Analysis of normal CLP programs

François FAGES
Roberta GORI

LIENS - 97 - 17

Analysis of normal CLP programs

François FAGES
Roberta GORI*

LIENS - 97 - 17

December 1997

Laboratoire d'Informatique de l'Ecole Normale Supérieure
45 rue d'Ulm 75230 PARIS Cedex 05

Tel : (33)(1) 44 32 30 00
Adresse électronique : fages@dmi.ens.fr

* Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy

Adresse électronique : gori@di.unipi.it

Analysis of normal CLP programs

François Fages
LIENS CNRS, Ecole Normale Supérieure,
45 rue d'Ulm, 75005 Paris, France,
fages@dmi.ens.fr
Roberta Gori
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy
gori@di.unipi.it

Abstract

In this paper we present a dataflow analysis method for normal constraint logic programs interpreted with finite failure or constructive negation. We apply our method to a well known analysis for logic programs on the Herbrand Universe: the depth(k) analysis for approximating the set of computed answer constraints. The analysis is correct w.r.t. SLDNF resolution and optimal w.r.t. constructive negation.

1 Introduction

The semantic-based data flow analysis of definite constraint logic programs (CLP) is nowadays standard practice. Several systems have been implemented and proved useful at either helping the programmer to find errors, through type checking and declarative diagnosis, or improving program efficiency, through program transformation and compilation (see e.g. [12, 3, 19]).

Much work has been done in this context using the theory of abstract interpretation [6] for helping the design of such analyzers and proving their correctness. A key point in abstract interpretation is the choice of a reference semantics from which one can abstract the properties of interest. While it is always possible to use the operational semantics as it contains all information, it is possible to get rid of useless details, by choosing a more abstract semantics as reference semantics. Choosing the most abstract logical least model semantics of definite logic programs limits the analysis to type inference properties, that approximate the ground success set. Non-ground model semantics have thus been developed, under the name of the S-semantics approach [1], and proved useful for a wide variety of goal-independent analysis ranging from groundness, to sharing, call patterns, etc. All the intermediate fixpoint semantics of definite CLP programs comprised between the most abstract logical one and the most concrete operational one, form in fact a hierarchy of semantics related by abstract interpretation, in which one can define a notion of the best reference semantics [13].

On the other hand, much less work has been done on the analysis of normal (constraint) logic programs, although the finite failure principle, and hence SLDNF resolution, are standard practice. One reason for this is that the formal operational semantics of normal logic programs (either SLDNF resolution [15] or constructive negation [21]) are quite complicated, and the generalization of the S-semantics, which captures only the set of computed answer constraints, to normal logic programs has been open for a while [1].

In this paper we present an analysis method for normal CLP programs. It is based on the generalized S-semantics given in [9] and on the hierarchy described in [10]. One important contribution of the paper is the definition of a normal form for first order constraints on the Herbrand Universe, which is suitable for analysis. In fact the normal form allows us to define an abstraction function which is a congruence wrt the equivalence on constraints induced by the Clark's equality theory. On the domain of constraints in normal form, we define a depth(k) analysis for normal logic programs which approximates the constraint answer set for positive and negative atoms. We show that the analysis is correct and optimal.

This analysis can be compared with other work for the analysis of normal logic programs. In [18], Marriott and Sondergaard proposed a framework based on Fitting's semantics [11]. Fitting's least three-valued model semantics is an abstraction (in fact a non recursively enumerable one yet easier to define) of Kunen's three-valued logical semantics [14] that is more faithful to SLDNF resolution [15] and complete w.r.t. constructive negation [21]. Therefore the choice of Fitting's semantics as reference semantics implies *de facto* a loss of precision in the analysis. Furthermore because these reference semantics are ground, the analysis based on them are limited to type inference properties. We give an example of depth(k) analysis which illustrates the differences between both methods.

2 Preliminaries

2.1 Constraints

The first-order language of constraints is defined on a countably infinite set of variables V and on a signature Σ composed of a set of predicate symbols containing *true* and $=$, and of sets of n-place function symbols for each arity n (constants are functions with arity 0). A *primitive constraint* is an atomic proposition of the form $p(t_1, ..., t_n)$, where p is a predicate symbol in Σ and the t_i's are Σ, V-terms. A *constraint* is a well-formed first-order Σ, V-formula. The set of free variables in an expression e is denoted by $Var(e)$. Sets of variables will be denoted by $X, Y, ...$ For a constraint c, we shall use the notation $\exists c$ (resp. $\forall c$) to represent the constraint $\exists X\ c$ (resp. $\forall X\ c$) where $X = Var(c)$. A constraint is in *prenex form* if all its quantifiers are in the head.

The intended interpretation of constraints is defined by fixing a Σ-structure $\mathcal{X}$. An $\mathcal{X}$-valuation for a Σ, V-expression is a mapping $\theta : V \to \mathcal{X}$ which extends by morphism to terms and primitive constraints. Logical connectives and quantifiers are interpreted as usual, a constraint c is $\mathcal{X}$-solvable iff $\mathcal{X} \models \exists c$.

The only property required on $\mathcal{X}$ is that constraints are decidable in $\mathcal{X}$, so that $\mathcal{X}$ can be presented by an axiomatic theory $th(\mathcal{X})$ satisfying:

1. (soundness) $\mathcal{X} \models th(\mathcal{X})$,
2. (completeness) for any constraint c $th(\mathcal{X}) \models \exists c$ or $th(\mathcal{X}) \models \neg\exists c$.

As a constraint is any Σ, V-formula, $th(\mathcal{X})$ is a complete theory, and thus all models of $th(\mathcal{X})$ are elementary equivalent.

For instance, *Clark's equational theory* (CET) provides such a complete decidable theory for the structure $\mathcal{H}$ of the *Herbrand domain* with first-order equality constraints [17] [14]. In the following all our examples will be given in this domain.

2.2 Normal CLP($\mathcal{X}$) programs

$CLP(\mathcal{X})$ programs are defined using an extra finite set of predicate symbols Π disjoint from Σ. An *atom* has the form $p(t_1, ..., t_n)$ where $p \in \Pi$ and the t_i's are Σ, V-terms. A *literal* is either an atom (positive literal) or a negated atom $\neg A$ (negative literal). A *normal (resp. definite) $CLP(\mathcal{X})$ program* is a finite set of clauses of the form $A \leftarrow c|L_1, ..., L_n$, where $n \geq 0$, A is an atom, called the head, c is a constraint, and $L_1, ..., L_n$ are literals (resp. atoms). The *local variables* of a program clause is the set of free variables in the clause which do not occur in the head.

