Proceedings of the 15th International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2015 3–7July, 2015.

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

Felicidad Aguado[1], **Pablo Ascariz**[2], **Pedro Cabalar**[1], **Gilberto Pérez**[1] **and Concepción Vidal**[1]

[1] *Department of Computer Science, University of Corunna, Spain*

emails: `aguado@udc.es`, , `cabalar@udc.es`, `gperez@udc.es`, `concepcion.vidalm@udc.es`

Abstract

In this paper we provide an application of automated theorem proving for formal verification of Answer Set Programming (ASP). In particular, we have implemented the basic definitions of the ASP denotational semantics in the language of the PVS theorem prover, using afterwards some libraries and features of this prover to obtain certified proofs in an automated way.

1 Introduction

Answer Set Programming (ASP)[2] 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 [3] 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) [5] (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 [1]. 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 formalizable in terms of standard set theory and partial order relations, opening the possibility of 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 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. We have additionally included an example of proof session in an Appendix.

2 Brief overview of PVS

PVS is 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 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,

ISBN: 978-84-617-2230-3

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 use the alternative characterisation of HT in terms of Gödel's three-valued logic G_3 [4]. 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 $\mathcal{I}_c \subseteq \mathcal{I}$. The PVS encoding of these 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
```

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 `RT(i)`.

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

We define an ordering among partial interpretations 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 is actually equivalent to `R_ord` defined 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:

$$\begin{array}{ll} \forall v \in \mathcal{I} \;\; v \leq v_t & \forall v, u \in \mathcal{I} \;\; \text{if } v \leq u \text{ then } v_t = u_t \\ \forall v \in \mathcal{I}_c \;\; v_t = v & \forall v, u \in \mathcal{I} \;\; \text{if } v \leq u \text{ and } u \in \mathcal{I}_c \text{ then } v_t = u \end{array}$$

```
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
```

Given a set of interpretations $S \subseteq \mathcal{I}$ we will define some abbreviations:

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

To avoid too many parentheses, we will assume that $\downarrow$, $\uparrow$ and subindex c have more priority than standard set operations $\cup$, $\cap$ and $\setminus$.

ISBN: 978-84-617-2230-3

```
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 $X, Y \subseteq \mathcal{I}$,*

$$\begin{array}{l} (X_c\downarrow)_c = X_c \\ \textit{If } X \subseteq Y \textit{ then } X\downarrow \subseteq Y\downarrow \\ \textit{If } X \subseteq Y \textit{ then } X\uparrow \subseteq Y\uparrow \end{array} \qquad \begin{array}{rcl} (X\cup Y)\downarrow & = & X\downarrow \cup Y\downarrow \\ (X\cup Y)\uparrow & = & X\uparrow \cup Y\uparrow \\ (X\cap Y)\uparrow & \subseteq & X\uparrow \cap Y\uparrow \\ (X\cap Y)\downarrow & \subseteq & X\downarrow \cap Y\downarrow \end{array}$$

```
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))
```

ISBN: 978-84-617-2230-3

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 *For any total-closed set of interpretations S, it holds that $(\overline{S})_c\downarrow \subseteq \overline{(S_c\downarrow)}$*

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

Corollary 6 *For any total-closed set of interpretations S, it holds that $(\overline{S})_c\downarrow \subseteq \overline{S}$*

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

4 Valuation of formulas

A *formula* α is defined by the grammar:

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

ISBN: 978-84-617-2230-3

where α_1 and α_2 are formulae in their turn and $p \in \Sigma$ is any atom. We denote by $\neg\alpha \overset{\text{def}}{=} \alpha \to \bot$ and $\top \overset{\text{def}}{=} \neg\bot$. 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 syntactic encoding of expressions is then as follows using 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 powerfull 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 is initial Algebra defined by the constructors.

