D. Turi

Extending S-interpretations to logic programs with negation

Computer Science/Department of Software Technology Report CS-R8948 November

1989

D. Turi

Extending S-interpretations to logic programs with negation

Computer Science/Department of Software Technology Report CS-R8948 November

Extending S-Interpretations to Logic Programs with Negation

Daniele Turi
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
and
Department of Computer Science, University of Pisa,
Corso Italia 40, 56100 Pisa, Italy

S-Interpretations together with the notion of s-truth are used in [FLMP] to define a least model semantics that resembles very much the least Herbrand model semantics and that provides a strongly complete declarative modelling for positive logic programming (ie Horn clause logic + SLD-resolution). Defining *cs-interpretations* as sets of *constrained atoms* we extend s-interpretations to general programs (ie logic programs with negation). Chan's constructive negation has inspired our definition and provides a procedural support for the declarative framework we set up. In this, to define the notion of cs-truth, we introduce *most general disunifiers* between conjunctions of atoms and cs-interpretations.

Stratified programs are general programs much related to positive ones. In [ABW] a fixpoint construction is introduced to produce a minimal Herbrand model which extends to stratified programs the least Herbrand model for positive programs. In [P1] an alternative characterization of such model is given (perfect Herbrand model). Applying these two constructs to cs-models we obtain the unique *perfect cs-model*.

We end the paper conjecturing a strong completeness for our cs-semantics wrt Przymusinski's interpreter for stratified programs and constructive negation (SLSC-resolution).

1980 Mathematics Subject Classification (1985): 68Q55, 68T15.
1986 CR Categories : F.4.1, F.3.2.

1. Background

1.1. S-Interpretations

The semantics based on s-interpretations (*s-semantics*) is a declarative semantics for *positive logic programming* (ie Horn clause logic + SLD-resolution). It has been introduced in FALASCHI, LEVI, MARTELLI AND PALAMIDESSI [FLMP] in order to provide a complete characterization for SLD-resolution computed answers, which are modelled only in part by all other declarative semantics. Take the *least Herbrand model semantics* and its completeness result, for instance. Only the existence of an SLD-answer-substitution is characterized:

$$M_P \models \exists Q \iff \exists \text{ SLD-answer-substitution for } Q \qquad (1)$$

(where M_P is the least Herbrand model of the program P and Q is query a containing only positive atoms).
When queries in universal form are taken into consideration the least Herbrand model is misleading. For instance, given the program $P = p(a) \leftarrow$, we have $M_P \models \forall x.p(x)$, thus completeness would imply that the identity be an SLD-anwer-substitution to the query, while the only answer to it is $\{x/a\}$. (This is also a *modularity* problem: when we add to P above a totally uncorrelated clause like $q(b) \leftarrow$ we have that $\forall x.p(x)$ is not entailed by the least Herbrand model anymore.)

The notion of *logical consequence of the program* provides a more accurate characterization:

$$P \models \forall Q\theta \iff \exists \text{ SLD-answer-substitution for } Q \text{ which is more general than } \theta. \qquad (2)$$

The problem here is that not only the computed answers are logical consequences of the program, but also all their possible instances. Thus, it would be more correct to take (2) as a soundness rather than as a completeness result. Moreover, we miss here the *constructivity* which the immediate consequence operator provides (with its least fixpoint) to the least Herbrand model semantics. Actually, as

Report CS-R8948
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

suggested in PRZYMUSINSKI [P], this last problem can be overcome by adding Clark's Equational Theory (CET) to the programs. CET axiomatizes the unification algorithm's terms handling, so that when added to a program P it imposes a structure on its interpretations which allows a generalization of the immediate consequence operator. There are more of these operators — one for each interpretation of CET — and every minimal model of P+CET is the least fixpoint of one of them ($T^J_P\uparrow\omega$, with J model of CET). In terms of a program's logical consequences, the collection $MIN(P)$ of minimal models of P+CET has the same representative power as the entire collection of models for P. That is, we can restate (2) as:

$$MIN(P) \models \forall Q\theta \;\Leftrightarrow\; \exists \text{ SLD-answer-substitution for } Q \text{ which is more general than } \theta. \qquad (3)$$

Notice that $MIN(P)$ is, in general, an infinite collection.

The *s-semantics* encompasses all previous semantics, combining:
- constructivity;
- single model;
- modularity;
- (strong) completeness.

This is obtained by abstracting the infinite collection $MIN(P)$ into a single structure — a 'supra-model' (*s-model*) for P. This s-model is a partially specified model, the presence of variables allowing different mappings. Here is an example of s-interpretation:

EXAMPLE 1.1.
The s-interpretation

$$I = \{\, p(f(x)) \,\}$$

stands for the whole family of interpretations

$$\{\, p(f(t)) \,\}_{t\in D}\ ,$$

where D is any given domain for I.
For $P = p(f(x)) \leftarrow$, I is a synthetic representation of $MIN(P)$. □

A major point for the s-semantics is that it is based on natural generalizations of notions from the Herbrand semantics. The *Herbrand s-universe* of a program P is simply the Herbrand universe of the language of P extended with infinite variables. Since in LP variables are always closed (by the implicit universal quantification of each clause), a distinction between them is necessary only inside atoms: $p(x,y)$ is equivalent to $p(x,z)$, but different from $p(x,x)$. Variables are thus grouped in equivalence classes and B^S_P, the *Herbrand s-base* of P, is built up from the Herbrand s-universe modulo such equivalence. S-Interpretations are then *subsets* (this is the original meaning of the prefix *s*, but we definitely prefer *supra*) of the Herbrand s-base.