In order to characterize precise operational aspects of CLP programs, such as sets of computed answer constraints, the formal semantics of $CLP(\mathcal{X})$ programs will be defined by sets of constrained atoms. With $Var(A)$ we intend the free variables in the atom A. A *constrained atom* is a couple $c|A$ where c is an $\mathcal{X}$-solvable constraint such that $Var(c) \subseteq Var(A)$. The set of constrained atoms is denoted by $\mathcal{B}_{\mathcal{X}}$. A *constrained interpretation* is a subset of $\mathcal{B}_{\mathcal{X}}$. A *three-valued* or *partial constrained interpretation* is a couple of constrained interpretations $< I^+, I^- >$, one I^+ representing the true things, and one I^- for the false things (note that because of our interest in abstract interpretations we do not impose any consistency condition).

2.3 Galois insertions and abstract interpretation

Abstract interpretation [6, 7] is a theory developed to reason about the abstraction relation between two different semantics. The theory requires the two semantics to be defined on domains which are poset. $(\mathcal{C}, \preceq)$ (the concrete domain) is the domain of the concrete semantics, while $(\mathcal{A}, \leq)$ (the abstract domain) is the domain of the abstract semantics. The partial order relations reflect an approximation relation. The two domains are related by a pair of functions α (*abstraction*) and γ (*concretization*), which form a Galois insertion.

Definition 2.1 (Galois insertion) *Let $(\mathcal{C}, \preceq)$ be the concrete domain and $(\mathcal{A}, \leq)$ be the abstract domain. A* Galois insertion $\langle \alpha, \gamma \rangle : (\mathcal{C}, \preceq) \rightleftharpoons (\mathcal{A}, \leq)$ *is a pair of maps $\alpha : C \to \mathcal{A}$ and $\gamma : \mathcal{A} \to \mathcal{C}$ such that*

1. *α and γ are monotonic,*
2. *$\forall x \in C. \, x \preceq (\gamma \circ \alpha)(x)$ and*
3. *$\forall y \in \mathcal{A}. \, (\alpha \circ \gamma)(y) = y$.*

Given a concrete semantics and a Galois insertion between the concrete and the abstract domain, we want to define an abstract semantics. The concrete semantics is the least fixpoint of a semantic function $F : \mathcal{C} \to \mathcal{C}$. The abstract semantic function $\tilde{F} : \mathcal{A} \to \mathcal{A}$ is *correct* if $\forall x \in \mathcal{A}. \, \alpha(F(\gamma(x))) \preceq \tilde{F}(x)$. The abstract semantics is *optimal* (most precise) if the equality holds.

F is in turn defined as "composition" of "primitive" operators. Let $f : \mathcal{C}^n \to \mathcal{C}$ be one such an operator and assume that $\tilde{f}$ is its abstract counterpart. Then $\tilde{f}$ is *(locally) correct* w.r.t. f if

$$\forall x_1, \ldots, x_n \in \mathcal{A} \ \alpha(f(\gamma(x_1), \ldots, \gamma(x_n))) \preceq \tilde{f}(x_1, \ldots, x_n).$$

According to the theory, for each operator f, there exists an optimal (most precise) locally correct abstract operator $\tilde{f}$ defined as
$\tilde{f}(y_1, \ldots, y_n) = \alpha(f(\gamma(y_1), \ldots, \gamma(y_n)))$.

2.4 Constructive negation

Constructive negation is a principle of inference introduced by Chan for normal logic programs in [2], that generalizes smoothly to normal CLP($\mathcal{X}$) programs providing them with a sound and complete [21] operational semantics w.r.t. Kunen's logical semantics [14]. In the case of a normal CLP($\mathcal{X}$) program, Kunen's semantics is simply the set of three-valued consequences of the program's completion and the theory $th(\mathcal{X})$.

The S-semantics of definite logic programs [1] has been generalized to normal CLP($\mathcal{X}$) programs in [9] for a version of constructive negation, called constructive negation by pruning. The idea of the fixpoint operator, which captures the set of computed answer constraints, is to consider a *non-ground finitary* (hence continuous) version of Fitting's operator. Here we give a definition of the operator $T_P^{\mathcal{D}}$ which is parametric w.r.t. the domain $\mathcal{D}$ of constrained atoms and the operations on constraints.

Definition 2.2 *Let P be a normal CLP($\mathcal{X}$) program. $T_P^{\mathcal{D}}$ is an operator over $\mathcal{P}(\mathcal{B}_{\mathcal{D}}) \times \mathcal{P}(\mathcal{B}_{\mathcal{D}})$ defined by*

$$\begin{array}{ll}
T_P^{\mathcal{D}}(I)^+ = \{c|p(X) \in \mathcal{B}_{\mathcal{D}} : & \textit{there exist a clause in } P \textit{ with local variables } Y, \\
& C = p(X) \leftarrow d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n. \\
& c_1|A_1, ..., c_m|A_m \in I^+,\ c_{m+1}|A_{m+1}, ..., c_n|A_n \in I^- \\
& \textit{such that } c = \overline{\overline{\exists Y}}\ (d\ \overline{\overline{\wedge}} c_1 \overline{\overline{\wedge}} \ldots \overline{\overline{\wedge}} c_n) \\
& \textit{is } \mathcal{H}\textit{-satisfiable}\} \\
T_P^{\mathcal{D}}(I)^- = \{c|p(X) \in \mathcal{B}_{\mathcal{D}} : & \textit{for each clause defining } p \textit{ in } P \textit{ with loc. var. } Y_k, \\
& C_k = p(X) \leftarrow d_k|A_{k,1}, ..., A_{k,m_k}, \beta_k. \\
& \textit{there exist } e_{k,1}|A_{k,1}, ..., e_{k,m_k}|A_{k,m_k} \in I^-, \\
& e_{k,m_k+1}|A_{k,m_k+1}, ..., e_{k,n_k}|A_{k,n_k} \in I^+,\ n_k \geq m_k \\
& \textit{where for } m_k + 1 \leq j \leq n_k,\ \neg A_{k,j} \textit{ occurs in } \beta_k, \\
& \textit{such that } c = \overline{\overline{\bigwedge}}\ \overline{\overline{\forall}} Y_k\ (\overline{\overline{\neg}}\ d_k \overline{\overline{\vee}}\ e_{k,1} \ldots \overline{\overline{\vee}}\ e_{k,n_k}), \\
& \textit{is } \mathcal{H}\textit{-satisfiable}\}.
\end{array}$$

Where the operations $\overline{\overline{\exists}}, \overline{\overline{\forall}}, \overline{\overline{\neg}}, \overline{\overline{\vee}}, \overline{\overline{\wedge}}$, *are the corresponding operations on the constraint domain of* $\mathcal{D}$.

In the case of a normal CLP($\mathcal{X}$) program, the operator $T_P^{\mathcal{X}}$ defines with its least fixpoint a generalized S-semantics which is fully abstract for the observation of the set of computed answer constraints with constructive negation by pruning [9]. By soundness it approximates also the set of computed answer constraints under the SLDNF resolution rule, or under the Prolog strategy.

