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

F. Aguado\textsuperscript{1}, P. Ascariz\textsuperscript{2}, P. Cabalar\textsuperscript{3}, G. Pérez\textsuperscript{4}, C. Vidal\textsuperscript{5}

\textsuperscript{1,3,4,5}Department of Computer Science
University of Corunna, SPAIN.
\{aguado, cabalar, gperez, concepcion.vidalm\}@udc.es
\textsuperscript{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)\textsuperscript{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 \( \Gamma \) are defined by a kind of minimisation among models of \( \Gamma \) 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 \( \Gamma \). 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 par-
tial 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 math-
ematical 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 ac-
tively used in industrial applications, being also the “prover of choice” of
NASA Langley Research Center, which actually maintains its own PVS li-
brary 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 pow-
erful 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

²https://github.com/nasa/pvslib
types, and abstract data types. Typechecking in this language requires the services of a theorem prover 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:

\[
\text{bottom}(S: \text{set}[I]): \text{set}[I] = \{i: I \mid \text{EXISTS} (j: I): (\text{member}(j, S) \text{ AND } R\text{ord}2(i, j))\}
\]

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

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
```
The signature \( \Sigma \)

An atom \( p \)

The set \( \{0,1,2\} \)

The set of all partial interpretations \( \mathcal{I} \)

The set of all total interpretations \( \mathcal{I}_c \)

Partial interpretations \( u, v \) and \( i, j \)

The total interpretation associated to \( i \) \( \nu_i \)

The order relation \( \nu \leq u \) \( R_\text{ord}(i,j: I) \)

The set of all partial interpretations \( S \)

The complementary of \( S \) \( \text{comp}(S) \)

The subset of classical interpretations \( S_c \)

Elements in \( S \) or below \( \text{bottom}(S) \)

Elements in \( S \) or above \( \text{top}(S) \)

Formulas \( \alpha, \beta \)

Falsum \( \bot \)

Disjunction \( \alpha \lor \beta \)

Conjunction \( \alpha \land \beta \)

Implication \( \alpha \rightarrow \beta \)

Denotation(s) \( \{\alpha\} \)

Denotes...

Page

5

5

5

5

5

5

7

7

9

9

9

12

12

12

12

15
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:

\[
v_t(p) \overset{\text{def}}{=} \begin{cases} 2 & \text{if } v(p) = 1 \\ v(p) & \text{otherwise} \end{cases}
\]

Our PVS encoding represents \( v_t \) as the function \( \text{RT}(i) \).

\[
\text{RT}(i: \mathcal{I}): \mathcal{I}C = \\
\lambda(t: T): \text{IF } i(t) = 0 \text{ THEN } 0 \text{ ELSE } 2 \text{ ENDIF}
\]

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 \( R_{\text{ord2}} \) below:

\[
R_{\text{ord2}}(i,j: \mathcal{I}): \text{bool} = \\
\text{IF (FORALL (t: T): (i(t) \leq j(t)) AND (i(t) = 0 IMPLIES j(t) = 0))} \\
\text{THEN TRUE ELSE FALSE} \\
\text{ENDIF}
\]

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

\[
R_{\text{ord}}(i,j: \mathcal{I}): \text{bool} = \\
\text{IF ((FORALL (t: T): (i(t) \leq j(t))) AND RT(i) = RT(j)) THEN TRUE} \\
\text{ELSE FALSE} \\
\text{ENDIF}
\]

\[R_{\text{ord\_same}}: \text{LEMMA FORALL (i,j: \mathcal{I}): (R_{\text{ord}}(i,j) IFF R_{\text{ord2}}(i,j))}\]

We can use PVS to certify that \( \leq \) is, indeed, a partial order relation:

\[
R_{\text{reflexive}}: \text{LEMMA FORALL (i: \mathcal{I}): R_{\text{ord}}(i,i)}
\]

\[
R_{\text{antisymmetric}}: \text{LEMMA FORALL (i,j: \mathcal{I})}: 
\]
(R\text{ord}(i,j) \text{ AND } R\text{ord}(j,i)) \implies i = j