All the properties of Herbrand interpretations still hold for the *s*- extension: the *s*-interpretations of a program (together with the usual set inclusion) form a complete lattice; every program has a least Herbrand *s*-model; such least *s*-model is both the intersection of all *s*-models and the least fixpoint of the immediate *s*-consequence operator.

The definition of s-model requires a reformulation of the notion of truth:

$$I \models_S A \qquad \textit{iff}\ \exists\, \alpha \in I \text{ such that } \alpha \text{ is more general than } A$$

$$I \models_S A_1,\ldots,A_n \qquad \textit{iff}\ \exists\, \alpha_1,\ldots,\alpha_n \in I \text{ s.t. } (\alpha_1,\ldots,\alpha_n) \text{ is more general than } (A_1,\ldots,A_n)$$

$$I \models_S A \leftarrow A_1,\ldots,A_n \quad \textit{iff}\ \forall\, \alpha_1,\ldots,\alpha_n \in I \quad \exists\, \theta = \text{mgu}((A_1,\ldots,A_n),\ (\alpha_1,\ldots,\alpha_n)) \Rightarrow A\theta \in I.$$

(By "$\in$" we intend here a membership which takes variables redenominations into account, ie such that $p(x,y) \in \{ p(x,z)\}$ and $p(x,y) \notin \{ p(x,x)\}$. For any two atoms — or vectors of atoms — α_1 and α_2 we say that α_1 is *more general* than α_2 when there exists a substitution θ s.t. $\alpha_1\theta = \alpha_2$.)

With this we do not introduce an alternative logic but we simply provide a synthetic way to construct entire families of models for P+CET. To prove this we have first to define the *immediate s-consequence operator* — the elementary unit of s-model construction:

$$
\begin{aligned}
\alpha \in T_P^S(I) \;\; \textit{iff} \;\; & \exists\, A \leftarrow A_1,\dots,A_n \text{ in } P \\
& \exists\, \alpha_1,\dots,\alpha_n \in I \quad \exists\, \theta \text{ such that} \\
& \quad \theta = \mathrm{mgu}((A_1,\dots,A_n),\ (\alpha_1,\dots,\alpha_n)) \\
& \quad \alpha = A\theta
\end{aligned}
$$

Every model in $MIN(P)$ can be obtained as a 'projection' of $T_P^S\uparrow\omega$, the least Herbrand s-model of P. By a 'projection' we intend here a *single* state application to an opportune redenomination of the s-model, equivalent to the original one (for the definition of *state* see APT [A]). Such redenomination, which we shall denote by "$\star$", is necessary in order to pass at once from a set like $\{ p(x) \}$ to $\{ p(a), p(b) \}$. (In this particular case $\{ p(x) \}^\star$ could be $\{ p(x), p(y) \}$ with projecting state ρ: $\rho(x) = a$ and $\rho(y) = b$.)

THEOREM 1.2. *Let P be a positive logic program.*

$$\forall\, J \text{ model of CET } \exists\, \rho \text{ over J such that } \quad T_P^J\uparrow\omega = (T_P^S\uparrow\omega)^\star \rho\,.$$

PROOF. See Theorem 3.4 in TURI [T]. □

But the most important result for the s-semantics is its *strong completeness theorem* (for the proof see [FLMP]):

$$M_P^S \models_S Q\theta \;\Leftrightarrow\; \theta \text{ is an SLD-answer-substitution for } Q \tag{4}$$

(where M_P^S denotes P's least s-model).

1.2. Logic Programs with Negation

To improve the expressive power of logic programming on finite domains and to make of logic programming an adequate formalism for knowledge representation a non-monotonic form of negation (*negation as failure*) has been introduced: negative literals are allowed to appear in queries and, recursively, in bodies of program clauses (subqueries). SLD-resolution extended with the negation as (finite) failure rule (*SLDNF-resolution*) provides the computational mechanism for such programs (*general* programs) and queries with negation. Unfortunately, recursion through negation, because of its non-monotonic nature, may bring about writing ill-founded programs for which no consistent interpretation can be found. As a consequence, a declarative modelling of the operational behaviour of logic programs with negation is never complete, (unless we adopt a three-valued logic as in KUNEN [K]). Moreover, a semantics like the least Herbrand model's cannot be given, for general programs do not have a least Herbrand model, but a collection of minimal Herbrand models whose intersection is not a model and whose union is inconsistent (in general).

When recursion through negation is forbidden the resulting programs are called *stratified*, since they can be regarded as structured in 'strata', each stratum being a subprogram whose predicates appearing in negative literals are defined in some strictly lower stratum. The first stratum is thus a positive program (eventually empty). The next strata can be regarded as positive programs which may use negative information concerning some previously defined programs.

