Set Based-Analysis of Logic Programs via Abstract Interpretation

Giorgio Levi Walter Volpi
Dipartimento di Informatica, Università di Pisa,
Corso Italia 40, 56125 Pisa, Italy.
EMAIL:levi@di.unipi.it TEL: +39-50-887246 FAX: +39-50-887226

Abstract: Abstract Interpretation and Set-Based Analysis are static analysis techniques. We show that, in the case of logic programs, Set-Based Analysis can be reconstructed as an instance of Abstract Interpretation.

Namely, we prove that *if P is a logic program, then the least solution of the system of equations extracted from P by the Set-Based Analysis can be expressed in terms of an abstract semantics definable using the Abstract Interpretation technique.*

Keywords: Logic Programming, Abstract Interpretation, Set-Based Analysis.

1 Introduction

Abstract Interpretation [1, 2] and *Set-Based Analysis* [5] are static analysis techniques.

The basic idea of *Abstract Interpretation* is to replace the domain of computation by an abstract domain and perform the computation over the latter. If the abstract domain is noetherian, the abstract semantics can be computed in a finite number of steps.

In the case of logic programs [6], *Set-Based Analysis* can roughly be described as follows:

- extract from the logic program P a set of equations S_0,
- perform on the equations set a finite sequence of transformations $S_0 \to S_1 \to \ldots \to S_n$,
- return the set of equations S_n.

All the transformations preserve the least solution of S_0. The main properties of S_n are:

- the semantics of the least solution of S_n is a conservative approximation of the least model of P,
- S_n can be used as a basis for logic program analysis because it is decidable whether an atom belongs to the least solution of S_n.

Set-Based Analysis was claimed by the authors of [5] not to be an instance of abstract interpretation. Patrick and Radhia Cousot have later shown [3] that Abstract Interpretation can be used to build a finite syntactic expression whose meaning is the semantics of the least solution of the set of equations extracted by the Set-Based Analysis. In the case of logic programs, we show a more direct result, using much simpler techniques. Namely we show that: *the least solution of the system of equations extracted from a logic program by the Set-Based Analysis can be expressed in terms of an abstract semantics.*

2 Preliminary definitions

Let $\Sigma \overset{\text{def}}{=} (Cos(\Sigma), Var(\Sigma), Fun(\Sigma), Pre(\Sigma))$ a *signature* where $Cos(\Sigma)$ is a finite set of *constant symbols*, $Var(\Sigma)$ is a denumerable set of *variable symbols*, $Fun(\Sigma)$ is a finite set of *function symbols* and $Pre(\Sigma)$ is a finite set of *predicate symbols*. We assume that $Cos(\Sigma)$, $Var(\Sigma)$, $Fun(\Sigma)$ and $Pre(\Sigma)$ are pairwise disjoint and that exist a function $arity : Fun(\Sigma) \cup Pre(\Sigma) \to \omega . f \in Fun(\Sigma) \Rightarrow arity(f) \geq 1$.

$Ter(\Sigma)$ ($GroTer(\Sigma)$) is the set of (ground) terms buit over the signature Σ. $Ato(\Sigma)$ ($GroAto(\Sigma)$) is the set of (ground) atoms built over the signature Σ. $Exp(\Sigma) \overset{\text{def}}{=} Ter(\Sigma) \cup Ato(\Sigma)$. $GroExp(\Sigma) \overset{\text{def}}{=} GroTer(\Sigma) \cup GroAto(\Sigma)$. $Body(\Sigma) \overset{\text{def}}{=} \{(b_1, b_2, \ldots, b_n) : b_1, b_2, \ldots, b_n \in Ato(\Sigma)\}$. $Cla(\Sigma) \overset{\text{def}}{=} \{h \leftarrow \overline{B}. : h \in Ato(\Sigma), \overline{B} \in Body(\Sigma)\}$. $Pro(\Sigma) \overset{\text{def}}{=} \wp(Cla(\Sigma))$ is the set of logic programs built over the signature Σ.

If $e \in Exp(\Sigma)$, $var(e)$ is the set of variable symbols that occours in e and $Gro_\Sigma(e)$ is the set of ground instances of e in Σ.

