A Compositional Proof Method of Partial Correctness for Normal Logic Programs

Gérard Ferrand and Arnaud Lallouet
Université d’Orléans LIFO — BP 6759 — F-45067 Orleans Cedex 2.
Tel: (33) 38.41.70.10, Fax: (33) 38.41.71.37
e-mail: {Gerard.Ferrand | Arnaud.Lallouet}@lifo.univ-orleans.fr

Abstract
This paper presents a new proof method of partial correctness for logic programs with negation based on a proof modularity. We prove in a compositional way that Fitting’s or the well-founded semantics of the program is included in a specification. We give conditions for an abstract semantics to be compositional and we base our proof method on this property. We present also conservative but compositional extensions of Fitting’s and of the well-founded semantics. As an illustration, an application is made to the module system of the Gödel language. Moreover, our method is suitable for incremental validation since it does not require all parts of the program to be implemented.

Keywords: Logic Programming — Negation — Compositional semantics — Validation — Partial correctness — Modules — Well-founded semantics — Fitting’s Semantics.

1 Introduction

Having validated software is one important issue of software engineering. But as soon as program size increases, the problem becomes more crucial. Usually, large programs are cut into modules or libraries, and software components are designed to be re-used. Our aim is to be able to validate a software component in such a way that we could re-use both component and knowledge of its validation, whatever the proof method used.

In this paper, we present a compositional proof method of partial correctness for normal logic programs with import called here units. This method is based on an extension of a proof method given by [6] for definite programs and [8] for normal logic programs. We choose to take up the theoretical framework of these works, in which a program is identified to the set of its ground instances and where the semantics, as well as the specification is denoted by a set of literals. The property of partial correctness of a program P wrt a specification S consists in proving the inclusion Semantics \( P \subseteq S \). The use of sets of literals allows us to remain in a purely set-theoretical framework. However, in practice, sets of literals are likely to be represented
by logical assertions and sets of ground rules by programs.

A proof method of partial correctness has been given in [8] for the well-

founded semantics, and indirectly in [7] for Fitting's semantics. The dual

approach—completeness—, has been studied by Mal'fion [18] in a framework

which unifies the well-founded, Fitting and Kunen's semantics.

But these results do only concern what is called "programming in the

small" and for larger programs, these non-compositional methods become

quickly intractable since they consider the whole program at the same time.

Our approach allows to cut large programs into pieces (and this is already
done if the program is made out of modules) and validate each piece sepa-

rately.

Our work gives answers to the following problems. What sense can we

give to the notion of correctness for a unit? How can we re-use the knowledge

of units validation in order to get a whole program correct? Moreover, it

answers a question raised in the conclusion of [3]:

"Compositionality is not addressed by the existing semantics of

normal programs. It is not clear how to determine the models of

a normal program from the models of its components."

To set up our compositional proof method, we only need the semantics
given to a unit to be compositional and the existence of a proof method for

a single unit. If ⊻ is a composition operator for programs, a compositional

semantics wrt ⊻ is such that the semantics of \( P_1 \bowtie P_2 \) can be deduced

from the semantics of \( P_1 \) and \( P_2 \). In our framework, we only focus on the

union operator, because our purpose is only to validate a program piece by

piece. To have this, the semantics needs to depend on a set of literals as

input. Then proving the partial correctness consists in proving the inclu-
sion Semantics\( P(S) \subseteq S' \): the output of the unit is proved correct (\( \subseteq S' \))

assuming the correctness of the input (the specification \( S \)).

We emphasize on the fact that our proof method only needs the compo-

sitionality of the semantics. This allows to use any proof method to validate

a unit. Then we present two compositional semantics. The first one is an

extension of the well-founded semantics, while the second extends Fitting's

semantics. We prove that they fulfill the requirements of a compositional

semantics. Moreover, we give proof methods for individual units. For the

well-founded semantics, we state the compositionality only if the system

of units is hierarchical, that is to say that there is no circular dependency

between the units.

The programming language Gödel [14] is an ideal framework to apply

our theoretical methods since it includes a module system that is moreover

hierarchical. In this context, we define a suitable notion of specification and

we build modular proofs for Gödel programs.

This paper is organized as follows: first, we recall some elements of

validation of logic programs and we set up our new proof method for an

abstract semantics \( SEM \). Then we introduce two compositional semantics
for normal programs. At last, the application to the language Gödel is detailed and the method is illustrated by an example. The results in this paper are greatly simplified for definite programs. The reader can find the missing proofs in [9].