In [10] we have shown that this operator defines a hierarchy of reference semantics related by abstract interpretation, that extends the hierarchy defined by Giacobazzi for definite logic programs [13].

3 Normal forms in CET

In order to define abstractions on constrained atoms we need to define suitable normal forms for first-order constraints. Given a theory $th(\mathcal{X})$ we are interested in working with equivalence classes of constraints w.r.t. the equivalence of the constraints in $th(\mathcal{X})$. Namely c is equivalent to c' if $th(\mathcal{X}) \models c \leftrightarrow c'$. This is why we need the abstraction function on the concrete constraint domain to be a congruence. This is a necessary property since it permits to prescind from the syntactic form of the constraints.

We restrict our attention on normal CLP($\mathcal{H}$) programs and need to achieve this property in CET. We thus need to introduce a normal form for first-order equality constraints, in a similar way to what has been done for the analisys of definite programs where the normal form is the unification solved form [16]. Here we shall define a new notion of “false-simplified” normal forms, where the normal form that we consider is based on Colmerauer's solved forms for inequalities [4] and Maher's transformations for first-order constraints [17].

First let us motivate the need of a “false-simplified” form. Let us call a *basic constraint* an equality or an inequality between a variable and a term. The abstraction function will be defined inductively on the basic constraints, and it will sometimes (e.g. for groundness analysis) abstract to *true* some inequalities. Consider, for example, the following constraint $d = (X = f(a) \wedge X \neq f(a))$, which is $\mathcal{H}$-equivalent to $false$. If the abstraction of $X \neq f(a)$ is $true$ then the abstraction of d will be the abstraction of $X = a$, but that constraint cannot be $\mathcal{H}$-equivalent to the abstraction of $false$. Therefore we need to define a normal form where the constraints which are $\mathcal{H}$-equivalent to $false$, are eliminated.

Definition 3.1 *Consider a constraint d in prenex disjunctive form, $d = \Delta(\vee_i A_i)$, where Δ is a sequence of quantified variables and $\vee_i A_i$ is a finite disjunction. d is in a $false$-simplified form if, either there does not exist a proper subset I of the i's such that $\mathcal{H} \models \Delta(\vee_i A_i) \leftrightarrow \Delta(\vee_{i \in I} A_i)$, or such an I exists and there exists also a subset K of I, such that $\vee_{j \notin I} A_j$ is $\mathcal{H}$-equivalent to $\vee_{k \in K} A_k$.*

The latter condition assures that we really eliminate constraints that are $\mathcal{H}$-equivalent to $false$ and that are not just redundant in the constraint. Now the existence of a *false-simplified* form for any constraint can be proved simply with the following:

Algorithm 3.1 *Input: a constraint in prenex disjunctive form $d = \Delta(\vee_i A_i)$.*

1. *Consider the partition I and J of the A_i's, such that $A_i \in I$ if $\mathcal{H} \models \exists\Delta(A_i)$, $A_i \in J$ otherwise.*
2. *Repeat $I := I \cup S$ as long as there exists an $S \subseteq J$ such that $\mathcal{H} \models \exists\Delta(\vee_{i \in S} A_i)$ and for all $j \in S$ $\mathcal{H} \not\models \exists\Delta(\vee_{i \in (S \setminus \{j\})} A_i)$.*
3. *Let $S \in J \setminus I$ be any minimal set such that $\mathcal{H} \models \exists\Delta(\vee_{s \in S} A_s \vee_{i \in I} A_i)$ and $\mathcal{H} \models \exists\Delta(\vee_{s \in S} A_s \vee_{i \in I} A_i) \leftrightarrow d$, do $I := I \cup S$,*
4. *Output: $\Delta(\vee_{i \in I} A_i)$.*

The idea of the algorithm is to find a subset of the conjunctions A_i's (the A_i with $i \in I$) such that $\Delta(\vee_i A_i)$ with $i \in I$ is in *false-simplified* form. In the first step we select the A_i's such that $\Delta(A_i)$ is $\mathcal{H}$-satisfiable. In this case, in fact, A_i can not be $\mathcal{H}$-equivalent to $false$ and it can be put it in the set I. In the second step from the remaining A_i's we select the set of A_i's such that their Δ quantified disjunction is $\mathcal{H}$-satisfiable, since we check that all the A_i's are necessary for the quantified disjunction to be $\mathcal{H}$-satisfiable, the considered A_i's can not be $\mathcal{H}$-equivalent to $false$. At the end of this process if the resulting constraint is $\mathcal{H}$-equivalent to the input constraint we stop. Otherwise we add a minimum number of the not yet selected A_i's such the $\Delta(\vee_i A_i)$ for the selected i's is $\mathcal{H}$-equivalent to the input constraint. Since we add a minimum number of not yet selected A_i's, we are sure that the resulting constraint is in *false-simplified* form.

Theorem 3.1 *For any input constraint $c = \Delta(\vee_i A_i)$, algorithm 3.1 terminates and computes a* false-simplified *form of c.*

Note that all the false-simplified forms of a constraint c are $\mathcal{H}$-equivalent. Now we can define the normal form, $Res(c)$, of a first-order equality constraint c, as the result of the following steps:

1. put the constraint c in prenex disjunctive normal form c_1,
2. compute a unification solved form for each conjunction of equalities
3. for each equality $x = t$ in a conjunction, substitute t to x at each occurrence of x in the inequalities of the same conjunction.
4. simplify the inequalities by applying the following rules:
   (a) replace $f(t_1, \ldots, t_n) \neq f(s_1, \ldots, s_n)$ by $t_1 \neq s_1 \vee \ldots \vee t_n \neq s_n$.
   (b) replace $f(t_1, \ldots, t_n) \neq g(s_1, \ldots, s_n)$ by $true$.
   (c) replace $t \neq x$ by $x \neq t$ if t is not a variable.
5. Return a $false$-simplified form of the resulting constraint .

Note that all these steps preserve $\mathcal{H}$-equivalence, the fourth step is Colmerauer's simplification algorithm for inequalities [4] and the first three transforms are usual for CET formulas [17], hence we get:

Proposition 3.1 $\mathcal{H} \models \phi \leftrightarrow Res(\phi)$

Our concrete constraints domain $\mathcal{NC}$ will be the subset of constraints in $\mathcal{C}$ which are in *normal* form. The concrete logical operations on $\mathcal{NC}$ will be thus defined with normal forms:

Definition 3.2 *Let $c_1, c_2 \in \mathcal{NC}$,*
$c_1 \tilde{\wedge} c_2 = Res(c_1 \wedge c_2)$ $\quad c_1 \tilde{\vee} c_2 = Res(c_1 \vee c_2)$
$\tilde{\neg} c_1 = Res(\neg c_1)$ $\quad \tilde{\exists} X c_1 = \exists X c_1$ $\quad \tilde{\forall} X c_1 = \forall X c_1$

We denote by $\mathcal{B}$ the set of constrained atoms with constraints in $\mathcal{NC}$, and by $(\mathcal{I}, \subseteq)$ the complete lattice of partial constrained interpretations, not necessarily consistent, formed over $\mathcal{B}$.