If X and Y are sets then a *partial function from X to Y* is a set $f \subseteq X \times Y$ for which $\forall x, y, y'.(x, y) \in f, (x, y') \in f \Rightarrow y = y'$. We write $X \rightharpoonup Y$ for the set of all partial functions from X to Y. Moreover, if $f \in X \rightharpoonup Y$ then $dom(f) \overset{\text{def}}{=} \{x : \exists y \in Y.(x, y) \in f\}$ and ϵ is the partial function whose domain is empty.

Finally, $Sub(\Sigma) \overset{\text{def}}{=} \{\sigma : \sigma \in Var(\Sigma) \rightharpoonup Ter(\Sigma) : x \in dom(\Sigma) \rightarrow \sigma(x) \neq x\}$ is the set of substitution built over the signature Σ.

3 Set-Based Analysis for logic programs

In this section we describe a re-elaboration of concepts concerning Set-Based Analysis taken from [5]. If $P \in Pro(\Sigma)$, the meaning of the least solution of the system of equations extracted from P by the Set-Based Analysis is $\tau_P \uparrow \omega$. The definition of τ_P is based on the concept of *set-substitution*. A set-substitution is like an ordinary substitution except that variables are mapped onto sets of ground terms rather than to terms. We write $Sub^*(\Sigma)$ for the set of all set-substitutions built over the signature Σ.

Definition 1 $Sub^*(\Sigma) \overset{\text{def}}{=} Var(\Sigma) \rightharpoonup \wp(GroTer(\Sigma))$

$(Sub^*(\Sigma), \leq^*)$ is a complete lattice where $\leq^*$ is defined as follows.

Definition 2 $\forall \psi_1, \psi_2 \in Sub^*(\Sigma).\psi_1 \leq^* \psi_2 \overset{\text{def}}{\Leftrightarrow} \forall < x, T_1 > \in \psi_1.\exists < x, T_2 > \in \psi_2.T_1 \subseteq T_2$

Let $(Sub^*(\Sigma)_\perp, \leq^*_\perp)$ be the lifting [8] of $(Sub^*(\Sigma), \leq^*)$. Now we define a function Ψ which takes as input a collection S of variables and a collection Θ of substitutions and returns a single set-substitution or $\perp$.

Definition 3 *The function* $\Psi : \wp(Var(\Sigma)) \times \wp(Sub(\Sigma)) \rightarrow Sub^*(\Sigma)_\perp$ *is defined as*

$$\forall S \subseteq Var(\Sigma), \forall \Theta \subseteq Sub(\Sigma).\ \Psi(S, \Theta) \overset{\text{def}}{=} \begin{cases} \perp & \text{if } \Theta = \emptyset \\ \psi & \text{otherwise} \end{cases}$$

where:

$$dom(\psi) \overset{\text{def}}{=} S \cap \{x : \exists \sigma \in \Theta. x \in dom(\sigma)\}, \quad \forall x \in dom(\psi).\psi(x) \overset{\text{def}}{=} \bigcup_{\sigma \in \Theta,\, x \in dom(\sigma)} \sigma(x).$$

A set-substitution can be applied to expressions. The result of the application $\mid E \mid \psi$ of the set-substitution ψ to an expressions $E \in Exp(\Sigma)$ is a set of ground instances of E as shown by the following definition.

Definition 4 *The function* $|.| : Exp(\Sigma) \times Sub^*(\Sigma) \rightarrow \wp(GroExp(\Sigma))$ *is defined as*

- $\forall \psi \in Sub^*(\Sigma), \forall x \in Var(\Sigma).\ |x|\psi \overset{\text{def}}{=} \begin{cases} \psi(x) & \text{if } x \in dom(\psi) \\ GroTer(\Sigma) & \text{otherwise} \end{cases}$
- $\forall \psi \in Sub^*(\Sigma), \forall c \in Const(\Sigma).\ |c|\psi \overset{\text{def}}{=} \{c\}$
- $\forall \psi \in Sub^*(\Sigma), \forall f \in Fun(\Sigma)$. *arity* $(f)=n$, $\forall t_1, t_2, \ldots, t_n \in Ter(\Sigma)$.
$$|f(t_1, t_2, \ldots, t_n)|\psi \overset{\text{def}}{=} \{f(s_1, s_2, \ldots, s_n) : \forall i = 1, 2, \ldots, n.s_i \in |t_i|\psi\}$$
- $\forall \psi \in Sub^*(\Sigma)$, $\forall p \in Pre(\Sigma)$. *arity* $(p)=n$, $\forall t_1, t_2, \ldots, t_n \in Ter(\Sigma)$.
$$|p(t_1, t_2, \ldots, t_n)|\psi \overset{\text{def}}{=} \{p(s_1, s_2, \ldots, s_n) : \forall i = 1, 2, \ldots, n.s_i \in |t_i|\psi\}.$$

Finally we define the approximate immediate consequences operator τ_P.