2 General considerations about validation of logic programs

Validation of programs has been studied by various authors. Our work takes place in the continuity of the framework considered by [8], [18] and partially by [6] but is extended in order to take compositionality into account. We briefly recall these results to set the vocabulary and the notations.

We are only interested here in the validation of the declarative meaning of the program, excluding operational properties. Here, validation consists in comparing the actual declarative semantics of a program to a specification. In this framework, we consider that both the semantics and the specification are sets of ground literals. A program is identified to the ground instances of its clauses, thus leading to an infinite set of rules. Basically, a program is partially correct if its semantics is included in the specification and complete if the converse holds.

A proof method is a method that allows to prove such an inclusion. Proof methods are different depending on the considered semantics, the way the specification is expressed and the property that is to be proved.

We call proof methods described in [8] inductive proof methods. Another proof method has been given in the case of definite programs in [6] based on the notion of annotation. The knowledge that a unit is proved correct by such a method can be re-used in our compositional proof method.

2.1 Definite programs

The case of definite programs has been studied in [6]. For a definite program P, we consider the semantics given by lfp T_P, the least fixed point of the usual immediate consequence operator T_P. A specification S is any set of atoms. We say that:

- P is partially correct wrt S iff lfp T_P ⊆ S.
- P is complete wrt C iff C ⊆ lfp T_P.

If we replace lfp by gfp (greatest fixed point), we get another notion:

- P is sufficient wrt C iff C ⊆ gfp T_P. Sufficiency has a close relationship with debugging [7].

In this paper, we will call these properties atomic properties since they only deal with sets of atoms.
2.2 Normal programs

In this paper, we use the following notations.

- $\mathcal{HB}$ denotes the Herbrand base.
- For $a \in \mathcal{HB}$, $-a = -a$, $\neg a = a$ and $|a| = |\neg a| = a$.
- If $A \subseteq \mathcal{HB}$ is a set of atoms, $\overline{A} = \mathcal{HB} - A$ denotes the complement of $A$ in $\mathcal{HB}$ and $\neg A = \{-a | a \in A\}$.
- For $I \subseteq \mathcal{HB} \cup \neg \mathcal{HB}$, $I^+ = I \cap \mathcal{HB}$, $I^- = \{a \in \mathcal{HB} | \neg a \in I\}$ and $|I| = \{|l| | l \in I\}$.
- A normal program $P$ is a set of clauses $h \leftarrow B$ where $h \in \mathcal{HB}$ and $B \subseteq \mathcal{HB} \cup \neg \mathcal{HB}$.
- For $I \subseteq \mathcal{HB} \cup \neg \mathcal{HB}$, $\overline{I}$ is the conjugate of $I$ and is defined by $\overline{I} = I^- \cup \neg I^+ = \{l \in \mathcal{HB} \cup \neg \mathcal{HB} | \neg l \notin I\}$.
- For $I \subseteq \mathcal{HB} \cup \neg \mathcal{HB}$ and $J \subseteq \mathcal{HB}$, $I|_J = \{l \in I | |l| \in J\} = (J \cap I^+) \cup \neg(J \cap I^-)$. We say that $I|_J$ is the restriction of $I$ to $J$.
- Here $T_P$ is the mapping $T_P : 2^{\mathcal{HB} \cup \neg \mathcal{HB}} \rightarrow 2^{\mathcal{HB}}$ defined by $T_P(I) = \{h \in \mathcal{HB} | \exists B, h \leftarrow B \in P$ and $B \subseteq I\}$.
- For each $T$, we define the operator $T_P : 2^{\mathcal{HB}} \rightarrow 2^{\mathcal{HB}}$ by $T_P(J) = T(J \cup \neg J')$ for $J, J' \subseteq \mathcal{HB}$.

For normal logic programs, we need a semantics included in $\mathcal{HB} \cup \neg \mathcal{HB}$. Two main semantics are considered here: Fitting’s semantics [10] and the well-founded semantics [13]. We mostly use an alternate definition of the well-founded semantics [1] [22] which is also used in [8].

The well-founded semantics $WF(P)$ is defined as the $\lf_p$ of the operator $\Psi : 2^{\mathcal{HB} \cup \neg \mathcal{HB}} \rightarrow 2^{\mathcal{HB} \cup \neg \mathcal{HB}}$ defined by $\Psi_P(I) = \lf_p T_P(I) \cup \neg \lf_p T_P(I^-)$.

Fitting’s semantics $Fit(P)$ is defined as the $\lf_p$ of the operator $\Phi : 2^{\mathcal{HB} \cup \neg \mathcal{HB}} \rightarrow 2^{\mathcal{HB} \cup \neg \mathcal{HB}}$ defined by $\Phi_P(I) = T_P(I) \cup \neg T_P(I)$.