R\text{transitive}: \text{LEMMA}\ \forall (i,j,k: I): (R\text{ord}(i,j) \text{ AND } R\text{ord}(j,k)) \implies R\text{ord}(i,k)

R\text{partial}\_\text{order}: \text{LEMMA}\ partial\_\text{order}\[I\](R\text{ord})

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

\[
\forall v \in \mathcal{I} \ v \leq v_t \quad \forall v, u \in \mathcal{I} \text{ if } v \leq u \text{ then } v_t = u_t
\]

\[
\forall v \in \mathcal{I}_c \ v_t = v \quad \forall v, u \in \mathcal{I} \text{ if } v \leq u \text{ and } u \in \mathcal{I}_c \text{ then } v_t = u
\]

RT\text{ord}: \text{LEMMA}\ \forall (i: I): R\text{ord}(i,\text{RT}(i))

RT\text{classic}: \text{LEMMA}\ \forall (i: IC): \text{RT}(i) = i

RT\text{ord}\_\text{mon}: \text{LEMMA}\ \forall (i, j: I): R\text{ord}(i,j) \implies \text{RT}(i) = \text{RT}(j)

RT\text{ord}\_\text{uniq}: \text{LEMMA}\ \forall (i: I, j: IC): R\text{ord}(i,j) \implies \text{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:

\[
\overline{S} \overset{\text{def}}{=} \{u \in \mathcal{I} \mid u \notin S\}
\]

\[
S_c \overset{\text{def}}{=} \{u \in \mathcal{I}_c \mid u \in S\} = \mathcal{I}_c \cap S
\]

\[
S \downarrow \overset{\text{def}}{=} \{u \in \mathcal{I} \mid \text{there exists } v \in S, v \geq u\}
\]

\[
S \uparrow \overset{\text{def}}{=} \{u \in \mathcal{I} \mid \text{there exists } v \in S, v \leq u\}
\]

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:

\begin{align*}
i &:: \text{VAR } I \\
\text{ic} &:: \text{VAR } IC \\
\text{comp}(S: \text{set}[I]): \text{set}[I] &= \{i: I \mid \text{NOT member}(i,S)\} \\
\text{classic}(S: \text{set}[I]): \text{set}[IC] &= \{\text{ic} \mid \text{member}(\text{ic},S)\}
\end{align*}
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 \( X, Y \subseteq I \),

\[
\begin{align*}
(X_c)_c &= X_c \\
\text{If } X \subseteq Y \text{ then } X \downarrow &\subseteq Y \downarrow \\
\text{If } X \subseteq Y \text{ then } X \uparrow &\subseteq Y \uparrow \\
(X \cup Y)_\downarrow &= X \downarrow \cup Y \downarrow \\
(X \cup Y)_\uparrow &= X \uparrow \cup Y \uparrow \\
(X \cap Y)_\downarrow &\subseteq X \downarrow \cap Y \downarrow \\
(X \cap Y)_\uparrow &\subseteq X \uparrow \cap Y \uparrow
\end{align*}
\]

classic_bottom_classic: LEMMA FORALL (X: set[I]):

\[
\text{classic(bottom(classic(X)))) = classic(X)
\]

bottom_subset: LEMMA FORALL (X, Y: set[I], i: I):

\[
\text{subset?(X,Y) IMPLIES subset?(bottom(X),bottom(Y))}
\]

top_subset: LEMMA FORALL (X, Y: set[I], i: I):

\[
\text{subset?(X,Y) IMPLIES subset?(top(X),top(Y))}
\]

union_bottom: LEMMA FORALL (X,Y: set[I]):

\[
\text{bottom(union(X,Y)) = union(bottom(X),bottom(Y))}
\]

union_top: LEMMA FORALL (X,Y: set[I]):

\[
\text{top(union(X,Y)) = union(top(X),top(Y))}
\]

intersection_bottom: LEMMA FORALL (X,Y: set[I]):

\[
\text{subset?(bottom(intersection(X,Y)), intersection(bottom(X),bottom(Y)))}
\]

intersection_top: LEMMA FORALL (X,Y: set[I]):

\[
\text{subset?(top(intersection(X,Y)), intersection(top(X),top(Y)))}
\]
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)
```
Proposition 5 If $S$ is a total-closed set of interpretations, then $(\overline{S})_c \subseteq (\overline{S}_c)$. 

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

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 $\alpha$ with the grammar:

$$\alpha ::= \bot \mid p \mid \alpha_1 \land \alpha_2 \mid \alpha_1 \lor \alpha_2 \mid \alpha_1 \to \alpha_2$$

where $\alpha_1$ and $\alpha_2$ are formulae in their turn and $p \in \Sigma$ is any atom. We denote by $\neg \alpha \defeq \alpha \to \bot$ and $\top \defeq \neg \bot$. By $\mathcal{L}_\Sigma$ we denote the language of all well-formed formulae for signature $\Sigma$ or just $\mathcal{L}$ when the signature is clear from the context.