Definition 5 *If* $P \in Pro(\Sigma)$ *the operator* $\tau_P : \wp(GroAto(\Sigma)) \rightarrow \wp(GroAto(\Sigma))$ *is defined as*
$\forall J \subseteq GroAto(\Sigma)$.
$$a \in \tau_P(J) \overset{\text{def}}{\Leftrightarrow} \exists h \leftarrow \overline{B}. \in P.(\psi = \Psi(var(h), \{\sigma \in Sos^*(\Sigma) : [\overline{B}]\sigma \subseteq J\}) \neq \perp) \wedge (a \in \mid h \mid \psi).$$

An example of computation of $\tau_P \uparrow \omega$ is the following.

Example 1 *Let* $P = \{p(f(a,b)).,p(f(b,a)).,r(X) \leftarrow p(f(X,X)).,s(f(Y,Z)) \leftarrow p(f(Y,Z)).\}$.

- $\tau_P \uparrow 0 = \emptyset$
- $\tau_P \uparrow 1 = \{p(f(a,b)), p(f(b,a))\}$
- $\tau_P \uparrow 2 = \{p(f(a,b)), p(f(b,a)), s(f(a,a)), s(f(a,b)), s(f(b,a)), s(f(b,b))\}$
- $\tau_P \uparrow 3 = \tau_P \uparrow 2 = \tau_P \uparrow \omega$.

4 Denotational semantics

This section provides a denotational semantics for logic programs and a family of semantics obtained by using Abstract Interpretation.

4.1 Denotational Semantics of logic programs

In this subsection we define the denotational semantics of logic programs.

Definition 6 *The concrete domain is the complete lattice* $(C, \leq_C)$ *where:*

- $C \overset{\text{def}}{=} \{f \in Cla(\Sigma) \rightharpoonup \wp(GroAto(\Sigma)) :< h \leftarrow \overline{B}., A > \in f \Rightarrow A \subseteq Gro_\Sigma(h)\}$
- $\forall f_1, f_2 \in C. f_1 \leq_C f_2 \overset{\text{def}}{\Leftrightarrow} \forall < c, A_1 > \in f_1. \exists < c, A_2 > \in f_2. A_1 \subseteq A_2$.

Every element of C is a partial function from clauses to sets of ground atoms. If f is an element of the set C and $c = h \leftarrow \overline{B}.$ is an element of the f's domain, then $f(c)$ is a set of ground instances of h.

The denotational semantics of the logic program P, $Den[P]$, is defined as the least fixpoint of the operator Y_P, defined as follows.

Definition 7 *If* $P \in Pro(\Sigma)$, *then* $Y_P : C \to C$ *is defined as*

$$\forall f \in C. Y_P(f) \overset{\text{def}}{=} Lub_C\{c \lhd f : c \in P\} ,$$

where $\lhd : Cla(\Sigma) \times C \to C$ *is defined as*

$$\forall c = h \leftarrow \overline{B}. \in Cla(\Sigma), \forall f \in C. c \lhd f \overset{\text{def}}{=} Inst_\Sigma(c, Unif_\Sigma(\overline{B}, f))$$

and

1. $Unif_\Sigma : Body(\Sigma) \times C \to \wp(Sub(\Sigma))$ *is defined as*

   $$\forall (b_1, b_2, \ldots, b_n) \in Body(\Sigma), \forall f \in C.\ Unif_\Sigma((b_1, b_2, \ldots, b_n), f) \overset{\text{def}}{=} \begin{cases} \{\epsilon\} & \textit{if n=0} \\ \Theta & \textit{otherwise} \end{cases}$$

   where

   - $\Theta \overset{\text{def}}{=} \{\sigma \in Sub(\Sigma) : \forall i = 1, 2, \ldots, n. \exists c_i \in dom(f).[b_i]\sigma \in f(c_i)\}$.