Its $\lf_p$ is also the $\lf_p$ of another operator [9]:

$\Psi' : 2^{\mathcal{HB} \cup \neg \mathcal{HB}} \rightarrow 2^{\mathcal{HB} \cup \neg \mathcal{HB}}$ defined by $\Psi'_P(I) = \lf_p T_P(I) \cup \neg \lf_p T_P(I^-)$.

This alternate definition is very close to the one of the well-founded semantics: only $\lf_p$ has been replaced by $\gamma f_p$ in the negative part. Validation results for these two semantics are recalled hereafter.

- **Partial correctness.** If we call $SEM_P \subseteq \mathcal{HB} \cup \neg \mathcal{HB}$ the program’s semantics ($SEM_P = Fit(P)$ or $SEM_P = WF(P)$), partial correctness wrt the set of literals $S$ means the inclusion $SEM_P \subseteq S$. 
- For the well-founded semantics, a correct and complete proof method is given in [8] and consists, like for definite programs, in finding an intermediate specification $S'$ such that $WF(P) \subseteq S' \subseteq S$. A sufficient condition is that $\Psi_P(S') \subseteq S'$, thus leading to the following checkings:

1. $WF^+(P) \subseteq S'^+$, i.e. $\lfp T_P S' \subseteq S'^+$. A sufficient condition to have this is to have $T_{P,S'}(S'^+) \subseteq S'^+$ i.e $\forall h \leftarrow B, B \subseteq S' \implies h \in S'^+$. This is called bottom-up closure in [8].

2. $WF^-(P) \subseteq S'^-$, or $S'^- \subseteq \lfp T_P S'$. A condition to have this is to have a well-founded ordering $\prec$ upon $S'^-$ such that $\forall h \in S'^-, \exists B, h \leftarrow B \in P, B \subseteq \bar{S}$ and $\forall b \in B^+, b \prec h$. This is symmetrically called top-down closure in [8]. The condition $b \prec h$ is called decreasing criterion.

- For Fitting’s semantics, [7] presents the equivalent of a proof method from the debugging point of view. Basically it is the same as for the well-founded semantics except that it does not need the decreasing criterion.

- **Completeness.** Completeness has been studied in [18] and is not in the scope of this paper.

Normal logic programs contain definite programs as a particular case, so it seems natural that proof methods for definite programs are just a particular case of the others. In fact, partial correctness for the well-founded semantics leads to atomic partial correctness and atomic completeness wrt respectively $S'^+$ and $S'^-$. This is due to the fact that, for definite programs, $\Psi_P(I)$ is a constant function, so $\lfp \Psi_P = \lfp T_P \cup - \lfp T_P$. For Fitting’s semantics defined by $\Phi'$, the same kind of remark leads to prove atomic partial correctness and sufficiency.

3 Compositional validation

Our aim is to find some modular proof methods: we want to consider a program as the union of several components and we want the correctness of the program to come only from the correctness of each of its components. This is what we call compositional validation, whatever the nature of each individual proof for a component.

This supposes a suitable notion of correctness for a single component, and also suitable notions of semantics and specification for a single component. It follows that the semantics of the program should be a function of the semantics of its components. By the way, a component could also be seen as the union of smaller components and conversely, the program itself should be the particular case of a component for a trivial decomposition.
These motivations lead us to consider a compositional semantics. A lot of work has been done to give logic programming such a semantics, especially as a foundation for module systems [16] [21] [11] [19] [12] [17] [4] [2] [5].

Our choice for the semantics of a component is imposed by the purpose of giving it proof methods which generalize the aforementioned ones for programs. The chosen semantics are precisely defined in section 4. When we only consider definite programs, they look like [21] or [11] but they just consider the union operator for it is the only one that matters for our compositional proof methods. [4] defines a semantics in terms of admissible model and [2] in terms of clauses, both for definite programs.

Programs with negation are considered by [17] but the purpose is to build a transformation system for deductive databases modules and this approach uses a condition of stratification inside the modules.

### 3.1 Unit and system of units

In order to prepare a program for composition with others, we need to take into account the fact that some information can be computed outside the program. This leads to the notion of unit: a program where predicates occurring in the body of clauses may be defined outside.

**Definition 3.1 (Unit)** A unit $u$ is a pair $(H, P)$ where $H \subseteq H_B$, $P$ is a logic program and for each clause $h \leftarrow B \in P$, $h \in H$.