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

```pvs
Sig_form[T: TYPE\*:] : DATATYPE
BEGIN
  bot: bot?
END
```
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:

\[
\begin{align*}
    v(\bot) & \overset{\text{def}}{=} 0 \\
    v(\alpha \rightarrow \beta) & \overset{\text{def}}{=} \begin{cases} 
        2 & \text{if } v(\alpha) \leq v(\beta) \\
        v(\beta) & \text{otherwise}
    \end{cases} \\
    v(\alpha \land \beta) & \overset{\text{def}}{=} \min(v(\alpha), v(\beta)) \\
    v(\alpha \lor \beta) & \overset{\text{def}}{=} \max(v(\alpha), v(\beta))
\end{align*}
\]

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

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 \( v \in I \) and any \( \alpha, \beta \in L \),

- \( v(\alpha \land \beta) = 2 \iff v(\alpha) = 2 \) and \( v(\beta) = 2 \)
- \( v(\alpha \lor \beta) = 2 \iff v(\alpha) = 2 \) or \( v(\beta) = 2 \)

**caractv_form_and2**: Lemma \( \forall (t1, t2: \text{Sig_form}, i: I): v_form(op_and(t1, t2), i) = 2 \iff v_form(t1, i) = 2 \) AND \( v_form(t2, i) = 2 \)

**caractv_form_or2**: Lemma \( \forall (t1, t2: \text{Sig_form}, i: I): v_form(t1, i) = 2 \) AND \( v_form(t2, i) = 2 \)
Proposition 8 For any \( v \in \mathcal{I} \) and any \( \alpha, \beta \in \mathcal{L} \),

- \( v(\alpha \rightarrow \beta) = 2 \iff v(\alpha) = 0 \ or \ v(\beta) = 2 \ or \ v(\alpha) = 1 = v(\beta) \)

- \( v(\alpha \rightarrow \beta) = 0 \iff v(\alpha) \neq 0 \ and \ v(\beta) = 0 \)

- \( v(\alpha \rightarrow \beta) = 1 \iff v(\alpha) = 2 \ and \ v(\beta) = 1 \)

caractv_form_imp2: Lemma FORALL (t1,t2: Sig_form,i: I):
\[
v_{\text{form}}(\text{op\_imp}(t1,t2),i) = 2 \iff v_{\text{form}}(t1,i)=0 \ OR \ v_{\text{form}}(t2,i)=2 \ OR \ (v_{\text{form}}(t1,i)=1 \ AND \ v_{\text{form}}(t2,i)=1)
\]
caractv_form_imp0: Lemma FORALL (t1,t2: Sig_form,i: I):
\[
v_{\text{form}}(\text{op\_imp}(t1,t2),i) = 0 \iff \neg(v_{\text{form}}(t1,i)=0) \ AND \ v_{\text{form}}(t2,i)=0
\]
caractv_form_imp1: Lemma FORALL (t1,t2: Sig_form,i: I):
\[
v_{\text{form}}(\text{op\_imp}(t1,t2),i) = 1 \iff v_{\text{form}}(t1,i)=2 \ AND \ v_{\text{form}}(t2,i)=1
\]

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 \( \alpha \in \mathcal{L} \),

- \( v(\alpha) = 0 \iff v_{\text{f}}(\alpha) = 0 \)

- \( v(\alpha) = 2 \iff v_{\text{f}}(\alpha) = 2 \)

- \( v(\alpha) \geq 1 \iff v_{\text{f}}(\alpha) = 2 \)

caract_val_zero: Lemma FORALL (t: Sig_form,i: I):
\[
v_{\text{form}}(t,i)=0 \iff v_{\text{form}}(t,\text{RT}(i))=0
\]
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 $\alpha$, its denotation $\llbracket \alpha \rrbracket$ is a set of interpretations representing all the models of $\alpha$. In our case, we recursively define $\llbracket \alpha \rrbracket$ as follows:

- $\llbracket \bot \rrbracket = \emptyset$
- $\llbracket p \rrbracket = \{v \in I : v(p) = 2\}$
- $\llbracket \alpha \rightarrow \beta \rrbracket = (\llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket) \cap (\llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket)_{c \downarrow}$
- $\llbracket \alpha \land \beta \rrbracket = \llbracket \alpha \rrbracket \cap \llbracket \beta \rrbracket$
- $\llbracket \alpha \lor \beta \rrbracket = \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket$

This recursive definition is captured in the PVS code below:

```pvs
denotation(s: Sig_form): RECURSIVE set[I] =
  CASES s OF
    bot: emptyset,
    atom(t): \{i: I \mid 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
```
An important result is that the denotational of a formula $\alpha$ is, indeed, an equivalent reformulation of the set $G_3$ models of $\alpha$. This is stated by the theorem below.