4 Depth k analysis for constructive negation

The idea of depth k analysis was first introduced in [20], by Sako e Tamaki. The domain of depth k analysis was then used in order to approximate the ground successful and failure set for normal program in [18] and to approximate the computed answer constraints set for positive programs in [5]. As in [5], we want to approximate an infinite set of computed answer constraints by means of a constraints $depth(k)$ cut, i.e. constraints where

the equalities and inequalities are between variables and cut terms which have a depth no greater than k.
Since the result of an abstract conjunction is a "more general" constraint than the abstraction of the conjunction of the concrete constraints, we need to identify on the abstract domain, sets of constrained atoms which have the same downward closure.

Our concrete domain is the complete lattice $(\mathcal{I}, \subseteq)$, of the previous section. Since our aim is to approximate the computed answer constraints, the fixpoint semantics we choose in the hierarchy [10] is the one which generalizes the S-semantics to normal CLP programs, defined by the $T_P^{\mathcal{H}}$ operator (cf def. 2.2). The version we take here is the one defined with the concrete operations in NC: $\tilde{\wedge}$, $\tilde{\vee}$, $\tilde{\neg}, \tilde{\exists}, \tilde{\forall}$.

4.1 The abstract domain

Terms are cut by replacing each-subterm rooted at depth greater than k with a new fresh variable taken from a set $\tilde{V}$, (disjoint from V). The depth(k) terms represent each term obtained by instantiating the variables of $\tilde{V}$ with terms built over V.
Consider the function $|| : Term \to Term$ such that

$$|t| = \begin{cases} 1 & \text{if } t \text{ is a constant or a variable} \\ max\{|t_1|, \ldots, |t_n|\} + 1 & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

and a given positive integer k. $\alpha_k(t)$ represents the term which can be obtained from the concrete one by substituting a fresh variable (belonging to $\tilde{V}$) to each subterm t' in t, such that $|t| - |t'| = k$.
Consider now the abstract basic constraints

$$\mathcal{ANC} = \left\{ \begin{array}{ll} c \mid & \text{c is a constraint built with} \\ & \text{the logical connectives } \vee, \wedge, \forall \text{ and } \exists \text{ on the} \\ & \textit{ABasicConstraints,} \text{ which is in normal form} \end{array} \right\}$$

and the abstract constraint built with the abstract basic constraints

$$\mathcal{ANC} = \left\{ \begin{array}{ll} c \mid & \text{c is a constraint built with} \\ & \text{the logical connectives } \vee, \wedge, \forall \text{ and } \exists \text{ on the} \\ & \textit{ABasicConstraints,} \text{ which is in normal form} \end{array} \right\}$$

Definition 4.1 *An abstract constrained atom is a couple $c|A$ such that $c \in \mathcal{ANC}$ and c is a $\mathcal{H}-$solvable constraint, A is an atom and $Var(c) \subseteq Var(A)$. With $\mathcal{B}^a$ we intend the set of constrained atoms.*

The abstract domain will be the set of partial interpretation on abstract constrained atoms. A *partial abstract constrained interpretation* for a program, is a couple of set of abstract constrained atoms, $I^a =< {I^a}^+, {I^a}^- >$, not necessary consistent. We consider $\mathcal{I}^a = \{I^a | \; I^a \text{ is a partial interpretation } \}$.

Respect to the case of definite logic programs [5], we need to define a different order on the abstract constraint domain.
One problem is the fact that the result of an abstract *and* operation on the abstract constraint domain will be "more general" than the abstraction of the result of the corresponding concrete operation.
The order on the abstract domain has to be faithful to the idea that the answer constraint that we can compute on the abstract domain is an approximation (where here approximation means "is implied under $\mathcal{H}$") of the abstraction of the one calculated on the concrete domain.
For this purpose, first, we need to formalize the idea of " be more general" between constraints containing equalities between variables and cut terms ($X = f(Y)$ with $Y \in \tilde{V}$) and constraints containing equalities between variables and terms which are not cut ($X = f(H)$ with $H \in V$). $f(Y)$ and $f(H)$ do not represent the same term on the abstract (and consequently on the concrete) domain, since the first represent a term deeper than k while the second term represent himself. The order on the abstract domain, should take into account this difference. For this, $X = f(Y)$ with $Y \in \tilde{V}$, should be considered "less general" than $X = f(H)$ with $H \in V$. Note that with inequalities we will not have this problem since they won't contain cut terms.

Definition 4.2 *Let $c \in \mathcal{ANC}$. $Den : \mathcal{ANC} \rightarrow \mathcal{NC}$.*
$Den(c) = \{c' |$ *c' is obtained by c replacing the variable in $\tilde{V}$ with terms longer than 0* $\}$

We can now define a relation on the abstract constraints domain.

Definition 4.3 *Let $c, c' \in \mathcal{ANC}$.*
$c \tilde{\leq}_a c'$ *if $\forall \tilde{c} \in Den(c)$ there exists $\tilde{c}' \in Den(c')$ such that $\mathcal{H} \models \tilde{c} \rightarrow \tilde{c}'$*

we consider the order $\leq_a$ induced by the preorder $\tilde{\leq}_a$, namely the order obtained considering the classes modulo the equivalence induced by $\tilde{\leq}_a$.
We define the downward closure of a couple of sets wrt the $\leq_a$ order,

Definition 4.4 *Consider a couple of sets of constrained atoms B.*
With $\downarrow^a B$ we mean the downward closure of $< B^+, B^- >$.
$c|A \in \downarrow^a B^+$ *if there exists $c'|A \in B^+$ and $c \leq_a c'$,*
$c|A \in \downarrow^a B^-$ *if there exists $c'|A \in B^-$ and $c \leq_a c'$.*

We define the preorder $\tilde{\leq}$ on the domain $\mathcal{I}^a$.