The intended meaning is that in a unit, $H$ is seen as a local Herbrand base, since everything produced by the unit is in $H$. In a module framework, we could say that the language exported by the module is in $H$ while on the other side all external definitions ("import") are in $H$. This definition is close to the one of open programs [4]. A unit which imports nothing is identified to a program.

Units are made to combine with others. The following definitions concern systems of units.

**Definition 3.2 (System of units)** A system of units is a set $\mathcal{U}$ where every $i \in \mathcal{U}$ is a unit $(H_i, P_i)$ and $\forall \forall i, j \in \mathcal{U}, i \neq j \implies H_i \cap H_j = \emptyset$.

**Definition 3.3 (Sum of a system)** We call the unit $u = (H, P)$ with $H = \bigcup_{i \in \mathcal{U}} H_i$ and $P = \bigcup_{i \in \mathcal{U}} P_i$ the sum unit of a system $\mathcal{U}$.

A system of units is hierarchical if there is no positive circular dependency between units:

**Definition 3.4 (Hierarchical system of units)** For $i, j \in \mathcal{U}$, we say that $i \triangleright j$ iff there is a clause $h \leftarrow B \in P_i$ such that $B^+ \cap H_j \neq \emptyset$.

A system of units is hierarchical if the relation $\triangleright$ is well-founded.
The need of an import makes us consider that the genuine semantics of a unit \( u = (H, P) \) is given by a function \( S \mapstoSEM_{u}(S) \), for \( S \subset \overline{H} \cup \neg \overline{H} \). This semantics has to be a conservative extension of the one for programs on which it is based \((SEM_{P})\), thus leading to the following requirements.

- \( SEM_{u}(S) \subset H \cup \neg H \).
- if \( H = H \land B \), then the unit \( u \) can be identified with the program \( P \) for the only \( S \) that should be considered is \( S = \emptyset \) and then \( SEM_{u}(S) = SEM_{P} \).

In order to take into account an imported set of literals, the definition of partial correctness has to be extended: a unit is proved correct assuming the correctness of the input. Hence for units, a specification is made out of two parts: the input part and the output part.

**Definition 3.5 (Specification)** A specification for a unit \( u \) is a pair \( (S, S') \) where \( S \subset \overline{H} \cup \neg \overline{H} \) is the input part and \( S' \subset H \cup \neg H \) the output part.

**Definition 3.6 (Partial correctness for a unit)** We say that a unit \( u \) is partially correct wrt \((S, S')\), or in short \( u \) is \( PC \) wrt \((S, S')\) if \( SEM_{u}(S) \subset S' \).

### 3.2 A compositional proof method

To be compositional in our sense, a semantics must fulfill the following requirements in addition to the previous ones.

**Definition 3.7 (Compositional semantics)** Let \( \mathcal{U} \) be a system of units and \( u \) its sum. \( SEM \) is compositional if:

- \( S \mapsto SEM_{u}(S) \) is monotonic.
- Let \( \mu : 2^{\overline{H} \cup \neg H} \rightarrow 2^{\overline{H} \cup \neg H} \) be the monotonic operator defined by \( \mu(I) = \bigcup_{i \in \mathcal{U}} SEM_{i}(S \cup I|_{\overline{H}_{i}}) \). Then \( \mu = SEM_{u}(S) \).

The operator \( \mu \) can be viewed as the immediate consequence operator, not for the clauses of the units but for the whole units themselves. To be compositional, it is needed that the semantics obtained by this way coincides with the semantics of the union of the system of units’ clauses.

In the next section, we present some semantics suitable for compositionality, but for the exposition of the proof method, we assume that \( SEM \) is compositional. This abstract notion of compositionality is sufficient to state a new proof method to prove the partial correctness of a sum unit: the proof method by decomposition.

**Theorem 3.8 (Proof method by decomposition)** \( u \) is \( PC \) wrt \((S, S')\) if for each unit \( i \in \mathcal{U} \), \( i \) is \( PC \) wrt \((S \cup S'|_{\overline{H}_{i}}, S'|_{H_{i}})\).
Proof. $u$ is P.C. wrt $(S, S')$ means that $SEM_u(S) \subseteq S'$. But since $SEM$ is compositional, it is to say that $\forall_{(S, S') \subseteq S'}$. A sufficient condition to have this is $\mu(S') \subseteq S'$, i.e $\forall v \in U, SEM_i(S \cup S'|_{T_i}) \subseteq S'$. But $SEM_i() \subseteq H_i \cup -H_i$, so it is equivalent to $SEM_i(S \cup S'|_{T_i}) \subseteq S'|_{H_i}$.