Procedurally, we define *stratified logic programming* as non-recursively nested positive logic programming together with negation as failure. Queries and subqueries may contain negative literals, but SLD-resolution and negation as failure are never mixed, for strata cannot be ascended until the selected literal in the (sub-) query has been resolved. Przymusinski's SLS-resolution ([P]) is the interpreter which exploits this idea.

Also declaratively, stratified programs encompass positive ones. Although they have no least Herbrand model, there is a particular minimal Herbrand model (*perfect* Herbrand model) which corresponds to the intended meaning of the programs and which can be expressed via a fixpoint construction very much resembling that for positive programs (see Apt, Blair and Walker [ABW] and Przymusinski [P], or chapter 7 of Apt [A]). Calling M_P the perfect Herbrand model for a stratified program P, we can restate the completeness result (1) from the previous section as:

$$M_P \models \exists Q \;\Leftrightarrow\; \exists \text{ SLS-answer-substitution for } Q \qquad (1')$$

(where Q is a *non-floundered* query – see next chapter).

To complete the analogy with positive programs we may consider stratified programs extended with CET. Then, if we call $PERF(P)$ the collection of perfect models for P+CET, we have the following completeness result, which extends (3) from the previous section:

$$PERF(P) \models \forall Q\theta \;\Leftrightarrow\; \exists \text{ SLS-answer-substitution for } Q \text{ more general than } \theta, \qquad (3'a)$$

$$PERF(P) \models \neg\exists Q \;\Leftrightarrow\; \text{SLS-resolution fails on } Q. \qquad (3'b)$$

Also here Q is assumed to be non-floundered.

2. Purpose

The aim of this paper is to extend the s-semantics to stratified programs, so to have an equivalent for SLS-resolution of the completeness result (4). For such extension to be significant, not only the strong completeness but also the other properties of the s-semantics (constructivity, single model and modularity) should hold.

The major obstacle to our aim is the 'groundness' constraint which is imposed on negation as failure by classical interpreters: a negative literal in a query or subquery is selected only if it is ground. This is a procedural requirement which has no declarative modelling, so that classical completeness results like (1') and (3') are always stated in terms of non-floundered queries, that is queries whose negative literals are or become ground before selection. Since non-floundering is not decidable, it is a major problem for logic programming in general. But to us, the restriction to ground negation is a problem even independently of floundering: the s-semantics' power rests upon the use of non-ground atoms in the interpretations, so that there can be no sound extension for it when negative literals are to be grounded.

The solution to these problems is Chan's *constructive negation* ([C]), a sound way to implement a non-ground negation as failure. Here are some examples of constructive negation:

Example 2.1.

The program

$$p(a)\leftarrow,$$

under constructive negation, answers to the query

$$\leftarrow\neg\, p(x)$$

with

$$[x\neq a]. \quad \square$$

Example 2.2.

$p \leftarrow \neg q(x)$

$q(f(x)) \leftarrow q(x)$

$q(0) \leftarrow$

$\leftarrow p$

true. □

EXAMPLE 2.3.
Replacing the first clause in Example 2.2 with

$p(x) \leftarrow \neg q(x)$

we have

$[x \neq 0,\ x \neq f(0),\ x \neq f^2(0), \ldots]$. □

EXAMPLE 2.4.

$q(x,y,z) \leftarrow p(x,x), p(y,y), \neg\, p(x,y)$

$p(x,x) \leftarrow$

$\leftarrow q(x,y,z)$

$[x \neq y]$. □

We may sum up the main features of constructive negation as follows:
i) constructive negation subsumes (ground) negation as failure;
ii) it overcomes floundering;
iii) it is logically sound;
iv) it treats variables in negative literals in a 'default' manner —complementary to the treatment of variables in positive literals.

(The significance of the last property can be argued by those standard argumentations behind non-monotonic reasoning.)

Although it has been introduced to extend SLDNF-resolution, the constructive negation is a notion of wider application. For instance, in PRZYMUSINSKI [P2] it is used to extend SLS-resolution. And it is from such an interpreter (*SLSC-resolution*) that we expect completeness wrt to the semantics we propose.

3. FRAMEWORK

3.1. CS-Interpretations

On the following definition we base our extension: a *constrained atom* (*c-atom*) is an atom together with a conjunction of inequalities (*constraint*), which can eventually be empty. (For the definition of 'inequalities' see CHAN [C].)

EXAMPLE 3.1.
The following are c-atoms:

$p(x)[x \neq a,\ x \neq b]$,

$$p(f(x))[x \neq g(a)],$$
$$q(f(x))[\forall u.x \neq g(u)],$$
$$p(a),$$
$$q(x). \quad \square$$

Notice that we have not specified whether the constraint must be finite or not, but we believe this depends on the application one has in mind.

A *constrained s-interpretation (cs-interpretation)* is then a set of c-atoms. Clearly, cs-interpretations are those extensions of s-interpretations we are looking for. Actually, we should reduce them modulo the equivalence class on the variables (compare with what below Example 1.1), but the reader should not be deceived by this understatement. For instance, it should not come as a surprise when we treat

$$\{\, p(x)[x \neq a] \}$$

as equivalent to

$$\{\, p(x)[x \neq a],\ p(y)[\, y \neq a] \}.$$

3.2. Constrained Substitutions and Unifiers

We now need to define *c-substitutions* as well as unifiers between atoms (of a program) and c-atoms (of a cs-interpretation). We do not give a formal definition, but we show what *most general unifiers* have to look like in this context. Referring to CHAN [C] this should suffice to generalize.