2. $Inst_\Sigma : Cla(\Sigma) \times \wp(Sub(\Sigma)) \to C$ *is defined as:*

   $$\forall c = h \leftarrow \overline{B}. \in Cla(\Sigma), \forall \Theta \in \wp(Sub(\Sigma)).\ Inst_\Sigma(c, \Theta) \overset{\text{def}}{=} \{< c, \{[h]\sigma : \sigma \in \Theta\} >\}$$

Y_P is a continuous function. Hence $Den[P] \overset{\text{def}}{=} Y_P \uparrow \omega$. An example of $Den[P]$ is the following.

Example 2 *If* P *is the logic program in example 1, then:*
$Den[P] = \{\ < p(f(a,b)).,\{p(f(a,b))\} >,\ \ < p(f(b,a)).,\{p(f(b,a))\} >,$
$< r(X) \leftarrow p(f(X,X)).,\emptyset >,\ \ < s(f(Y,Z)) \leftarrow p(f(Y,Z)).,\{s(f(a,b)), s(f(b,a))\} >\ \}.$

$Den[P]$ is related to the least Herbrand model $T_P \uparrow \omega$ by the following equation.
$T_P \uparrow \omega \overset{\text{th}}{=} \Pi(Den[P])$, where $\Pi : C \to \wp(GroAto(\Sigma))$ is defined as $\forall f \in C. \Pi(f) \overset{\text{def}}{=} \bigcup_{c \in dom(f)} f(c)$.

4.2 A family of abstract denotational semantics

If (α, γ) is a Galois insertion of $(A, \leq_A)$ in $(C, \leq_C)$ (c.f.,e.g. [1, 2]) then the *abstract denotational semantics*, $Den^a[P]$, of the logic program P is defined as the least fixpoint of the operator Y_P^a, formally defined as follows.

Definition 8 *If $P \in Pro(\Sigma)$, then $Y_P^a : A \to A$ is defined as*

$$\forall g \in A. Y_P^a(g) \overset{\text{def}}{=} Lub_A\{c \lhd^a g : c \in dom(g)\},$$

where $\lhd^a : Cla(\Sigma) \times A \to A$ is defined as

$$\forall c \in Cla(\Sigma), \forall g \in A. c \lhd^a g \overset{\text{def}}{=} \alpha(c \lhd \gamma(g)).$$

We can prove that if α and γ are continuous functions, then Y_P^a is a continuous function. Hence $Den^a[P] \overset{\text{th}}{=} Y_P^a \uparrow \omega$.

A particular class of Galois insertions is the class of *observables*. An *observable* (α, γ) of $(A, \leq_A)$ in $(C, \leq_C)$ is a Galois insertion of $(A, \leq_A)$ in $(C, \leq_C)$ such that:

- $(A, \leq_A)$ satisfies the following properties:
  - $A \subseteq Cla(\Sigma) \rightharpoonup L$,
  - $(L, \preceq_L)$ is a complete lattice,
  - $\forall g_1, g_2 \in A. g_1 \leq_A g_2 \overset{\text{def}}{\Leftrightarrow} \forall < c, l_1 > \in g_1. \exists < c, l_2 > \in g_2. l_1 \preceq_L l_2$.
- $(\alpha, \gamma) : (C, \leq_C) \rightleftharpoons (A, \leq_A)$ satisfies the following properties:
  - $\forall f \in C. dom(\alpha(f)) = dom(f)$,
  - $\exists abs : \{f \in C : card(dom(f)) = 1\} \to L.\ \forall c \in dom(f). \alpha(f)c = abs(\{< c, f(c) >\})$.

An observable (α, γ) of $(A, \leq_A)$ in $(C, \leq_C)$ define an abstraction relation between the concrete domain $(C, \leq_C)$ and the abstract domain $(A, \leq_A)$. Note that if $f \in C$ and $c \in dom(f)$, then $\alpha(f)c$ depend on $f(c)$ and on the syntactic structure of the clause c. Moreover, if (α, γ) is an observable of $(A, \leq_A)$ in $(C, \leq_C)$, then $\forall g \in A. Y_P^a(g) \overset{\text{th}}{=} \alpha(Y_P \gamma(g))$.