Remark 3.9 This proof method is complete in a theoretical sense since $u$ is P.C. wrt $(S, S')$ iff there exists $S''$ such that $S'' \subseteq S'$ and $\forall v \in U$, $SEM_i(S \cup S''|_{T_i}) \subseteq S''|_{H_i}$. Such an $S''$ always exists since one can take $SEM_0(S)$. Actually, we have $\forall_{\mu \subseteq S'} \exists S'' \subseteq S'$ such that $\mu(S'') \subseteq S''$.

As a matter-of-fact, a suitable compositional semantics must have its proof method for a single unit. But even if not, our method is still applicable, because it considers only the compositional aspect. We only need to know that the unit has been proved, whatever the method used to get this result.

In this exposition, the conjugate specification plays a crucial role, leading to an elegant, powerful and nevertheless practical framework. Let us define this notion.

Definition 3.10 (Conjugate specification) The conjugate specification $\sim (S, S')$ of $(S, S')$ is a specification $(C, C')$ such that $C = \overline{S'}|_{H}$ and $C' = \overline{S'}|_{H}$.

Intuitively, if $S$ represents "strong" properties, i.e atoms which are certainly known to be true or false, $C$ represents the "weak" associated properties, i.e atoms which are not false or not true. Notice that the conjugate is an involutive mapping, since $\sim (S, S') = (S, S')$.

Moreover, this duality is echoed when proving properties of programs. This is the basis for the practical implementation of our method. A specification $(S, S')$ is made out of four sets of atoms $(S^+, S^-, S'^+, S'^-)$. Let's call $(C, C')$ the conjugate of $(S, S')$. Instead of giving the specification $(S, S')$ whose negative part may be tricky to explicit, the user gives four other sets of atoms: $S^+, C^+, S'^+, C'^+$ (see theorem 4.5 and example 4.9). The negative parts are the following: $S^- = \overline{H} - C^+$ and $S'^- = H - C'^+$. By considering the specification $C = \overline{S'}$, [8] contained the seeds of this idea but restricted to sets of atoms.

Since we know that the proof for an individual unit is made easier with the conjugate specification, we want to keep this advantage for our new proof method. The following lemma shows that the restrictions we made on sets in the proof method by decomposition are conservative wrt the conjugate specification.

Lemma 3.11 Let $(C, C') \sim (S, S')$. For every $i \in U$, let $S_i = S \cup S'|_{T_i}$, $S'_i = S'|_{H_i}$, and $(C_i, C'_i) \sim (S_i, S'_i)$. Then $C_i = \overline{C}|_{T_i}$ and $C'_i = C'|_{H_i}$.

A proof can be found in [9].
4 Normal logic programs

In this section are presented compositional extensions of the well-founded and Fitting’s semantics and their associated proof methods. To handle imported sets of literals, the usual mapping $T_P$ for normal programs is extended as follows.

Definition 4.1 (Extended $T_P$ operator) For a unit $u$ and an imported set of literals $S \subseteq \overline{H} \cup -\overline{H}$, we define $T^S_u(I) = T_P(S \cup I)$.

Note that $T^S_u(I) \subseteq H$.

4.1 Well-founded semantics

We propose the following extension of the well-founded semantics in order to handle an imported set of literals $S$: $SEM_u(S) = \text{lfp } \Psi^S_u$ where, for $I \subseteq H \cup \neg H$:

$$\Psi^S_u(I) = \text{lfp } T^S_{u,I_\neg} \cup \neg(H - \text{lfp } T^\neg_{u,H-I_\neg})$$

Theorem 4.2 The semantics $SEM_u : S \mapsto \text{lfp } \Psi^S_u$ is compositional if the unit system is hierarchical.

A detailed proof is given in [9]. Note that the assumption that the system of units is hierarchical is necessary, as shown by a counter-example presented in [9].

We also want a proof method to prove the partial correctness of a unit wrt a specification $(S, S')$ for this semantics. This means to prove the inclusion: $\text{lfp } \Psi^S_u \subseteq S'$. It is enough to have $\Psi^S_u(S') \subseteq S'$. There is a positive and a negative part that can be proved separately:

1. $\text{lfp } T^S_{u,S'} \subseteq S'^+$

2. $H - \text{lfp } T^\neg_{u,H-S'} \subseteq S'^-$

With the conjugate specification $\sim (S, S') = (C, C')$, one can see that the second inclusion is equivalent to:

2. $C'^{-} \subseteq \text{lfp } T^C_{u,C'}$

A practical interest of this method, as we said above, is that the negative part of the specification is expressed with the positive part of the conjugate specification $C^+$ and $C'^+$, which is much more natural than $S^-$ and $S'^-$. See the example 4.9 to appreciate this fact.