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:

$$v(\bot) \overset{\text{def}}{=} 0 \qquad v(\alpha \wedge \beta) \overset{\text{def}}{=} \min(v(\alpha), v(\beta))$$

$$v(\alpha \to \beta) \overset{\text{def}}{=} \begin{cases} 2 & \text{if } v(\alpha) \leq v(\beta) \\ v(\beta) & \text{otherwise} \end{cases} \qquad v(\alpha \vee \beta) \overset{\text{def}}{=} \max(v(\alpha), v(\beta))$$

```
I_form: TYPE = [Sig_form -> S3]
IC_form: TYPE = [Sig_form -> S2]
I_IC_form: JUDGEMENT IC_form SUBTYPE_OF I_form

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 <<
```

ISBN: 978-84-617-2230-3

In PVS, all functions must be total. The `MEASURE` part provides the information to the typechecker and prover to ensure the termination of recursion.

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 $\alpha, \beta \in \mathcal{L}$,*

- $v(\alpha \rightarrow \beta) = 2 \Longleftrightarrow v(\alpha) = 0$ *or* $v(\beta) = 2$ *or* $v(\alpha) = 1 = v(\beta)$
- $v(\alpha \rightarrow \beta) = 0 \Longleftrightarrow v(\alpha) \neq 0$ *and* $v(\beta) = 0$
- $v(\alpha \rightarrow \beta) = 1 \Longleftrightarrow v(\alpha) = 2$ *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
```

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.

5 Denotational Semantics

We define now the *denotation* of a formula α, written $[\![\, \alpha \,]\!]$, recursively as follows

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

```
denotation(s: Sig_form): RECURSIVE set[I] =
    CASES s OF
      atom(t): {i: I | i(t) = 2},
      bot: emptyset,
      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 <<
```

The next result has also been proved by structural induction.

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))
```

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

ISBN: 978-84-617-2230-3

```
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))
```

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

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

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

```
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)))
```

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

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

ISBN: 978-84-617-2230-3

```
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))
```

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))
```

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)))
```

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-generated proofs for several fundamental properties of ASP, constituting the first case of automated formal verification for this paradigm.

Acknowledgements

This research was partially supported by Spanish MEC project TIN2013-42149-P.

References

[1] Felicidad Aguado, Pedro Cabalar, David Pearce, Gilberto Pérez, and Concepción Vidal. A denotational semantics for equilibrium logic, 2015. Unpublished draft available at: `http://www.dc.fi.udc.es/~cabalar/denotational.pdf`.

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

[3] 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.

[4] Kurt Gödel. Zum intuitionistischen aussagenkalkül. *Anzeiger der Akademie der Wissenschaften Wien, mathematisch, naturwissenschaftliche Klasse*, 69:65–66, 1932.

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

[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.

Appendix. An example of proof session

The following example is a PVS proof session of the *caractv_form_or*2

- This is what we should prove:

```
caractv_form_or2 :

  |-------
{1}   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
```

- With skolemization we eliminate the for all quantifier.

```
 Rule? (skeep)
Skolemizing and keeping names of the universal formula in (+ -),
this simplifies to:
caractv_form_or2 :

  |-------
{1}   v_form(op_or(t1, t2), i) = 2 IFF v_form(t1, i) = 2 OR v_form(t2, i) = 2
```

```
Rule? (split)
Splitting conjunctions,
this yields  2 subgoals:
caractv_form_or2.1 :

  |-------
{1}   v_form(op_or(t1, t2), i) = 2 IMPLIES v_form(t1, i) = 2 OR v_form(t2, i) = 2

Rule? (flatten)
Applying disjunctive simplification to flatten sequent,
this simplifies to:
caractv_form_or2.1 :

{-1}  v_form(op_or(t1, t2), i) = 2
  |-------
{1}   v_form(t1, i) = 2
{2}   v_form(t2, i) = 2

Rule? (expand "v_form" -1)
Expanding the definition of v_form,
this simplifies to:
caractv_form_or2.1 :

{-1}  max(v_form(t1, i), v_form(t2, i)) = 2
  |-------
[1]   v_form(t1, i) = 2
[2]   v_form(t2, i) = 2
```

