_var, i) = 2 IMPLIES denotation(op_and2_var)(i)
[-6]  denotation(op_and2_var)(i) IMPLIES v_form(op_and2_var, i) = 2
```

```
  |-------
[1]   intersection(denotation(op_and1_var), denotation(op_and2_var))(i)
```

The proof of the subgoal 3.1 is completed by instantiation of the lemma with the components `op_and1_var` and `op_and2_var`, making some simplifications afterwards. Then, the system shows the subgoal 3.2:

```
This completes the proof of denot_charac.3.1.


denot_charac.3.2 :


[-1]  v_form(op_and1_var, i) = 2 IFF member(i, denotation(op_and1_var))
[-2]  v_form(op_and2_var, i) = 2 IFF member(i, denotation(op_and2_var))
  |-------
{1}   member(i, denotation(op_and(op_and1_var, op_and2_var))) IMPLIES
       v_form(op_and(op_and1_var, op_and2_var), i) = 2
```

Once this proof of subgoal 3 is complete, the system displays the next goal, corresponding to the $\vee$ operator.

```
This completes the proof of denot_charac.3.


denot_charac.4 :


  |-------
{1}   FORALL (op_or1_var: Sig_form[T], op_or2_var: Sig_form[T]):
        ((FORALL (i: I):
            v_form(op_or1_var, i) = 2 IFF
             member(i, denotation(op_or1_var)))
          AND
          (FORALL (i: I):
             v_form(op_or2_var, i) = 2 IFF
              member(i, denotation(op_or2_var))))
```

```
        IMPLIES
        (FORALL (i: I):
           v_form(op_or(op_or1_var, op_or2_var), i) = 2 IFF
            member(i, denotation(op_or(op_or1_var, op_or2_var))))
```

The proof of the fourth subgoal (disjunction) follows the same pattern than the previous one, applying in this case Lemma `caractv_for_or2`.

```
This completes the proof of denot_charac.4.


denot_charac.5 :


  |-------
{1}   FORALL (op_imp1_var: Sig_form[T], op_imp2_var: Sig_form[T]):
        ((FORALL (i: I):
            v_form(op_imp1_var, i) = 2 IFF
             member(i, denotation(op_imp1_var)))
          AND
          (FORALL (i: I):
             v_form(op_imp2_var, i) = 2 IFF
              member(i, denotation(op_imp2_var))))
         IMPLIES
         (FORALL (i: I):
            v_form(op_imp(op_imp1_var, op_imp2_var), i) = 2 IFF
             member(i, denotation(op_imp(op_imp1_var, op_imp2_var))))
```

Once we apply skolemization, the proof is divided into two branches, one for each direction of the biconditional. The first direction would correspond to $v(\alpha \to \beta) = 2 \Longrightarrow v \in [\![\, \alpha \to \beta \,]\!]$:

```
denot_charac.5.1 :


[-1]  FORALL (i: I):
        v_form(op_imp1_var, i) = 2 IFF member(i, denotation(op_imp1_var))
```

```
[-2]  FORALL (i: I):
        v_form(op_imp2_var, i) = 2 IFF member(i, denotation(op_imp2_var))
  |-------
{1}   v_form(op_imp(op_imp1_var, op_imp2_var), i) = 2 IMPLIES
       member(i, denotation(op_imp(op_imp1_var, op_imp2_var)))
```

We expand the definition of denotation for the conditional, $[\![\alpha \to \beta]\!] = (\overline{[\![\alpha]\!]} \cup [\![\beta]\!]) \cap (\overline{[\![\alpha]\!]} \cup [\![\beta]\!])_c\downarrow$, and the set intersection:

```
denot_charac.5.1 :


[-1]  v_form(op_imp(op_imp1_var, op_imp2_var), i) = 2
[-2]  FORALL (i: I):
        v_form(op_imp1_var, i) = 2 IFF denotation(op_imp1_var)(i)
[-3]  FORALL (i: I):
        v_form(op_imp2_var, i) = 2 IFF denotation(op_imp2_var)(i)
  |-------
{1}   member(i,
             union(comp(denotation(op_imp1_var)), denotation(op_imp2_var)))
       AND
       member(i,
              bottom(extend[I[T], IC[T], bool, FALSE]
                          (classic(union(comp(denotation(op_imp1_var)),
                                         denotation(op_imp2_var))))))
```

With the command `split`, subgoal 5.1, $v \in (\overline{[\![\alpha]\!]} \cup [\![\beta]\!]) \cap (\overline{[\![\alpha]\!]} \cup [\![\beta]\!])_c\downarrow$, is unfolded into 5.1.1, $v \in \overline{[\![\alpha]\!]} \cup [\![\beta]\!]$, and 5.1.2., $v \in (\overline{[\![\alpha]\!]} \cup [\![\beta]\!])_c\downarrow$. We begin proving membership in $\overline{[\![\alpha]\!]} \cup [\![\beta]\!]$.