EXAMPLE 3.2.

$$mgu(\, p(a,x),\ p(y,f(z))[\forall u.z \neq g(u)]\,) = \{y=a,\ x=f(z)\}[\forall u.z \neq g(u)]. \quad \square$$

EXAMPLE 3.3.

$$mgu(\, p(a,x),\ p(y,f(y))[\forall u.y \neq g(u)]\,) = \{y=a,\ x=f(a)\}. \quad \square$$

Notice that in Example 3.3 the constraint is dropped, since it is *valid* (under CET). Instead, the constraint in Example 3.2 is just *satisfiable.*

EXAMPLE 3.4.

There is no *mgu* of

$$(\, p(\, g(a),x),\ p(y,z)[\forall u.y \neq g(u)]\,)$$

because when we apply

$$mgu(\, p(\, g(a),x),\ p(y,z)\,) = \{\, y=g(a),\ x=z\}$$

to

$$[\forall u.y \neq g(u)]$$

we obtain

$$[\forall u.g(a) \neq g(u)]$$

which is clearly *unsatisfiable.* $\square$

To summarize, a most general unifier between an atom and a c-atom

- does not introduce new constraints
- can reduce or even delete a constraint if either a part or the whole constraint results valid under 'canonical' unification
- exists iff the canonical unifier exists and satisfies the constraint in the c-atom.

3.3. Immediate Consequence Operator for CS-Interpretations

To find a proper definition for the immediate consequence operator is the core of the problem to us: applying to it the same fixpoint construction as in Apt, Blair and Walker [ABW] we shall obtain the standard cs-model for stratified programs.

We put for a c-atom α, a cs-interpretation I and a general program P:

$$\begin{aligned}
\alpha \in T_P^{CS}(I) \;\; \textit{iff} \;\; & \exists\, A \leftarrow A_1,\ldots,A_n,\neg B_1,\ldots,\neg B_m \text{ in } P \\
& \exists\, \alpha_1,\ldots,\alpha_n \in I \;\; \exists\, \theta \text{ such that} \\
& \quad \theta = mgu((A_1,\ldots,A_n),(\alpha_1,\ldots,\alpha_n)) \\
& \quad \exists\, \eta \text{ such that} \\
& \quad\quad \eta = \boldsymbol{mgd}((B_1,\ldots,B_m)\theta,\; I) \\
& \quad\quad \alpha = A\theta\eta
\end{aligned}$$

By *mgd* we intend *most general disunifier* — a cardinal notion to which we shall devote the rest of this chapter.

3.4. Most General Disunifiers

In order to provide the reader with a good insight, we shall proceed with examples. In the sequel, $\boldsymbol{B}$ will stand for $(B_1,\ldots,B_m)\theta$ and $\boldsymbol{\beta}$ will stand for $(\beta_1,\ldots,\beta_m)$.

Roughly speaking, a disunification is a complemented unification. However, much care is needed for the following reasons:

1) In general, complementing an mgu gives rise to more than just one mgd:

Example 3.5.

$$mgu(\,p(x,y),\ \{\,p(a,b)\}\,) = \{x=a,\ y=b\}$$

$\Rightarrow$ two mgd's:

$$mgd_1 = [x \neq a]$$

$$mgd_2 = \{x=a\}[\,y \neq b] \quad \square$$

Notice that $[x \neq a]$ and $[y \neq b]$ overlap, thus an order is needed. In the sequel — again following Chan [C] — we shall always proceed as above, ie from left to right.

2) We first need to collect all possible unifications of $\boldsymbol{B}$ with c-atoms of I and then disunify *globally*:

Example 3.6.

$$mgd(\,p(x),\ \{\,p(a),p(b)\}\,) = [x \neq a,\ x \neq b]. \quad \square$$

For this reason in the definition of T_P^{CS} we put $mgd(\boldsymbol{B}, I)$.

3) It is possible to disunify only if there is not in I any vector of c-atoms $\boldsymbol{\beta}$ more general than $\boldsymbol{B}$:

   EXAMPLE 3.7.

   Neither $p(f(x))$ nor $p(x)$ disunify with $\{p(y)\}$. □

4) Finally, we also have to take care that a combination of unifiers does not produce an empty substitution when restricted to the variables of $\boldsymbol{B}$:

   EXAMPLE 3.8.