Definition 4.5 *Consider $I, J \in \mathcal{I}^a$.*
$$I^a \tilde{\leq} J^a \leftrightarrow \begin{array}{l} \forall c|A \in I^{a^+} \; \exists c'|A \in J^{a^+} \text{ such that } c \leq_a c' \text{ and} \\ \forall c|A \in I^{a^-} \; \exists c'|A \in J^{a^-} \text{ such that } c \leq_a c' \end{array}$$

It is worth noting that $I \tilde{\leq} J$ iff $\downarrow^a I \subseteq \downarrow^a J$ and $I \tilde{\equiv} J$ iff $\downarrow^a I = \downarrow^a J$.
$\tilde{\leq}$ is a preorder. We consider the order $\leq$ induced by the preorder $\tilde{\leq}$, namely the order obtained considering the classes modulo the equivalence induced by $\tilde{\leq}$. Then our abstract domain will be the equivalence classes (w.r.t. $\tilde{\equiv}$) of $\mathcal{I}^a$, $(\mathcal{I}^a_{/\tilde{\equiv}}, \leq)$.

4.2 The abstraction function

Let us now define the abstraction function. To this aim we first define the function α_c on constraints. The main idea is to define the α_c on the basic constraint in the following matter, an equality $X = t$ is abstracted with $X = \alpha_k(t)$, while an inequality $X \neq t$ is abstracted with $X \neq t$ if $|t| \leq k$ and with $true$ otherwise.

As before, we write $\Delta(c)$ for the constraint c' in normal form, with Δ we indicate the sequence of quantified variables of c', with c the quantifier free part of c'.

Definition 4.6 *Let* $\alpha_c : \mathcal{NC} \rightarrow \mathcal{ANC}$*:*
$\alpha_c(\Delta(c)) = \Delta, \Delta' \alpha_c(c)$ *where* $\Delta' = \exists d_1, \exists d_2, .., $ *where* $d_i \in (\tilde{V} \cap Var(\alpha_c(c)))$

$$\begin{array}{ll} \alpha_c(X = t) = (X = \alpha_k(t)), & \alpha_c(false) = false, \quad \alpha_c(true) = true, \\ \alpha_c(X \neq t) = (X \neq t) \text{ if } |t| \leq k, & \alpha_c(X \neq t) = (true) \text{ if } |t| > k, \\ \alpha_c(A \wedge B) = \alpha_c(A) \wedge \alpha_c(B), & \alpha_c(A \vee B) = \alpha_c(A) \vee \alpha_c(B) \end{array}$$

The first definition means that all the new variables introduced by the cut terms have to be considered existentially quantified.
In order to define the α operator on sets of constrained atoms, we first give a definition of an auxiliary $\tilde{\alpha}$ function.

Definition 4.7 *Let* $\tilde{\alpha} : \mathcal{I} \rightarrow \mathcal{I}^a$*:* $\tilde{\alpha} =< \tilde{\alpha}^+, \tilde{\alpha}^- >$

$$\begin{array}{ll} \tilde{\alpha}^+(\mathcal{I}) = \{c|A \mid & \textit{such that there exists } c'|A \in I^+ \textit{ and } \alpha_c(c') = c\} \\ \tilde{\alpha}^-(\mathcal{I}) = \{c|A \mid & \textit{such that there exists } c'|A \in I^- \textit{ and } \alpha_c(c') = c\} \end{array}$$

To obtain the abstraction function is sufficient to consider the equivalent classes of $\tilde{\alpha}(I)$.

Definition 4.8 *Let* $\alpha : \mathcal{I} \rightarrow \mathcal{I}^a_{/\doteqdot}$*:*
$\alpha =< \alpha^+, \alpha^- >, \quad \alpha^+(\mathcal{I}) = [\tilde{\alpha}^+(\mathcal{I})]_{/\doteqdot}, \quad \alpha^-(\mathcal{I}) = [\tilde{\alpha}^-(\mathcal{I})]_{/\doteqdot}.$

Consequently the function γ on the sets of abstract constraints is automatically determined as follows:

Definition 4.9 *Let* $\gamma : \mathcal{I}^a_{/\doteqdot} \rightarrow \mathcal{I}$*:*

$$\begin{array}{rll} \gamma(I^a_{/\doteqdot}) = & \cup\{I \mid & \alpha(I) \leq I^a_{/\doteqdot}\} = \\ & \cup\{I \mid & \forall c|A \in \alpha^+(I)\ \exists c'|A \in I^{a^+} \textit{ such that } c \leq_a c' \textit{ and} \\ & & \forall c|A \in \alpha^-(I)\ \exists c'|A \in I^{a^-} \textit{ such that } c \leq_a c'\} = \\ & \cup\{I \mid & \downarrow^a \tilde{\alpha}(I) \subseteq \downarrow^a I^a\} = \\ & \cup\{I \mid & \tilde{\alpha}(I) \subseteq \downarrow^a I^a\} \end{array}$$

$< \alpha, \gamma >$ defines a Galois insertion between the concrete domain $(\mathcal{I}, \subseteq)$ and the abstract domain $(\mathcal{I}^a_{/\doteqdot}, \leq)$.

Lemma 4.1 *α is additive.*

Theorem 4.2 *$< \alpha, \gamma >$ is a Galois insertion of $(\mathcal{I}, \subseteq)$ into $(\mathcal{I}^a_{/\doteqdot}, \leq)$.*

4.3 α_c is a congruence w.r.t. the H-equivalence

As we have already pointed out in section 3, we want to work with $\mathcal{H}$-equivalence classes of constraints and for this purpose, we need to be sure that the function α_c previously defined on $\mathcal{NC}$ is a congruence w.r.t. the $\mathcal{H}$-equivalence. This means that if two constraints $c, c' \in \mathcal{NC}$ are $\mathcal{H}$-equivalent then also $\alpha_c(c)$ and $\alpha_c(c')$ have to be $\mathcal{H}$-equivalent.
Let us now introduce some preliminary notions

Definition 4.10 ([16]) *A solved form equation set E_1 is isomorphic to E_2 if there is a subset $\{X_1 = Y_1, \ldots, X_k = Y_k\}$ of E_1 where the Y_i's are distinct variables such that $E_2 = E_1\{X_1 \leftarrow Y_1, \ldots, X_k \leftarrow Y_k, Y_1 \leftarrow X_1, \ldots, Y_k \leftarrow X_k\}$.*

Lemma 4.3 ([16]) *Let E_1 be a set of equation in solved form. Then E_2 is an equivalent set of equations in solved form iff E_1 is isomorphic to E_2.*

In order to understand which are the constraints that are $\mathcal{H}$-equivalent is also useful to state the following result

Lemma 4.4 *Consider an arbitrary quantified X and a universally quantified term t. It does not exist arbitrary quantified $t_1, \ldots, t_n$, where $t_i \neq t$ such that $X \neq t$ is $\mathcal{H}$-equivalent to $\wedge_i X \neq t_i$.*

This is a consequence of the fact that we consider the models of the theory CET without the DCA axiom.

We can now state the theorem that assure that α is a congruence.