The following definitions are useful to state the conditions for these inclusions to hold:

Definition 4.3 (Bottom-up closure) We say that $u$ is bottom-up closed wrt $(Z, Z')$ if $\forall h \leftarrow B \in P, B \subseteq Z \cup Z' \Rightarrow h \in Z'^+$.
Definition 4.4 (Strong top-down closure) We say that $u$ is strongly top-down closed wrt $(Z, Z')$ if there exists a well-founded ordering $\prec$ upon $Z \cup Z'$ such that $\forall h \in Z^+, \exists B, h \leftarrow B \in P$ and $B \subseteq Z \cup Z'$ and $\forall b \in B \cap Z^+, b \prec h$.

Theorem 4.5 (Proof method of partial correctness wrt $S$ and $S'$ for the well-founded semantics) $u$ is P.C. wrt $(S, S')$ iff $\exists S''$ such that:

- $S'' \subseteq S'$.
- $u$ is bottom-up closed wrt $(S, S'')$.
- $u$ is strongly top-down closed wrt $\sim (S, S'')$.

Proof

$\Leftarrow$ A sufficient condition to have the first inclusion is $T_{u, S''}^S (S''^+) \subseteq S''^+$ which is obtained if $u$ is bottom-up closed wrt $(S, S'')$. A condition to have the second inclusion is to prove that $C''^+ \subseteq \text{gfp } T_{u, C''}^C$ which is obtained when $u$ is strongly top-down closed wrt $(C, C'')$.

$\Rightarrow$ Take $S'' = \text{gfp } \Phi_u^S$.

We can notice that the proof method is correct and complete because of the iff, but in practice, it may be difficult to get a suitable intermediate specification $S''$. The proof method by decomposition applies because the semantics is compositional. Then we get that the sum unit $u$ is P.C. wrt $(S, S')$ if for each unit $i \in U$, $i$ is bottom-up closed wrt $(S \cup S'[H_i], S'[H_i])$ and strongly top-down closed wrt $\sim (S \cup S'[H_i], S'[H_i])$. This emphasize the role of the conjugate specification, as we noticed above.

4.2 Fitting’s semantics

Fitting operator’s [10] definition is $\Phi_P (I) = T_P (I) \cup \overline{T_P (I)}$. For the purpose of a uniform characterization of Fitting’s and the well-founded semantics, we use here the other operator given in section 2.2. We propose the following extension to handle an imported set of literals.

$$\Phi_u^S (I) = \text{gfp } T_{u, I}^{S'} \cup \overline{(H \cdot \text{gfp } T_{u, I}^{S'})}$$

Theorem 4.6 The semantics $SEM_u : S \mapsto \text{gfp } \Phi_u^S$ is compositional.

A detailed proof can be found in [9].

Note that there is no need for the unit system to be hierarchical. As for the well-founded semantics, we want a proof method for a single unit, i.e. to prove that $u$ is P.C. wrt $(S, S')$. Due to the similarities between this operator and the one of the well-founded semantics, the properties to prove are similar too, except that there is now an inclusion in a $\text{gfp}$. This removes the need for a decreasing criterion in the top-down closure.
Definition 4.7 (Weak top-down closure) We say that \( u \) is weakly top-down closed wrt \((Z, Z')\) if \( \forall h \in Z^{+}, \exists B, h \leftarrow B \in P \) and \( B \subseteq Z \cup Z' \).

Theorem 4.8 (Proof method of partial correctness wrt \((S, S')\) for Fitting's semantics)
\( u \) is P.C. wrt \((S, S')\) iff \( \exists S'' \) such that:

- \( S'' \subseteq S' \).
- \( u \) is bottom-up closed wrt \((S, S'')\).
- \( u \) is weakly top-down closed wrt \((S, S'')\).

The proof method by decomposition is of course applicable since the semantics is compositional. Similarly to the well-founded semantics, it comes to check the bottom-up closure wrt \((S_i, S'_i)\) and the weak top-down closure wrt \((S_i, S'_i)\). \( S_i \) and \( S'_i \) are defined like in lemma 3.11.