   $mgu(p(x), p(a)) = \{x = a\}$

   $mgu(p(x), p(x)[x \neq a]) = [x \neq a]$

   $\{x = a\} \cup [x \neq a] = \epsilon \Rightarrow p(x)$ does not disunify with $\{p(a), p(x)[x \neq a]\}$. □

In order to deal with all mgu's between $\boldsymbol{B}$ and $\boldsymbol{\beta}$'s in I, it is useful to define

$$mgu(\boldsymbol{B}, I) = \{\theta\colon \exists\ \boldsymbol{\beta} \subseteq I \text{ s.t. } \theta = (\ mgu(\boldsymbol{B}, \boldsymbol{\beta}) \mid \boldsymbol{B}\ )\}$$

(where "$|\boldsymbol{B}$" restricts the mgu's to the variables of $\boldsymbol{B}$ – the only variables we are interested in). If we put

$$I \not\vdash_{CS} \boldsymbol{B} \ \textit{ iff } \ \neg\exists\ \boldsymbol{\beta} \subseteq I \text{ s.t. } \boldsymbol{\beta} \text{ is more general than } \boldsymbol{B} \text{ and}$$
$$mgu(\boldsymbol{B}, I) \text{ does not } \textit{cover } \boldsymbol{B}$$

(where by *covering* we intend the property illustrated by Example 3.8, ie a combination of substitutions producing ϵ) we have then that

$$\exists\ mgd(\boldsymbol{B}, I) \Leftrightarrow I \not\vdash_{CS} \boldsymbol{B}.$$

After this, we can concentrate on the *form* mgd's must have. Two steps are enough:

i) When $mgu(\boldsymbol{B}, I)$ is a singleton, say $\{\theta\}$, with

   $$\theta = \{x_1 = r_1, \ldots, x_h = r_h\}[\forall(s_1 \neq t_1), \ldots, \forall(s_k \neq t_k)],$$

   we have then that the set of all mgd's between $\boldsymbol{B}$ and I is determined by the 'complement' of θ:

   $$\begin{aligned}\bar{\theta} = \{\ & [\forall(x_1 \neq r_1)],\\ & \{x_1 = r_1\}[\forall(x_2 \neq r_2)],\\ & \quad \ldots\\ & \{x_1 = k_1, \ldots, x_{h-1} = r_{h-1}\}[\forall(x_h \neq r_h)],\\ & \{x_1 = r_1, \ldots, x_h = r_h,\ s_1 = t_1\}\\ & \{x_1 = r_1, \ldots, x_h = r_h,\ s_2 = t_2\}[\forall(s_1 \neq t_1)],\\ & \quad \ldots\\ & \{x_1 = r_1, \ldots, x_h = r_h,\ s_k = t_k\}[\forall(s_1 \neq t_1), \ldots, \forall(s_{k-1} \neq t_{k-1})]\ \}.\end{aligned}$$

ii) When $mgu(\boldsymbol{B}, I)$ is a set S of many c-substitutions, the set of mgd's between $\boldsymbol{B}$ and I has to be obtained by first complementing all c-substitutions of S independently and then putting together the constraints of those who share the 'equality-part':

EXAMPLE 3.9.

$$I = \{\, p(f(u),a),\ p(f(u),b),\ p(u,v)[u\neq a],\ p(a,f(v))\}$$

$$mgu(p(x,y), I) = \{\ \underset{\theta_1}{\{x=f(u),\ y=a\}},\ \underset{\theta_2}{\{x=f(u),\ y=b\}},\ \underset{\theta_3}{[x\neq a]},\ \underset{\theta_4}{\{x=a,\ y=f(v)\}}\ \}$$

$$\overline{\theta_1} = \{\ [\forall u.x\neq f(u)],\ \{x=f(u)\}[\,y\neq a\,]\ \}$$

$$\overline{\theta_2} = \{\ [\forall u.x\neq f(u)],\ \{x=f(u)\}[\,y\neq b\,]\ \}$$

$$\overline{\theta_3} = \{\ \{x=a\}\ \}$$

$$\overline{\theta_4} = \{\ [x\neq a],\ \{x=a\}[\forall v.y\neq f(v)]\ \}$$

$\Rightarrow$ three mgd's:

$$\eta_1 = [\forall u.x\neq f(u),\ x\neq a\,]$$

$$\eta_2 = \{x=f(u)\}[\,y\neq a,\ y\neq b\,]$$

$$\eta_3 = \{x=a\}[\forall v.y\neq f(v)].\quad \square$$

In general, for S set of c-substitutions, denoting by σ, σ_1,... *non* constrained substitutions (ie sets of equalities) and by μ, μ_1,... constraints (ie sets of inequalities), we put:

$$\sigma\mu_1 \cdots \mu_l \in \overline{S}\ \ \textit{iff}\ \ \forall\mu\ [(\exists\theta\in S \text{ s.t. } \sigma\mu\in\overline{\theta}) \Leftrightarrow (\exists i\in[1,l] \text{ s.t. } \mu=\mu_i)].$$

Now, we can finally give the formal definition of mgd: for a c-substitution η we put

$$\eta = mgd(\boldsymbol{B}, I)\ \ \textit{iff}\ \ I \not\models_{CS} \boldsymbol{B}\ \text{ and }\ \eta\in\overline{mgu(\boldsymbol{B}, I)}.$$

4. THEOREMS

4.1. *CS-Models*

In the previous section we have defined when a vector (conjunction) of atoms is not 'cs-true'. Here we simply complement such definition in order to define when an atom or a conjuction of atoms is true in a cs-interpretation:

$I \models_{CS} A$ *iff* either $\exists\ \alpha\subseteq I$ such that α is more general than A
or $mgu(A,I)$ covers A