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

$$v(\alpha) = 2 \text{ in } G_3 \text{ iff } v \in \llbracket \alpha \rrbracket$$

whose enunciate in PVS is as follows:

```pvs
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 $\alpha \in L$, $\llbracket \alpha \rrbracket$ is total-closed.

```pvs
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 $v \in I$ be a partial interpretation and $\alpha \in L$ a formula. Then:

$$v(\alpha) \neq 0 \text{ in } G_3 \text{ iff } v_t \in \llbracket \alpha \rrbracket$$

```pvs
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 $\alpha$ has denotation $I$, it means that $\alpha$ is a tautology.
Theorem 13  For any pair of formulae $\alpha, \beta$: $\llbracket \alpha \rrbracket \subseteq \llbracket \beta \rrbracket$ iff $\llbracket \alpha \to \beta \rrbracket = \mathcal{I}$.

Moreover, $\llbracket \alpha \rrbracket = \llbracket \beta \rrbracket$ iff $\llbracket \alpha \leftrightarrow \beta \rrbracket = \mathcal{I}$.

denotation_inclusion: LEMMA FORALL $(t_1,t_2: \text{Sig_form})$:
\[
\text{subset?}(\text{denotation}(t_1),\text{denotation}(t_2)) \iff \text{(FORALL } i: I\text{): member } (i,\text{denotation(op_imp}(t_1,t_2)))
\]
denotation_equal: LEMMA FORALL $(t_1,t_2: \text{Sig_form})$:
\[
\text{denotation}(t_1) = \text{denotation}(t_2) \iff \text{(FORALL } i: I\text{): member } (i,\text{denotation(op_and}(op_imp(t_1,t_2),op_imp(t_2,t_1))))
\]

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. $\llbracket \alpha \rrbracket \subseteq \llbracket \beta \rrbracket$ implies $\llbracket \gamma \to \alpha \rrbracket \subseteq \llbracket \gamma \to \beta \rrbracket$

2. $\llbracket \alpha \rrbracket \subseteq \llbracket \beta \rrbracket$ implies $\llbracket \beta \to \gamma \rrbracket \subseteq \llbracket \alpha \to \gamma \rrbracket$

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

mon_imp_left: LEMMA FORALL $(t_1,t_2,s: \text{Sig_form},i: I)$:
\[
\text{subset?}(\text{denotation}(t_1),\text{denotation}(t_2)) \implies \\
\text{subset?}(\text{denotation}(\text{op_imp}(s,t_1)),\text{denotation}(\text{op_imp}(s,t_2)))
\]

mon_imp_right: LEMMA FORALL $(t_1,t_2,s: \text{Sig_form},i: I)$:
\[
\text{subset?}(\text{denotation}(t_1),\text{denotation}(t_2)) \implies \\
\text{subset?}(\text{denotation}(\text{op_imp}(t_2,s)),\text{denotation}(\text{op_imp}(t_1,s)))
\]

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:
\[
\llbracket \alpha \to \beta \rrbracket = \llbracket \alpha \rrbracket_c \downarrow \cup (\llbracket \alpha \rrbracket \cap \llbracket \beta \rrbracket_c \downarrow) \cup \llbracket \beta \rrbracket
\]
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 $\alpha, \beta \in L$, it follows that:

1. $\llbracket \beta \rrbracket \subseteq \llbracket \alpha \rightarrow \beta \rrbracket$
2. $\llbracket \alpha \rightarrow \beta \rrbracket \subseteq \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket$
3. $\llbracket \alpha \rightarrow \beta \rrbracket \cap \llbracket \alpha \rrbracket \subseteq \llbracket \beta \rrbracket$

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 $\Sigma$, the system $L_{\Sigma}\{\bot, \land, \rightarrow\}$ is complete because given any pair of formulas $\alpha, \beta$ for $\Sigma$, it holds that:

$$\llbracket \alpha \lor \beta \rrbracket = \llbracket (\alpha \rightarrow \beta) \rightarrow \beta \rrbracket \cap \llbracket (\beta \rightarrow \alpha) \rightarrow \alpha \rrbracket.$$
Finally, we include a characterisation of models for negation:

**Proposition 18** For any formula $\alpha$:

1. $\denot{\neg \alpha} = \denot{\alpha}^c$

2. For any partial interpretation $v$, $v \in \denot{\neg \alpha}$ iff $v_t \in \denot{\alpha}$