5 The observable for the Set-Based Analysis

In this section we describe the relation between Set-Based Analysis and Abstract Interpretation for the logic programming paradigm.

Definition 9 *The abstract domain is the complete lattice $(A, \leq_A)$ where:*

- $A \overset{\text{def}}{=} \{g \in Cla(\Sigma) \rightharpoonup Sub^*(\Sigma)_\perp : \forall < h \leftarrow \overline{B}., \xi > \in g.(\xi = \perp) \vee (dom(\xi) = var(h))\}$,
- $\forall g_1, g_2 \in A. g_1 \leq_A g_2 \overset{\text{def}}{\Leftrightarrow} \forall < c, \xi_1 > \in g_1. \exists < c, \xi_2 > \in g_2. \xi_1 \leq_\perp^* \xi_2$.

If g is an element of the set A and c is an element of g's domain, then $g(c)$ is the symbol $\perp$ or a set-substitution ψ. The ψ's domain is the set of variables that occur in the head of c.

The abstraction function from the concrete domain $(C, \leq_C)$ to the abstract domain $(A, \leq_A)$ is defined as follows.

Definition 10 *The function $\alpha : C \to A$ is defined as*

1. $\forall f \in C.\ dom(\alpha(f)) \overset{\text{def}}{=} dom(f)$,
2. $\forall f \in C, \forall c = h \leftarrow \overline{B}. \in dom(f).\ \alpha(f)c \overset{\text{def}}{=} \begin{cases} \perp & \textit{if } f(c) = \emptyset \\ \Delta & \textit{otherwise} \end{cases}$

   where:

   - $dom(\Delta) \overset{\text{def}}{=} var(h),\ \forall x \in dom(\Delta). \Delta(x) \overset{\text{def}}{=} \{\sigma(x) : \exists \sigma \in Sub(\Sigma).[h]\sigma \in f(c)\}$.

Note that, if $f \in C$, $h \leftarrow \overline{B}. \in dom(f)$ and h is a ground atom, then $\alpha(f)c = \epsilon$.
An example of abstraction is the following.

Example 3 *If* $f \in C$, $f(p(f(X,X)) \leftarrow r(X).) = \{p(f(a,a)), p(f(b,b))\}$, *and* $g = \alpha(f)$ *then* $g(p(f(X,X)) \leftarrow r(X).) = \{< X, \{a,b\} >\}$.

The concretization function γ is introduced in the following.

Definition 11 *The function* $\gamma : A \rightarrow C$ *is defined as*

1. $\forall g \in A.\ dom(\gamma(g)) \overset{\text{def}}{=} dom(g)$,
2. $\forall g \in A, \forall c = h \leftarrow \overline{B}. \in dom(g).\ \gamma(g)c \overset{\text{def}}{=} \begin{cases} \emptyset & \text{if } g(c) = \bot \\ \mid h \mid g(c) & \text{otherwise.} \end{cases}$

An example of γ's application is the following.

Example 4 *If* $g \in A$, $g(p(f(X,X) \leftarrow r(X).) = \{< X, \{a,b\} >\}$, *and* $f = \gamma(g)$ *then* $f(p(f(X,X)) \leftarrow r(X).) = \{p(f(a,a)), p(f(a,b)), p(f(b,a)), p(f(b,b))\}$.

The functions α and γ satisfy the following properties:

1. α and γ are continuous functions. Hence $Den^a[P] = Y^a_P \uparrow \omega$.
2. (α, γ) is an observable. Hence $\forall g \in A. Y^a_P(g) = \alpha(Y_P \gamma(g))$.

An example of abstract denotation is the following.