4.3 Example

The example consists of three units computing inclusion of lists. We illustrate the adaptation power of the method by giving a program under implementation:

Example 4.9 (Inclusion of lists (taken from [8]))

Unit \( u_1 : P_1 = \{ \text{includ}(L_1, L_2) \leftarrow \text{not nclud}(L_1, L_2). \} \)
\( H_1 = \{ \text{includ}(L_1, L_2) \mid L_1 \text{ and } L_2 \text{ are ground terms} \}. \)

Unit \( u_2 : P_2 = \{ \text{nclud}(L_1, L_2) \leftarrow \text{elem}(X, L_1) \text{ and } \text{not elem}(X, L_2). \} \)
\( H_2 = \{ \text{nclud}(L_1, L_2) \mid L_1 \text{ and } L_2 \text{ are ground terms} \}. \)

Unit \( u_3 : P_3 \) is under implementation.
\( H_3 = \{ \text{elem}(X, L) \mid X \text{ and } L \text{ are ground terms} \}. \)

We are interested in the unit \( u \) sum of \( u_1 \) and \( u_2 \). Note that this unit still imports some literals. We denote by letters the following sets of atoms:

- \( X = \{ \text{includ}(l_1, l_2) \mid l_1 \text{ and } l_2 \text{ lists } \Rightarrow l_1 \subseteq l_2 \}. \)
- \( Y = \{ \text{nclud}(l_1, l_2) \mid l_1 \text{ and } l_2 \text{ lists } \Rightarrow l_1 \not\subseteq l_2 \}. \)
- \( Z = \{ \text{elem}(x, L) \mid \text{lists } \Rightarrow x \in L \}. \)
- \( X' = \{ \text{includ}(l_1, l_2) \mid l_1 \text{ and } l_2 \text{ lists and } l_1 \subseteq l_2 \}. \)
- \( Y' = \{ \text{nclud}(l_1, l_2) \mid l_1 \text{ and } l_2 \text{ lists and } l_1 \not\subseteq l_2 \}. \)
- \( Z' = \{ \text{elem}(x, L) \mid \text{lists and } x \in L \}. \)

As specifications, we take \( S^+ = Z \) and \( S'^+ = X \cup Y \). As stated in section 3.2, the specification \((S, S')\) is actually defined by \( S = S^+ \cup \neg(H - C^+) \) and \( S' = S^+ \cup \neg(H - C'^+) \) because the negative part of the specification is expressed with the conjugate one \((C, C')\). To have this, we take \( C^+ = Z' \) and \( C'^+ = X' \cup Y' \).

In order to apply the proof method by decomposition, we have to check that \( u_i \) is P.C. wrt \( S_i \) and \( S'_i \). It is sufficient to check that \( u_i \) is bottom-up closed wrt \( S_i \) and \( S'_i \) and \( u_i \) is (strongly of weakly) top-down closed wrt \( C_i \) and \( C'_i \). The individual specifications are:
\[ S_1^+ = Y \cup Z, \quad S_1^+ = X, \quad C_1^+ = Y' \cup Z', \quad C_1^+ = X'. \]
\[ S_2^+ = X \cup Z, \quad S_2^+ = Y, \quad C_2^+ = X' \cup Z', \quad C_2^+ = Y'. \]

Since the system of units is hierarchical, we can make proofs wrt the well-founded semantics as well as Fitting's. But since there is no recursive clause in the units \( u_1 \) and \( u_2 \), there is no need for the decreasing criterion and the proofs are the same in both cases. The individual proofs are obvious.

### 4.4 Definite programs

The particular case of definite programs is interesting because it greatly simplifies the method. Actually, since there is no negative literal in bodies of clauses, the operator becomes, for example for the well founded semantics, to:

\[ \Psi_u^S(I) = \text{ifp } T_u^S \cup (H - \text{ifp } T_u^{\overline{S}}) \]

One can see that the operator is constant, leading to prove atomic properties separately. For the well-founded semantics the properties are atomic partial correctness and atomic completeness: \( \text{ifp } T_u^S \subseteq S^+ \) and \( \overline{S} \subseteq \text{ifp } T_u^{\overline{S}} \). For Fitting's semantics, the properties are atomic partial correctness and atomic sufficiency.

### 5 Application to the Gödel module system

Above results are to be applied to module systems. Here we take the example of the programming language Gödel, described in [14]. A Gödel module is an entity syntactically defined by two parts: the export part and the local part. Either one of these parts may be empty. A module always has a name chosen from a set \( \mathcal{N} \) of module names. More details about the syntax can be found in [14]. Since our framework only concerns “pure” logic programs, we use here a subset of the real Gödel language.