Theorem 4.5 *Let $c, c' \in \mathcal{NC}$. If $\mathcal{H} \models c \leftrightarrow c'$ then $\mathcal{H} \models \alpha_c(c) \leftrightarrow \alpha_c(c')$.*

4.4 The abstract fixpoint operator

We now define the abstract operations that will take place instead of the concrete ones in the definition of fixpoint abstract operator. We will show that the abstract operations are a correct approximation w.r.t. the concrete operations on constraints.
First we define a new operator $\mathcal{M}$.
This operator is necessary since the logical *and* operation between two constraint is not in general an abstract constraint (its depth can be greater than k).
Moreover, in order to assure correctness, the $\mathcal{M}$ operator must also substitute with true all the inequality $X \neq t[Y \backslash t']$ where t' represent a term longer than k, namely $Var(t') \cap \tilde{V} \neq \emptyset$. Such inequalities represent on the concrete domain, inequalities longer than k, which would have been substituted with *true* by α.

Definition 4.11 *Let $\mathcal{M} : \mathcal{NC} \rightarrow \mathcal{ANC}$*

$$\mathcal{M}(\Delta(c)) = \Delta, \Delta'\mathcal{M}(c) \textit{ where } \Delta' = \exists d_1, \exists d_2, .., \textit{ where } d_i \in (\tilde{V} \cap Var(\alpha_c(c))).$$

$\mathcal{M}(X = t) = (X = \alpha_k(t))$

$\mathcal{M}(X \neq t) = (X \neq t)$ *if* $|t| \leq k$ *and* $Var(t) \cap \tilde{V} = \emptyset$

$\mathcal{M}(X \neq t) = (true)$ *if* $|t| > k$ *or* $Var(t) \cap \tilde{V} \neq \emptyset$

$\mathcal{M}(A \wedge B) = \alpha_c(A) \wedge \alpha_c(B)$, $\mathcal{M}(A \vee B) = \alpha_c(A) \vee \alpha_c(B)$

It is immediate to note that the operator $\mathcal{M}$ is similar to the α operator, the only difference is that is that $\mathcal{M}$ substitute with *true* all the inequalities between variables and cut terms.
Since $\mathcal{ANC}$ is a subset of $\mathcal{NC}$ the *Res* form is defined also on the abstract constraints domain.

Definition 4.12 *Let* $c_1, c_2 \in \mathcal{ANC}$
$c_1 \overset{a}{\tilde{\wedge}} c_2 = \mathcal{M}(Res(c_1 \wedge c_2))$, $c_1 \overset{a}{\tilde{\vee}} c_2 = Res(c_1 \vee c_2)$,
$\tilde{\neg}^a c_1 = \mathcal{M}(Res(\neg c_1))$, $\tilde{\exists}^a X\ c_1 = \exists X c_1$, $\tilde{\forall}^a X\ c_1 = \forall X c_1$,

The abstract operations $\overset{a}{\tilde{\wedge}}$, $\overset{a}{\tilde{\vee}}$, $\tilde{\exists}^a$, $\tilde{\forall}^a$ are corrects w.r.t. the concrete ones. Now that we have defined the abstract "and" operator we can better illustrate with an example its behavior.

Example 4.1 *Consider the concrete constraints* $c_1 = \forall K((Y = a \wedge U \neq f(f(K))) \vee Z = a)$ *and* $c_2 = (U = f(f(a)))$ *and* $k = 1$.
Now $\alpha_c(c_1) = (Y = a \vee Z = a)$ *and* $\alpha_c(c_2) = \exists V\ U = f(V)$.
$\alpha_c(c_1) \overset{a}{\tilde{\wedge}} \alpha_c(c_2) = \exists V((Y = a \wedge U = f(V)) \vee (Z = a \wedge U = f(V)))$.
While $\alpha_c(Res(c_1 \wedge c_2)) = \exists V(Z = a \wedge U = f(V))$.
As already pointed out, the abstract "and" gives as result a more general constraint than the abstraction of the one calculated by the concrete "and".

It should be now more clear why we have chosen such an order on the abstract constraint domain and on the subset of the constrained atoms domain. In order to show that the abstract operations are corrects we show a stronger property.

Theorem 4.6 *Let* $c_1, c_2 \in \mathcal{NC}$.
$\alpha_c(c_1) \overset{a}{\tilde{\wedge}} \alpha_c(c_2) \geq_a \alpha_c(c_1 \wedge c_2)$, $\alpha_c(c_1) \overset{a}{\tilde{\vee}} \alpha_c(c_2) = \alpha_c(c_1 \vee c_2)$,
$\tilde{\exists}^a x\ \alpha_c(c_1) = \alpha_c(\tilde{\exists} x\ c_1)$, $\tilde{\forall}^a x\ \alpha_c(c_1) = \alpha_c(\tilde{\forall} x\ c_1)$.

Unfortunately, the correctness property does not hold for our abstract "not" considering general constraints.

Example 4.2 *Consider* $c_1 = X \neq f(f(a))$ *and* $k = 1$.
$\alpha_c(\neg(c_1)) = \exists Y\ X = f(Y)$ *which does not implies* $\neg^a(\alpha_c(c_1))$.

It is worth noting that a more complex version of the "or" operator can be defined in order to obtain the correctness for general constraints. But since the not operator is used by the abstract fixpoint operator, only on "simpler " constraints (conjunctions of equalities between variables and terms, namely the program constraints), this is not worth it.

Lemma 4.7 *Consider $c_1 \in \mathcal{NC}$ such that $c_1 = \bigwedge_i X_i = t_i$. $\tilde{\neg}^a \alpha_c(c_1) = \alpha_c(\tilde{\neg}(c_1))$.*

Once that we have defined the abstract constraints domain and the abstract operations, we can define the abstract fixpoint operator. Since in the abstract domain we identify I and J if $\downarrow^a I = \downarrow^a J$, the abstract operator applied to an equivalence class I is the equivalent class of the abstract operator (which uses abstract operations and the abstract version of the program) working on $\mathcal{B}^a$, applied to $\downarrow^a I^a$.

Definition 4.13 *The abstract fixpoint operator: $\mathcal{I}^a_{/\tilde{\equiv}} \to \mathcal{I}^a_{/\tilde{\equiv}}$ is defined as follows, $T^{\mathcal{B}^a}_{S,P}([I^a]_{/\tilde{\equiv}}) = [T^{\mathcal{B}^a}_{S,\alpha(P)}(\downarrow^a I^a)]_{/\tilde{\equiv}}$, where the operations are $\tilde{\exists}^a$, $\tilde{\forall}^a$, $\tilde{\neg}^a$ on $\mathcal{ANC}$ and $\overset{a}{\tilde{\vee}}$, $\overset{a}{\tilde{\wedge}}$ on $\mathcal{ANC} \times \mathcal{ANC}$.*

By definition $T^{\mathcal{B}^a}_{S,P}$ is a congruence respect to the equivalence classes of the abstract domain. Note also that $T^{\mathcal{B}^a}_{S,P}$ is monotone on the $(\mathcal{I}^a_{/\tilde{\equiv}}, \leq)$. This comes from the fact that $[I]_{/\tilde{\equiv}} \leq [J]_{/\tilde{\equiv}}$ implies $\downarrow^a I \subseteq \downarrow^a J$.