$I \models_{CS} A_1,...,A_n$ *iff* either $\exists\ \alpha_1,...,\alpha_n\in I$ s.t. $(\alpha_1,...,\alpha_n)$ is more general than $(A_1,...,A_n)$
or $mgu((A_1,...,A_n),I)$ covers $(A_1,...,A_n)$

It is easy now from the definition of immediate cs-consequence to extend to cs-interpretations the notion of true program clause:

$$I \models_{CS} A \leftarrow A_1,\ldots,A_n,\neg B_1,\ldots,\neg B_m \quad \textit{iff} \quad \forall\ \alpha_1,\ldots,\alpha_n \in I$$
$$\exists\ \theta = mgu((A_1,\ldots,A_n),(\alpha_1,\ldots,\alpha_n))$$
$$\exists\ \eta = mgd((B_1,\ldots,B_m)\theta,\ I)$$
$$\Rightarrow \alpha = A\theta\eta$$

4.2. Standard CS-Model for Stratified Programs

In this section we generalize to cs-interpretations the fixpoint construction which in APT, BLAIR AND WALKER [ABW] is used to produce the intended model of a stratified program. We shall borrow both structure and notation from Chapter 7 of APT [A] to which we refer the reader for the necessary background. In the rest of this chapter we consider a general program P stratified by

$$P = P_1 \dot{\cup} \cdots \dot{\cup} P_n.$$

What we need is to prove that, for each $i = 1,\ldots,n$, $T^{CS}_{P_i}$ is *finitary* and *growing* and that the sequence $T^{CS}_{P_1},\ldots,T^{CS}_{P_n}$ is *local* (compare with [A]). The reader should be familiar with the definition of *finitary operator* (see eg LLOYD [L]) and then it should be clear that also for cs-interpretations the immediate consequence operator is finitary. $T^{CS}_{P_i}$ is said *growing* when

$$\forall I,J,M \quad I \subseteq J \subseteq M \subseteq T^{CS}_{P_i}\Uparrow\omega(I) \Rightarrow T^{CS}_{P_i}(J) \subseteq T^{CS}_{P_i}(M).$$

The double arrow stands for *cumulative powers*, whose definition (again from APT [A]) for any operator T on a complete lattice is as follows:

$$T\Uparrow 0(I) = I$$
$$T\Uparrow(k+1)(I) = T(T\Uparrow k(I)) \cup T\Uparrow k(I)$$
$$T\Uparrow\omega(I) = \bigcup_{k<\omega} T\Uparrow k(I).$$

LEMMA 4.1. (Compare with Lemma 7.13 in APT [A].) *$T^{CS}_{P_i}$ considered as an operator on the complete lattice $\{I : I \subseteq B^{CS}_P\}$ is growing.*

PROOF. We will show that if $I \subseteq J \subseteq M \subseteq T^{CS}_{P_i}\Uparrow\omega(I)$ then $T^{CS}_{P_i}(J) \not\subseteq T^{CS}_{P_i}(M)$ leads to a contradiction.

For every α belonging to $T^{CS}_{P_i}(J)$ we have that

$$\exists\ A \leftarrow A_1,\ldots,A_n,\neg B_1,\ldots,\neg B_m \text{ in } P_i$$
$$\exists\ \alpha_1,\ldots,\alpha_n \in J \quad \exists\ \theta = mgu((A_1,\ldots,A_n),(\alpha_1,\ldots,\alpha_n))$$
$$\exists\ \eta = mgd((B_1,\ldots,B_m)\theta,\ J) \text{ such that } \alpha = A\theta\eta.$$

Since J $\subseteq$ M then $\alpha_1,\ldots,\alpha_n$ belong to M too, so that α does not belong to M only if

$$\eta \neq mgd((B_1,\ldots,B_m)\theta,\ M).$$

That is, only if either

$$M \models_{CS} (B_1,\ldots,B_m)\theta \tag{a}$$

or

$$\eta \notin \overline{mgu((B_1,\ldots,B_m)\theta,\ M)} \tag{b}$$

Stratification implies that predicates appearing in negative literals of P_i are defined in some stricly lower strata, thus they cannot appear in heads of clauses of P_i. Therefore, the set of atoms occurring in negative literals of P_i is the same in I and in $T^{CS}_{P_i}\Uparrow\omega(I)$, thus (by hypothesis) the same in J and in M. But for either (a) or (b) to hold there must be some atoms in M which are not in J and that have same predicate symbols as some of the B_j's, for $j=1,...,m$. Here is the contradiction and the end of the proof. □

The sequence $T^{CS}_{P_1},...,T^{CS}_{P_n}$ is said *local* when, for $i=1,...,n$,

$$\forall I,J \quad I \subseteq J \subseteq M_n \Rightarrow T^{CS}_{P_i}(J)=T^{CS}_{P_i}(J\cap M_i).$$

Here by M_i, for $i=1,...,n$, we intend the 'cs-equivalent' of the fixpoint construction for P's standard model. That is:

$$M_1=T^{CS}_{P_1}\Uparrow\omega(\varnothing)$$

$$M_2=T^{CS}_{P_2}\Uparrow\omega(M_1)$$

$$\cdots$$

$$M_n=T^{CS}_{P_n}\Uparrow\omega(M_{n-1}).$$

In the sequel we put:

$$M^{CS}_P=M_n.$$

We do not prove the following lemma, since it is the 'cs-equivalent' of Lemma 7.14 in Apt [A] and similar arguments as in the previous proof can be applied:

Lemma 4.2. *The sequence of operators $T^{CS}_{P_1},...,T^{CS}_{P_n}$ considered on the complete lattice $\{I:I\subseteq B^{CS}_P\}$ is local.* □

By Corollary 7.10 and Lemma 7.11 in Apt [A] it follows then that M^{CS}_P is a minimal fixpoint and prefixpoint of T^{CS}_P. Thus to prove that it is a minimal cs-model for P it suffices to prove that:

Lemma 4.3. *For any cs-interpretation I and any general program P*

$$I \models_{CS} P \quad \textit{iff} \quad T^{CS}_P(I)\subseteq I$$

Proof. We simply adapt the proof of Lemma 6.1 in Falaschi, Levi, Martelli and Palamidessi [FLMP] to cs-interpretations:

$$\begin{aligned}
I \models_{CS} P \quad \textit{iff} \quad & \forall\, A\leftarrow A_1,...,A_n,\neg B_1,...,\neg B_m \text{ in } P \;\; \forall\, \alpha_1,...,\alpha_n\in I \text{ s.t.}\\
& \quad \exists\, \theta=mgu((A_1,...,A_n),(\alpha_1,...,\alpha_n)) \text{ and } \exists\, \eta=mgd((B_1,...,B_m)\theta,\, I) \Rightarrow \alpha=A\theta\eta.\\
\textit{iff} \quad & \forall\alpha \text{ s.t. } \exists\, A\leftarrow A_1,...,A_n,\neg B_1,...,\neg B_m \text{ in } P\\
& \qquad \exists\, \alpha_1,...,\alpha_n\in I \quad \exists\, \theta=mgu((A_1,...,A_n),(\alpha_1,...,\alpha_n))\\
& \qquad \exists\, \eta=mgd((B_1,...,B_m)\theta,\, I) \text{ and } \alpha=A\theta\eta.\\
& \qquad \Rightarrow \alpha\in I\\
\textit{iff} \quad & T^{CS}_P(I)\subseteq I. \quad \square
\end{aligned}$$

By all this:

Theorem 4.4.

M_P^{CS} is a minimal cs-model for P. □

4.3. Perfect CS-Models

To further the similarity between M_P and M_P^{CS} we have to prove that M_P^{CS} is the unique perfect cs-model of P.

Perfect models are a class of minimal models for general programs which has been introduced in PRZYMUSINSKI [P1] to provide a characterization of M_P independent from the chosen stratification. Their definition is based on the following notion of preference ($\leqslant$) between interpretations:

$$I_1 \leqslant I_2 \text{ iff } \forall \alpha \in I_1 \setminus I_2 \Rightarrow \exists \beta \in I_2 \setminus I_1 \text{ s.t. } \alpha > \beta.$$

Here, $>$ stands for any well founded ordering for the atoms of the interpretations, which are assumed to be sets of atoms (but not necessarily ground). Atoms whose predicate are defined in a stratified program have a natural ordering: $\alpha > \beta$ when in all possible stratifications the predicate in α is defined in a strictly higher stratum than the predicate in β.

We say that a (cs-) model is *perfect* when there is no other (cs-) model preferable to it. To prove that M_P^{CS} is the unique perfect cs-model for P we shall again follow APT [A], giving the cs-equivalents of its Lemmata 7.18ii), 7.19 and 7.20.

LEMMA 4.5.

$$\forall I, J \subseteq B_P^{CS} \quad I \leqslant J,\ J \leqslant I \Rightarrow I = J.$$

PROOF. Same as for Lemma 7.18ii) in APT [A]. □

Let us now put

$$P_i = P_1 \cup \cdots \cup P_i$$

and, for every N model of P

$$N_i = N \cap B_{P_i}^{CS}.$$

Notice that $M_i = M_P^{CS} \cap B_{P_i}^{CS}$.

LEMMA 4.6. (Compare with Lemma 7.19 in APT [A].) *Let N be a cs-model for P. Then for all i* = 1,...,*n we have* $M_i \leqslant N_i$.

PROOF.

By induction on i.

$i=1$) Since P_1 is a positive program M_1 is its least Herbrand model.

$i \rightarrow i+1$) Here it is convenient to follow the construction of M_{i+1} and induce on it:

$$M_{i+1} = \bigcup_{k=0}^{\infty} T_{P_{i+1}}^{CS} \Uparrow k(M_i)$$

$k=0$)

$$\begin{aligned} \alpha \in M_i \setminus N_{i+1} &\Rightarrow \alpha \notin N_i \\ &\Rightarrow \exists \beta \in N_i \setminus M_i \text{ and } \alpha > \beta \\ &\Rightarrow \beta \notin M_{i-1} \ \textit{(by stratification)}. \end{aligned}$$

$k \to k+1$) Let us put $M = T^{CS}_{P_{i+1}} \Uparrow k(M_i)$.