**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 prone 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

https://github.com/nasa/pvslib/tree/master/ASP.


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:

\[
    v(\alpha) = 2 \text{ in } G_3 \text{ iff } v \in \llbracket \alpha \rrbracket
\]

\[
\text{denot_charac: LEMMA FORALL (s: Sig_form, i: I):} \\
    v_{\text{form}}(s, i) = 2 \text{ IFF member}(i, \text{denotation}(s))
\]

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_form` (formulas). The first subgoal corresponds to the $\bot$ constant:

\[
\text{Rule? (induct "s")}
\]

Inducting on $s$ on formula 1, this yields 5 subgoals:

\[
\text{denot_charac.1 :}
\]

\[
|-------
\]

\[
\{1\} \text{ FORALL (i: I): } v_{\text{form}}(\text{bot}, i) = 2 \text{ IFF member}(i, \text{denotation}(\text{bot}))
\]

To prove this subgoal, we apply skolemization, expand the definition of $v_{\text{form}}$ and we finally simplify. Then, the system proceeds to show the second subgoal (atoms).

\[
\text{This completes the proof of denot_charac.1.}
\]

\[
\text{denot_charac.2 :}
\]

\[
|-------
\]
The proof is analogous to the previous case. For the subgoal corresponding to the \( \land \) operator, the system provides the induction hypotheses:

\[
\text{denot\_charac.3 :}
\]

\[
|-------
\]

\[
\{1\} \quad \text{FORALL (op\_and1\_var: Sig\_form[T]), op\_and2\_var: Sig\_form[T]):}
\]

\[
((\text{FORALL (i: I):})
\]

\[
v\_form(op\_and1\_var, i) = 2 \text{ IFF}
\]

\[
\text{member}(i, \text{denotation}(op\_and1\_var)))
\]

\[
\text{AND}
\]

\[
(\text{FORALL (i: I):})
\]

\[
v\_form(op\_and2\_var, i) = 2 \text{ IFF}
\]

\[
\text{member}(i, \text{denotation}(op\_and2\_var)))
\]

\[
\text{IMPLIES}
\]

\[
(\text{FORALL (i: I):})
\]

\[
v\_form(op\_and(op\_and1\_var, op\_and2\_var), i) = 2 \text{ IFF}
\]

\[
\text{member}(i, \text{denotation}(op\_and(op\_and1\_var, op\_and2\_var)))
\]

After applying skolemization and instantiation on interpretation \( i \), we obtain the following, where \(-1\) and \(-2\) correspond to the induction hypothesis:

\[
\text{denot\_charac.3 :}
\]

\[
\{-1\} \quad \text{v\_form}(op\_and1\_var, i) = 2 \text{ IFF member}(i, \text{denotation}(op\_and1\_var))
\]

\[
\{-2\} \quad \text{v\_form}(op\_and2\_var, i) = 2 \text{ IFF member}(i, \text{denotation}(op\_and2\_var))
\]
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:

Once some propositional simplifications are applied, we expand the definitions member and denotation and we apply Lemma caractv_form_and2 (Proposition 7) to obtain:
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 \(\lor\) 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)))))
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 \( v(\alpha \rightarrow \beta) = 2 \implies v \in [\alpha \rightarrow \beta] \):

denot_charac.5.1 :

[-1] FORALL (i: I):
  v_form(op_imp1_var, i) = 2 IFF member(i, denotation(op_imp1_var))
We expand the definition of denotation for the conditional, $\llbracket \alpha \rightarrow \beta \rrbracket = \left( \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket \right) \cap \left( \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket \right)_c \downarrow$, and the set intersection:

denot_charac.5.1 :

[-1] \( v_{\text{form}}(\text{op_imp}(\text{op_imp1_var}, \text{op_imp2_var}), i) = 2 \)

[-2] \( \text{FORALL (i: I):} \)

\( v_{\text{form}}(\text{op_imp1_var}, i) = 2 \) IFF \( \text{denotation(\text{op_imp1_var})(i)} \)

[-3] \( \text{FORALL (i: I):} \)

\( v_{\text{form}}(\text{op_imp2_var}, i) = 2 \) IFF \( \text{denotation(\text{op_imp2_var})(i)} \)

|-------