Lemma 4.8 *$T^{\mathcal{B}^a}_{S,P}$ is monotone on the $(\mathcal{I}^a_{/\tilde{\equiv}}, \leq)$.*

The proof that the abstract operator is correct w.r.t. the concrete one, is based on the correctness of the abstract operations on the abstract constraints domain.

Theorem 4.9 *$\alpha(T^{\mathcal{B}}_{S,P}(\gamma([I^a]_{/\tilde{\equiv}}))) \leq T^{\mathcal{B}^a}_{S,P}([I^a]_{/\tilde{\equiv}})$.*

4.5 The abstract operator is optimal

Consider a k greater than the maxima depth of the terms involved in the constraints of the clauses in the program P. In this case the abstract operator is also optimal. Before stating the main theorem, we need an intermediate result. The next lemma state (constructively in the proof) the existence of a $c_1', c_2' \in \mathcal{NC}$ such that

Lemma 4.10 *Consider $c_1, c_2 \in \mathcal{ANC}$ then $\exists c_1', c_2' \in \mathcal{NC}$, such that $c_1 \overset{a}{\tilde{\wedge}} c_2 = \alpha_c(c_1' \tilde{\wedge} c_2')$, and $c_1 \overset{a}{\tilde{\vee}} c_2 = \alpha_c(c_1' \tilde{\vee} c_2')$.*

Then

Theorem 4.11 $[T^{\mathcal{B}^a}_{S,P}([I^a]_{/\tilde{\equiv}})]_{/\tilde{\equiv}} \leq \alpha(T^{\mathcal{B}^a}_{S,P}(\gamma([I^a]_{/\tilde{\equiv}})))$

Let us finally discuss termination properties of the dataflow analyses presented in this section.
First note that the set of not equivalent (w.r.t. $\mathcal{H}$) set of constraints belonging to $\mathcal{ANC}$ is finite.

Lemma 4.12 *Assume that our alphabet has a finite number of function and predicate symbols. Our depth-k abstraction is ascending chain finite.*

4.6 An example

We now show how the depth-k analysis works on an example. The program that we consider is the same as the one in [18]. Consider the normal logic program P

$$
\begin{array}{ll}
P: & p(X) : -q(X,Y), \neg r(Y). \\
 & q(X,Y) : -X = Y. \\
 & r(X) : -X = f(a). \\
 & r(X) : -X = f(f(T)), r(X).
\end{array}
$$

Consider the concrete fixpoint semantics of the program P, which is reached after the first two steps :

$$
\begin{array}{cc}
T^{\mathcal{B}+}_{S,P} & T^{\mathcal{B}-}_{S,P} \\
X = Y|q(X,Y). & X \neq Y|q(X,Y). \\
X = f(a)|r(X). & \forall T(X \neq f(a) \wedge X \neq f(f(T)))|r(X). \\
\forall T(X \neq f(a) \wedge X \neq f(f(T)))|p(X). & X = f(a)|p(X).
\end{array}
$$

Consider now a depth-2 analysis with $Z \in \tilde{V}$. With $\downarrow^a c|A$ ($c \in \mathcal{ANC}$), we indicate a set of constrained atoms $\{c'|A \mid c' \in \mathcal{ANC}$ and $c' \leq_a c\}$ and $(\downarrow^a c \wedge \downarrow^a c')|A$ indicate the set $\{ (c_1 \wedge c_2)|A \mid c_1 \leq_a c$ and $c_2 \leq_a c'\}$. With $[I]$ ($I \in \mathcal{I}^a$), we intend the equivalence class of the set I w.r.t. $\tilde{\equiv}$.

$$
\begin{array}{cc}
T^{\mathcal{B}^{a+}}_{S,P} & T^{\mathcal{B}^{a-}}_{S,P} \\
\left[\begin{array}{c} X = Y|q(X,Y). \\ X = f(a)|r(X). \\ \downarrow^a X \neq f(a)|p(X). \end{array}\right] &
\left[\begin{array}{c} X \neq Y|q(X,Y). \\ X \neq f(a)|r(X). \\ \downarrow^a X = f(a)|p(X) = \{\ X = f(a)|p(X)\ \} \end{array}\right]
\end{array}
$$

Note that we have $(T^{\mathcal{B}+}_{S,P}, T^{\mathcal{B}-}_{S,P}) \subseteq \gamma(T^{\mathcal{B}^{a+}}_{S,P}, T^{\mathcal{B}^{a-}}_{S,P})$.

On this program Marriott and Sondergaard showed that their depth-2 analysis computed the approximation $\{p(a), q(x,x), r(f(a))\}$ for the success set, and $\{p(f(a)), q(a, f(a)), q(f(a), a), q(f(x), f(y)), r(a)\}$ for the failure set. This abstract interpretation is inconsistent as $q(f(a), f(a))$ is in both sets. It is not the case with our approach in this example (of course it could be the case with other examples), as by taking a non-ground reference semantics for the analysis we obtain much more precise results.

On the other hand, we do not obtain $X = a$ as a computed answer for $p(X)$. The reason is that the constraint $X = a$ is indeed not computed by constructive negation (and a fortiori not by SLDNF-resolution), and $p(a)$ is also not true in Kunen's semantics under Clark's equational theory (it is true however with the addition of the domain closure axiom DCA [17] to CET assuming a finite alphabet, in which case the computed answer in our case is DCA-equivalent to $X = a$). As already pointed out, the analysis of Marriott and Sondergaard is based on Fitting's semantics for which $p(a)$ is true in this example, but in general Fitting's semantics is non effective [11] and is thus already an approximation of any computable semantics. This is the second reason for our gain of precision.

5 Conclusion

Starting from the hierarchy of semantics defined [10], our aim was to show that well known analysis for logic programs could be extended to normal CLP programs. Based on the framework of abstract interpretation [7, 8], we have presented a depth-k analysis which is able to approximate the answer set of normal CLP programs. The analysis has been proven correct and also optimal w.r.t. constructive negation. Starting from a generalized S-semantics for normal CLP programs [9], we have defined a normal form for Herbrand constraints, which allowed us to define a function of abstraction which is a congruence for the equivalence in CET. This method is somehow general. Based on the semantics of the hierarchy in [10] and on the normal form of constraint presented in this paper, other well known analyses for logic programs can be extended to normal CLP programs. For example, starting from a suitable version of Clark's semantics presented in the hierarchy, a groundness analysis can be defined which can be proven correct and also optimal w.r.t. constructive negation. We have chosen to present, here, a $depk - k$ analysis, because it was also interesting to show how from inequalities can be derived useful information for the analysis.