$$\alpha \in T^{CS}_{P_{i+1}}(M) \setminus M \Rightarrow \exists\, A \leftarrow A_1,...,A_l, \neg B_1,...,\neg B_m \text{ in } P_{i+1}$$
$$\exists\, \alpha_1,...,\alpha_l \in M \quad \exists\, \theta = mgu((A_1,...,A_l),(\alpha_1,...,\alpha_l))$$
$$\exists\, \eta = mgd((B_1,...,B_m)\theta,\ M) \text{ such that } \alpha = A\theta\eta.$$

Since N_{i+1} is a model of P_{i+1} we have that if α does not belong to it then either

$$\exists j \in [1,l] \text{ s.t. } \alpha_j \notin N_{i+1} \qquad \text{(a)}$$

or

$$\exists j \in [1,m] \text{ s.t. } N_{i+1} \models_{CS} B_j\theta\eta \qquad \text{(b)}$$

In case (a) holds then there is a β in N_{i+1} and not in M_{i+1} such that $\alpha_j > \beta$. But then, by stratification, $\alpha > \beta$.

In case (b) holds then there must be a β in N_{i+1} which is more general than $B_j\theta\eta$, thus with the same predicate symbol. Clearly, β cannot belong to M_{i+1} and, again by stratification, $\alpha > \beta$. This concludes the proof. □

LEMMA 4.7.

$$\forall I,J \subseteq B^{CS}_P,\ \forall i = 1,...,n,\ \ I_i \leqslant J_i \Rightarrow I \leqslant J.$$

PROOF. Same as for Lemma 7.20 in APT [A]. □

We can then conclude:

THEOREM 4.8.

M^{CS}_P is the unique perfect cs-model for P.

PROOF. By Lemmata 4.5, 4.6 and 4.7. □

5. CONCLUSION

Defining cs-interpretations as sets of constrained atoms we have extended s-interpretations to general programs. Through the notion of most general disunifier an extension of the immediate consequence operator has also been given, so that a fixpoint construnction producing the unique perfect cs-model of stratified programs has been possible. This extends the results in APT, BLAIR AND WALKER [ABW] and in PRZYMUSINSKI [P1].

To complete our extension still two results are missing. One is that every perfect model for a stratified program P together with CET – ie every model in $PERF(P)$ – can be obtained as a projection of the unique perfect cs-model of P. This amounts to a simple generalization of Theorem 1.2 and we do not prove it here.

The other missing result is one of strong completeness of our cs-semantics. We expect it to hold wrt an interpreter like Przymusinski's SLSC-resolution (for which the '$PERF(P)$-semantics' is complete, but not strongly). It should sound as follows:

CONJECTURE 5.1.

$M^{CS}_P \models_{CS} Q\theta\eta \Leftrightarrow \theta\eta$ is an SLSC-answer-substitution for Q. □

ACKNOWLEDGEMENTS

This work was conducted during my stay at the *Centre for Mathematics and Computer Science* of Amsterdam, where, under the supervision of Professor Krzysztof R. Apt, I wrote my Master thesis, of which this paper is the last part. The same acknowledgements as in TURI [T] (the rest of the thesis) are due here. To those I wish to add just few words:

I have learned of the s-semantics from Professor Roberto Barbuti (University of Pisa) who also suggested me to extend it so to model stratified programs.

I would like to thank Dr Catuscia Palamidessi for her encouragement when I met the first obstacles and I was very close to give up the whole enterprise.

REFERENCES

[A] K.R. APT, *Introduction to Logic Programming,* Technical Report No. CS-R8826, Centre for Mathematics and Computer Science, Amsterdam, 1988, to appear as a chapter in Handbook of Theoretical Computer Science, J. van Leeuwen managing ed., North-Holland.

[ABW] K.R. APT, H.A. BLAIR and A. WALKER, *Towards a Theory of Declarative Knowledge,* in: Foundations of Deductive Databases and Logic Programming (Minker, J., ed.), Morgan Kaufmann, Los Altos, 1988, 89-148.

[C] D. CHAN, *Constructive Negation Based on the Completed Database,* in: Proc. 5th International Conference and Symposium on Logic Programming, (Kowalski, R.A. and K. Bowen, eds), The MIT Press, Cambridge, Mass., 1988, 111-125.

[FLMP] M. FALASCHI, G. LEVI, M. MARTELLI and C. PALAMIDESSI, *Declarative Modeling of the Operational Behaviour of Logic Languages,* Theoretical Computer Science, vol. 70, 1989.

[L] J.W. LLOYD, *Foundations of Logic Programming,* Second Edition, Springer Verlag, 1987.

[P] T. PRZYMUSINSKI, *On the Declarative and Procedural Semantics of Logic Programs,* Journal of Automated Reasoning, vol. 5, No. 2, 1989.

[P1] T. PRZYMUSINSKI, *On the Declarative Semantics of Deductive Databases and Logic Programs,* in: Foundation of Deductive Databases and Logic Programming, (Minker, J., ed.), Morgan Kaufmann, Los Altos, 1988, 193-216.

[P2] T. PRZYMUSINSKI, *On Constructive Negation in Logic Programming,* Draft paper.

[T] D. TURI, *Logic Programs with Negation: Classes, Models and Interpreters,* Technical Report No. CS-R8943, Centre for Mathematics and Computer Science, Amsterdam, 1989.