Gödel has a syntax-flattening mechanism which ensures the uniqueness of each symbol definition and allows resolution of various name conflicts and over-loadings. A Gödel program is a set of modules such that every used symbol is defined in only one module. In order to apply our results, we will precis the definition of a Gödel module:

**Definition 5.1 (Gödel module)** A Gödel module is a 6-tuple composed of:

- a name \( m \in \mathcal{N} \).
- two disjoint sets of module names \( \text{imp}(m) \) and \( \text{imp}(m) \) corresponding to module names respectively imported by the export part and the local part of the module \( m \). We will denote \( \text{imp}(m) = \text{imp}(m) \cup \text{imp}(m) \).
• two disjoint sets of literals \( \text{EXP}(m) \) et \( \text{LOC}(m) \) corresponding respectively to the exported language and to the local language of the module.

• a set of clauses \( P_m \) defining the predicates of \( \text{EXP}(m) \) and \( \text{LOC}(m) \).

We note \( H_m = |\text{EXP}(m) \cup \text{LOC}(m)| \). By its syntax-flattening mechanism, Gödel ensures that \( m \neq m' \implies H_m \cap H_{m'} = \emptyset \). \( H_m \) and the set of clauses \( P_m \) are chosen to be the unit associated to the module \( m \). We define the relation \( \text{re-exports} \) upon \( \mathcal{N} \) as follows:

**Definition 5.2 (Relation re-exports)** We say that “\( m \) re-exports \( m' \)” and we note \( \text{reexp}(m, m') \) iff \( m' \in \text{impf}(m) \).

Let \( \text{reexp}^* \) be its reflexive and transitive closure. We define the following set \( \text{EXP}^*(m) = \bigcup \{ \text{EXP}(m') | \text{reexp}^*(m, m') \} \).

**Definition 5.3 (Relation uses)** We say that “\( m \) uses \( m'' \)” iff \( \exists m' \in \text{impf}(m), \text{reexp}^*(m', m'') \). We define then \( \pi(m) = \{ m'' | m \text{ uses } m'' \} \).

**Remark 5.4** In a Gödel program, the dependency relation “uses” between modules is well-founded [14], defining therefore a hierarchical system of units. Hence we can apply our results concerning both Fitting’s and the well-founded semantics.

By definition, the semantics of a Gödel program is the semantics of the sum of its units \( P = \bigcup_m P_m \). Let’s take a specification \( S \subseteq \bigcup_m H_m \) split in as many parts as there are modules as follows:

\[
S = \bigcup_m (S | H_m) = \bigcup_m (S \cdot \text{EXP}(m) \cup S \cdot \text{LOC}(m)) \text{ with }
\]

\[
S \cdot \text{EXP}(m) \subseteq \text{EXP}(m) \quad \text{and} \quad S \cdot \text{LOC}(m) \subseteq \text{LOC}(m)
\]

By this split, our method leads to the validation of every single unit \( m \) wrt \( S \cdot \text{IN}(m) \) and \( S \cdot \text{OUT}(m) \) with:

\[
S \cdot \text{IN}(m) = \bigcup_{m' \in \pi(m)} S \cdot \text{EXP}(m') \quad \text{and} \quad S \cdot \text{OUT}(m) = S \cdot \text{EXP}(m) \cup S \cdot \text{LOC}(m)
\]

All the above results apply.

6 Conclusion.

Let’s recall briefly the main results. We consider programs with import called units for which an abstract semantics is a function taking in input a set of imported literals and producing another set of literals. In our point of view, a suitable semantics must extend conservatively a semantics of classical programs. It must be compositional wrt union of units and have a proof
method for a single unit. For such a semantics, we give a proof method of partial correctness by decomposition for a unit $u$ sum of a system $U$ which comes to validate independently each unit wrt a local specification: $u$ is partially correct wrt $S$ and $S'$ if for each unit $i \in U$, $i$ is partially correct wrt $(S \cup S'[i])$ and $S'[i]$. The semantics $S \rightarrow \mu \Psi_u$ and $S \rightarrow \mu \Phi_u$ given in section 4 are proved to be suitable for our proof method.

Our first perspective is to continue the theoretical work by addressing the problem of completeness. The approach used in [18] seems fruitful because it gives conditions for an interpretation to be included in the well-founded or Fitting's semantics. Another perspective is an implementation. It should provide some facilities, i.e. a way to describe sets of atoms for specifications by assertions or other proof methods such as an extension of the proof method with annotations [6]. Application to the module part of Standard Prolog [15] is another interesting experiment. But, the module specification is too unstable now for us to give the details of the adaptation. Still further is the extension to other formalisms, such as Contextual Logic Programming [20]. But some composition operators such as the overriding-union operator seem to need an extension of our framework, since they are clearly non-monotonic.

References