ISBN: 978-84-617-2230-3

```
Rule? (lemma "max_ge")
Applying max_ge
this simplifies to:
caractv_form_or2.1 :

{-1}  FORALL (a, b, c: real): max(a, b) >= c IFF (a >= c OR b >= c)
[-2]  max(v_form(t1, i), v_form(t2, i)) = 2
  |-------
[1]   v_form(t1, i) = 2
[2]   v_form(t2, i) = 2

Rule? (inst -1 "v_form(t1, i)" "v_form(t2, i)" "2")
Instantiating the top quantifier in -1 with the terms:
 v_form(t1, i), v_form(t2, i), 2,
this simplifies to:
caractv_form_or2.1 :

{-1}  max(v_form(t1, i), v_form(t2, i)) >= 2 IFF (v_form(t1, i) >= 2 OR v_form(t2, i) >= 2)
[-2]  max(v_form(t1, i), v_form(t2, i)) = 2
  |-------
[1]   v_form(t1, i) = 2
[2]   v_form(t2, i) = 2

Rule? (grind)
max rewrites max(v_form(t1, i), v_form(t2, i))
  to IF v_form(t1, i) < v_form(t2, i) THEN v_form(t2, i) ELSE v_form(t1, i) ENDIF
max rewrites max(v_form(t1, i), v_form(t2, i))
  to IF v_form(t1, i) < v_form(t2, i) THEN v_form(t2, i) ELSE v_form(t1, i) ENDIF
max rewrites max(v_form(t1, i), v_form(t2, i))
  to IF v_form(t1, i) < v_form(t2, i) THEN v_form(t2, i) ELSE v_form(t1, i) ENDIF
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of caractv_form_or2.1.

caractv_form_or2.2 :

  |-------
{1}   v_form(t1, i) = 2 OR v_form(t2, i) = 2 IMPLIES v_form(op_or(t1, t2), i) = 2

Rule? (flatten)
Applying disjunctive simplification to flatten sequent,
this simplifies to:
caractv_form_or2.2 :

{-1}  v_form(t1, i) = 2 OR v_form(t2, i) = 2
  |-------
{1}   v_form(op_or(t1, t2), i) = 2

Rule? (split)
```

```
Splitting conjunctions,
this yields  2 subgoals:
caractv_form_or2.2.1 :

{-1}  v_form(t1, i) = 2
  |-------
[1]   v_form(op_or(t1, t2), i) = 2

Rule? (expand "v_form" 1)
Expanding the definition of v_form,
this simplifies to:
caractv_form_or2.2.1 :

[-1]  v_form(t1, i) = 2
  |-------
{1}   max(v_form(t1, i), v_form(t2, i)) = 2



Rule? (typepred "v_form(t2, i)")
Adding type constraints for  v_form(t2, i),
this simplifies to:
caractv_form_or2.2.1 :

{-1}  s3?(v_form(t2, i))
[-2]  v_form(t1, i) = 2
  |-------
[1]   max(v_form(t1, i), v_form(t2, i)) = 2

Rule? (expand "s3?")
Expanding the definition of s3?,
this simplifies to:
caractv_form_or2.2.1 :

{-1}  v_form(t2, i) <= 2
[-2]  v_form(t1, i) = 2
  |-------
[1]   max(v_form(t1, i), v_form(t2, i)) = 2

Rule? (grind)
max rewrites max(v_form(t1, i), v_form(t2, i))
  to v_form(t1, i)
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of caractv_form_or2.2.1.

caractv_form_or2.2.2 :

{-1}  v_form(t2, i) = 2
```

```
  |-------
[1]   v_form(op_or(t1, t2), i) = 2

Rule? (expand "v_form" 1)
Expanding the definition of v_form,
this simplifies to:
caractv_form_or2.2.2 :

[-1]  v_form(t2, i) = 2
  |-------
{1}   max(v_form(t1, i), v_form(t2, i)) = 2

Rule? (typepred "v_form(t1, i)")
Adding type constraints for  v_form(t1, i),
this simplifies to:
caractv_form_or2.2.2 :

{-1}  s3?(v_form(t1, i))
[-2]  v_form(t2, i) = 2
  |-------
[1]   max(v_form(t1, i), v_form(t2, i)) = 2

Rule? (grind)
s3? rewrites s3?(v_form(t1, i))
  to v_form(t1, i) <= 2
max rewrites max(v_form(t1, i), v_form(t2, i))
  to IF v_form(t1, i) < v_form(t2, i) THEN v_form(t2, i) ELSE v_form(t1, i) ENDIF
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of caractv_form_or2.2.2.


This completes the proof of caractv_form_or2.2.

Q.E.D.
```

ISBN: 978-84-617-2230-3