```
Rule? (split)
Splitting conjunctions,
this yields  2 subgoals:
denot_charac.5.1.1 :
```

```
[-1]  v_form(op_imp(op_imp1_var, op_imp2_var), i) = 2
[-2]  FORALL (i: I):
        v_form(op_imp1_var, i) = 2 IFF denotation(op_imp1_var)(i)
[-3]  FORALL (i: I):
        v_form(op_imp2_var, i) = 2 IFF denotation(op_imp2_var)(i)
  |-------
{1}   member(i,
             union(comp(denotation(op_imp1_var)), denotation(op_imp2_var)))
```

Since this set is a union, we expand the definition of union and apply lemma `caractv_form_imp2` (Proposition 8) instantiated on `op_imp1_var` and `op_imp2_var`, simplifying afterwards:

```
denot_charac.5.1.1 :


{-1}  v_form(op_imp(op_imp1_var, op_imp2_var), i) = 2 IFF
       v_form(op_imp1_var, i) = 0 OR
        v_form(op_imp2_var, i) = 2 OR
         (v_form(op_imp1_var, i) = 1 AND v_form(op_imp2_var, i) = 1)
[-2]  v_form(op_imp(op_imp1_var, op_imp2_var), i) = 2
[-3]  FORALL (i: I):
        v_form(op_imp1_var, i) = 2 IFF denotation(op_imp1_var)(i)
[-4]  FORALL (i: I):
        v_form(op_imp2_var, i) = 2 IFF denotation(op_imp2_var)(i)
  |-------
{1}   member(i, comp(denotation(op_imp1_var)))
{2}   member(i, denotation(op_imp2_var))
```

From hypotheses `{-1}` and `[-2]` the proof is divided into three branches: 5.1.1.1 for the case $v(\alpha) = 0$, 5.1.1.2 when $v(\beta) = 2$ and 5.1.1.3 for the case $v(\alpha) = 1 = v(\beta)$. In each one of these three cases we must prove that $v \in \overline{[\![\, \alpha \,]\!]}$ or $v \in [\![\, \beta \,]\!]$. After simplifying, we get:

```
denot_charac.5.1.1.1 :
```

```
[-1]  v_form(op_imp1_var, i) = 0
{-3}  v_form(op_imp1_var, i) = 2 IFF denotation(op_imp1_var)(i)
  |-------
[1]   member(i, comp(denotation(op_imp1_var)))
[2]   member(i, denotation(op_imp2_var))
```

```
denot_charac.5.1.1.2 :

[-1]  v_form(op_imp2_var, i) = 2
{-2}  v_form(op_imp2_var, i) = 2 IFF denotation(op_imp2_var)(i)
  |-------
[1]   member(i, comp(denotation(op_imp1_var)))
[2]   member(i, denotation(op_imp2_var))
```

```
denot_charac.5.1.1.3 :

[-1]  (v_form(op_imp1_var, i) = 1 AND v_form(op_imp2_var, i) = 1)
{-2}  v_form(op_imp1_var, i) = 2 IFF denotation(op_imp1_var)(i)
  |-------
[1]   member(i, comp(denotation(op_imp1_var)))
[2]   member(i, denotation(op_imp2_var))
```

We prove that, in the first and third cases, v belongs to $\overline{[\![\,\alpha\,]\!]}$, whereas in the second case we show that it belongs to $[\![\,\beta\,]\!]$. In its turn, the proof for 5.1.2, $v \in \left(\overline{[\![\,\alpha\,]\!]} \cup [\![\,\beta\,]\!]\right)_c \downarrow$, is also divided into three parts; it is the longest and most complex proof, since we additionally have to apply the properties obtained for the operators `bottom` and `classic`. The rest of the proof, subgoal 5.2, $v \in [\![\,\alpha \to \beta\,]\!] \Longrightarrow v(\alpha \to \beta) = 2$, is tedious and extends for several pages. It implies the use of the following lemmas (some of them several times):

- caractv_form_imp2
- caract_val_zero
- RT_ord
- val_two
- classic_total
- caract_val_nozero

A complete description can be found in the PVS files in [1].