Example 5 *If P is the program in the example 1, then*

- $Y^a_P \uparrow 0 = \epsilon$ *(ϵ is the abstract function whose domain is empty)*
- $Y^a_P \uparrow 1 = \alpha(Y_P \gamma(Y^a_P \uparrow 0)) = \{\ < p(f(a,b)).,\epsilon >,\ < p(f(b,a)).,\epsilon >,\ < r(X) \leftarrow p(f(X,X)).,\bot >,\ < s(f(Y,Z)) \leftarrow p(f(Y,Z)).,\bot >\ \}$
- $Y^a_P \uparrow 2 = \alpha(Y_P \gamma(Y^a_P \uparrow 1)) = \{\ < p(f(a,b)).,\epsilon >,\ < p(f(b,a)).,\epsilon >,\ < r(X) \leftarrow p(f(X,X)).,\bot >,\ < s(f(Y,Z)) \leftarrow p(f(Y,Z)).,\{< Y,\{a,b\} >, < Z,\{a,b\} >\} >\ \}$
- $Y^a_P \uparrow 3 = \alpha(Y_P \gamma(Y^a_P \uparrow 2)) = Y^a_P \uparrow 2 = Den^a[P]$.

Finally we can prove the following equality $\tau_P \uparrow \omega = \Pi(\gamma(Den^a[P]))$. (1)
The main theorem needed to prove (1) is the following.

Theorem 1

If

$$f \in C, \quad c = h \leftarrow \overline{B}. \in dom(f), \quad U_\Sigma(\overline{B}, f) \neq \emptyset,$$
$$var(h) \cap var(\overline{B}) = \{x_1, \ldots, x_h\}, \quad var(h) \setminus var(\overline{B}) = \{y_1, \ldots, y_k\},$$

then

$\gamma(\alpha(c \lhd f))c$ *is the set of ground instances of h obtained by replacing the j-th occurrence of the variable:*

- x_i *with a term* $t_{i,j} = \sigma(x_i)$, *for some* $\sigma \in U_\Sigma(\overline{B}, f)$,
- y_i *with a term* $t_{i,j} \in GroTer(\Sigma)$.

Hence, if $W_P \overset{\text{def}}{=} \lambda f \in C.\gamma(\alpha(Y_P f))$, then $\forall f \in C.\tau_P(\Pi(f)) \overset{\text{th}}{=} \Pi(W_P f)$. Therefore we can prove that $\forall n \in \omega.\tau_P \uparrow n \overset{\text{th}}{=} \Pi(W_P \uparrow n)$.

Finally, $\forall n \in \omega.\gamma(Y^a_P \uparrow n) \overset{\text{th}}{=} W_P \uparrow n$ so $\forall n \in \omega.\Pi(\gamma(Y^a_P \uparrow n)) \overset{\text{th}}{=} \tau_P \uparrow n$. Hence, by continuity of Π, γ and Y^a_P, equality (1) holds.

Therefore *the semantics of the system of equations extracted by the Set-Based Analysis from P, i.e.* $\tau_P \uparrow \omega$*, can be expressed in terms of an abstract semantics, i.e.* $Den^a[P]$*, definable using Abstract Interpretation.*

An example of (1) is the following.

Example 6 *Consider the abstract semantics of the example 5. Then*

- $\gamma(Den^a[P]) =$
  $\{ < p(f(a,b)).,\{p(f(a,b))\} >, < p(f(b,a)).,\{p(f(b,a))\} >, < r(X) \leftarrow p(f(X,X)).,\emptyset >,$
  $< s(f(Y,Z)) \leftarrow p(f(Y,Z)).,\{s(f(a,a)),s(f(a,b)),s(f(b,a))\ ,s(f(b,b))\} > \}.$
- $\Pi(\gamma(Den^a[P])) = \{p(f(a,b)),p(f(b,a)),s(f(a,a)),s(f(a,b)),s(f(b,a)),s(f(b,b))\}$ *hence, for the result shown in example 1, the equality (1) holds.*

6 Abstract semantics and logic programs

If P is a logic program, $\tau_P \uparrow \omega$ can be expressed in terms of "standard" semantics of an approximate logic program P_{type} [4]. P_{type} is obtained by applying a syntactic transformation. In this section, we show a different syntactic transformation Tr. Let $Tr : Cla(\Sigma) \to Cla(\Sigma)$ be defined as

$$\forall c = h \leftarrow \overline{B}. \in Cla(\Sigma).Tr(c) \overset{\text{def}}{=} h' \leftarrow \overline{B}'.$$

where:

- h' is an atom obtained by replacing in h the j-th occurrence of the variable x_i by the new variable $y_{i,j}$,
- $\overline{B}'$ is the sequence of atoms $\overline{B}_{1,1},\ldots,\overline{B}_{1,n_1},\ldots,\overline{B}_{m,1},\ldots,\overline{B}_{m,n_m}$, where, for each $y_{i,j}$ variable in h', the body $\overline{B}_{i,j}$ is obtained by replacing in $\overline{B}$ each variable by a new variable except for x_i (if x_i occurs in $\overline{B}$), which is replaced by $y_{i,j}$.