\{1\} \( \text{member(i, union(comp(\text{denotation(\text{op_imp1_var})}), \text{denotation(\text{op_imp2_var}))})} \)

\text{AND}

\( \text{member(i, bottom(extend[I[T], IC[T], bool, FALSE]}

\left( \text{classic(union(comp(\text{denotation(\text{op_imp1_var})}), \text{denotation(\text{op_imp2_var}))))} \text{))} \right) \downarrow \)

With the command \text{split}, subgoal 5.1, \( v \in \left( \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket \right) \cap \left( \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket \right)_c \downarrow \), is unfolded into 5.1.1, \( v \in \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket \), and 5.1.2., \( v \in \left( \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket \right)_c \downarrow \). We begin proving membership in \( \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket \).

Rule? \text{(split)}

\text{Splitting conjunctions,}

this yields 2 subgoals:

denot_charac.5.1.1 :
\([-1]\) \(v_{\text{form}}(\text{op}\_\text{imp}(\text{op}\_\text{imp1}\_\text{var}, \text{op}\_\text{imp2}\_\text{var}), i) = 2\)
\([-2]\) \(\text{FORALL}(i: I):\)
\(\quad v_{\text{form}}(\text{op}\_\text{imp1}\_\text{var}, i) = 2 \iff \text{denotation}(\text{op}\_\text{imp1}\_\text{var})(i)\)
\([-3]\) \(\text{FORALL}(i: I):\)
\(\quad v_{\text{form}}(\text{op}\_\text{imp2}\_\text{var}, i) = 2 \iff \text{denotation}(\text{op}\_\text{imp2}\_\text{var})(i)\)

|-------
{1} member(i, union(comp(denotation(\text{op}\_\text{imp1}\_\text{var})), denotation(\text{op}\_\text{imp2}\_\text{var})))

Since this set is a union, we expand the definition of union and apply lemma \text{caractv}_{\text{form}}\_\text{imp2} (Proposition 8) instantiated on \text{op}\_\text{imp1}\_\text{var} and \text{op}\_\text{imp2}\_\text{var}, simplifying afterwards:

\[
\text{denot}\_\text{charac.5.1.1} :\]
\[
\{1\} \quad v_{\text{form}}(\text{op}\_\text{imp}(\text{op}\_\text{imp1}\_\text{var}, \text{op}\_\text{imp2}\_\text{var}), i) = 2 \iff \begin{array}{l}
\quad v_{\text{form}}(\text{op}\_\text{imp1}\_\text{var}, i) = 0 \ \text{OR} \\
\quad v_{\text{form}}(\text{op}\_\text{imp2}\_\text{var}, i) = 2 \ \text{OR} \\
\quad (v_{\text{form}}(\text{op}\_\text{imp1}\_\text{var}, i) = 1 \ \text{AND} \ v_{\text{form}}(\text{op}\_\text{imp2}\_\text{var}, i) = 1) \\
\end{array}
\]
\([-2]\) \(v_{\text{form}}(\text{op}\_\text{imp}(\text{op}\_\text{imp1}\_\text{var}, \text{op}\_\text{imp2}\_\text{var}), i) = 2\)
\([-3]\) \(\text{FORALL}(i: I):\)
\(\quad v_{\text{form}}(\text{op}\_\text{imp1}\_\text{var}, i) = 2 \iff \text{denotation}(\text{op}\_\text{imp1}\_\text{var})(i)\)
\([-4]\) \(\text{FORALL}(i: I):\)
\(\quad v_{\text{form}}(\text{op}\_\text{imp2}\_\text{var}, i) = 2 \iff \text{denotation}(\text{op}\_\text{imp2}\_\text{var})(i)\)

|-------
{1} member(i, comp(denotation(\text{op}\_\text{imp1}\_\text{var})))
{2} member(i, denotation(\text{op}\_\text{imp2}\_\text{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 \(v \in \llbracket \alpha \rrbracket\) or \(v \in \llbracket \beta \rrbracket\). After simplifying, we get:

\[
\text{denot}\_\text{charac.5.1.1.1} :\]

28
We prove that, in the first and third cases, \( v \) belongs to \( \llbracket \alpha \rrbracket \), whereas in the second case we show that it belongs to \( \llbracket \beta \rrbracket \). In its turn, the proof for 5.1.2, \( v \in (\llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket)_c \downarrow \), 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 \texttt{bottom} and \texttt{classic}. The rest of the proof, subgoal 5.2, \( v \in \llbracket \alpha \to \beta \rrbracket \Rightarrow v(\alpha \to \beta) = 2 \), 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].