An example of the clause tranformation by Tr is the following.

Example 7 $Tr(p(X_1,X_1) \leftarrow p(X_1,X_1).) = p(Y_{1,1},Y_{1,2}) \leftarrow p(Y_{1,1},Y_{1,1}),p(Y_{1,2},Y_{1,2}).$

If $P \in Pro(\Sigma)$, we define $\overline{P} \overset{\text{def}}{=} \bigcup_{c\in P} Tr(c)$. An example of $\overline{P}$ is the following.

Example 8 *If P is the program in the example 1, then* $\overline{P} =\{\ p(f(a,b)).\ ,\ \ p(f(b,a)).\ ,\ \ r(Y) \leftarrow p(f(Y,Y)).\ ,\ \ s(f(Y_{1,1},Y_{2,1})) \leftarrow p(f(Y_{1,1},N_1)),p(f(N_2,Y_{2,1})).\ \}.$

We can show that $\forall n \in \omega, \forall c \in P.Y_{\overline{P}} \uparrow n(Tr(c)) \overset{\text{th}}{=} \gamma(Y^a_P \uparrow n)(c)$ so $\forall n \in \omega.\Pi(Y_{\overline{P}} \uparrow n) \overset{\text{th}}{=} \Pi(\gamma(Y^a_P \uparrow n))$. Therefore $T_{\overline{P}} \uparrow \omega \overset{\text{th}}{=} \Pi(Den[\overline{P}]) \overset{\text{th}}{=} \Pi(\gamma(Den^a[P]))$.

So we can say that *the semantics of* $\overline{P}$*, i.e.* $T_{\overline{P}} \uparrow \omega$*, is justified in terms of an abstract semantics obtained using Abstract Interpretation, i.e.* $Den^a[P]$.

7 Conclusion

We have described a Galois insertion which captures the abstraction made by Set-Based Analysis in the logic programming case. The Galois insertion defines an abstract semantics which can be related with the semantics of a logic program $\overline{P}$. $\overline{P}$ is a finite syntactic expression which satisfies the following properties.

- the least model of $\overline{P}$ is the semantics of the least solution of the system of equations extracted from P by Set-Based Analysis.
- if a is an atom, then it is decidable the problem of establishing whether a is an element of the least model of $\overline{P}$ [4].
- the least model of $\overline{P}$ can be expressed using a tree automaton [7].

References

[1] P. Cousot and R. Cousot, *Abstract interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* In *Proc. Fourth ACM Symp. Principles of Programming Languages*, pages 238-252, 1977.

[2] P. Cousot and R. Cousot, *Systematic Design of Program Analysis Frameworks.* In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pages 269-282, 1979.

[3] P. Cousot and R. Cousot, *Formal language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation.* In *Conference Record of FPCA'95 - Conference on Functional Programming Languages and Computer Architecture*, pages 170-181, 1995.

[4] T. Fruhwirth, E. Shapiro, M. Y. Vardi and E. Yardeni, *Logic Programs as Types for Logic Programs.* In *Proceeding of the* 6th *IEEE-LICS*, pages 300-309, 1991.

[5] N. Heintze and J. Jaffar, *A finite presentation theorem for approximating logic programs.* In *Proceedings of the* 17th *ACM POPL*, pages 197-209, 1990.

[6] J. W. Lloyd, *Foundations of Logic Programming.* Springer-Verlag 1987.

[7] JM. Talbot, S. Tison and P. Devienne, *Set-Based Analysis for Logic Programming and Tree Automata*, In Pascal Van Hentenryck (ed.) volume 1302 of *Lecture Notes in Computer Science* : Static Analysis 4th International Symposium, SAS'97, pages 127-140. Springer-Verlag, 1997.

[8] G. Winskel, *The Formal Semantics of Programming Languages: An Introduction* In Michael Garey and Albert Meyer (ed.), *Foundations of Computing*, The MIT Press, 1993.