References

[1] A. Bossi, M. Gabbrielli, G. Levi, M. Martelli, "The s-semantics approach: theory and applications", Journal of Logic Programming, 19-20, pp.149-197, 1994.

[2] D. Chan, "Constructive negation based on the completed database", in: R.A. Kowalski and K.A. Bowen (eds), Proc. of the fifth International Conference on Logic Programming, MIT Press, Cambridge, MA, pp.11-125, 1988.

[3] B.Le Charlier, P.Van Hentenryck, "Experimental evaluation of a generic abstract interpretation algorithm for Prolog", ACM Trans. Programming Language Systems, 16(1):35-101, 1994.

[4] A.Colmerauer, "Equations and inequations on finite and infinite trees" Proc.International Conference on 5th Generation Computer Systems,pp.85-99,1984.

[5] M.Comini, G.Levi, C.Meo, G.Vitiello, "Abstract diagnosis", submitted for publication, Preliminary version in International Workshop on Tools and Environments for (Constraint) Logic Programming, ILPS'97 Postconference Workshop,1997.

[6] P. Cousot, R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction of approximation of fix-

points ", In Proc. 4th ACM Symposium on Principles of Programming Languages, pp.238-252, 1977.

[7] P. Cousot, R. Cousot, "Systematic design of program analysis frameworks", In Proc. 6th Annual Symposium on Principles of Programming Languages, pp.269-282, 1979.

[8] P. Cousot, R. Cousot, "Abstract interpretation and application to logic programs", Journal of Logic Programming, 13(2 and 3), pp.103-179, 1992.

[9] F. Fages, "Constructive negation by pruning", Journal of Logic Programming, 32(2):85-118, 1997.

[10] F. Fages, R.Gori "A hierarchy of semantics for normal constraint logic programs", In Proc. ALP, Springer-Verlag LNCS 1139, 1996.

[11] M. Fitting, "A Kripke/Kleene semantics for logic programs", Journal of Logic Programming, 2(4), pp.295-312, 1985.

[12] M.Garcia de la Banda, M. Hermenegildo, M. Bruynooghe, V. Dumortier, G. Janssens, W. Simoens, "Global analysis of constraint logic programs", ACM Transactions on Programming Languages and Systems, 18(5):564-614, 1996.

[13] R. Giacobazzi, ""Optimal" collecting semantics for analysis in a hierarchy of logic program semantics", Proc of 13th STACS'96, C. Puech and R Reischuk Ed., LNCS 1046, Springer Verlag, pp.503-514. 1996.

[14] K. Kunen, "Negation in logic programming", Journal of Logic Programming, 4(3), pp.289-308, 1987.

[15] K. Kunen, "Signed data dependencies in logic programming", Journal of Logic Programming, 7(3), pp.231-245, 1989.

[16] L.Lassez, J.Maher, K.Marriot, "Unification revisited", In Foundations of deductive databases and logic programming, Morgan-Kaufman, 1988.

[17] M.Maher, "Complete axiomatizations of the algebra of finite, rational and infinite trees", Third Symp. on Logic in Computer Science, pp.348-357, 1988. .

[18] K.Marriot, H.Sondergaard, "Bottom-up dataflow analysis of normal logic programs ", Journal of Logic Programming, 13:181-204, 1992.

[19] K.Marriot, H.Sondergaard, P.Stuckey, R¿Yap, 'Optimizing compilation for CLP($\mathcal{R}$)", Proc. 17th Annual Compouter Science Conference, 1994.

[20] T.Sato, H.Tamaki, "Enumeration of success Patterns in Logic Programs, Theoretical Computer Science, 34:227-240,1984.

[21] P. Stuckey, “Negation and constraint logic programming”, Information and Computation, 118(1), pp.12-33, 1995.

A Appendix: Proofs of theorems

Theorem A.1 (3.1) *For any input constraint $c = \Delta(\vee_i A_i)$, algorithm 3.1 terminates and computes a* false-simplified *form of c.*

Proof A.1 *The algorithm terminates since the A_i's in $c = \Delta(\vee_i A_i)$ are a finite number.*
We then show that it computes a false-simplified *form of c.*
$c = \Delta(\vee_i A_i)$ and I is the subset of the A_i's used in the method above.
Applying the method, the constraint that we obtain is, by construction (step 3), equivalent to the constraint $c = \Delta(\vee_i A_i)$.
We show that this resulting constraint is false-simplified.
Let us suppose that this was not the case. Then there exist a $\overline{I} \subseteq I$ such that $\mathcal{H} \models \Delta(\vee_{i\in\overline{I}} A_j) \leftrightarrow \Delta(\vee_i A_i)$ and for all K subset of $\overline{I}$, $\vee_{i\notin\overline{I}} A_i$ is not $\mathcal{H}$-equivalent to $\vee_{k\in K} A_k$. Then there exists at least an $A_{\bar{i}} \in I\backslash\overline{I}$. such that

- *$A_{\bar{i}}$ is entered in I in the first phase.*
  In this case $\mathcal{H} \models \exists\Delta(A_{\bar{i}})$. $\mathcal{H} \not\models \Delta(A_{\bar{i}}) \leftrightarrow false$. In this case or $\mathcal{H} \not\models \Delta(\vee_{i\in\overline{I}} A_j) \leftrightarrow \Delta(\vee_i A_i)$ or $A_{\bar{i}}$ is redundant and then there exists K subset of $\overline{I}$ such that $\vee_{i\notin\overline{I}} A_i$ is $\mathcal{H}$-equivalent to $\vee_{k\in K} A_k$.

- *$A_{\bar{i}}$ is entered in I in the second phase, with other A_j $j \in S$.*
  In this case $\mathcal{H} \models \exists\Delta(\vee_{s\in S} A_s)$ and $\mathcal{H} \not\models \exists\Delta(\vee_i \in (S\backslash j) A_i)$.
  $\mathcal{H} \not\models \Delta(\vee_{s\in S} A_s) \leftrightarrow false$. As before or $\mathcal{H} \not\models \Delta(\vee_{i\in\overline{I}} A_j) \leftrightarrow \Delta(\vee_i A_i)$ or $\Delta(\vee_{s\in S} A_s)$ is redundant and then there exists K subset of $\overline{I}$ such that $\vee_{i\notin\overline{I}} A_i$ is $\mathcal{H}$-equivalent to $\vee_{k\in K} A_k$.

- *$A_{\bar{i}}$ is entered in I in the third phase.*
  In this case S would not be minimal.

Proposition A.1 (3.1)

$$\mathcal{H} \models \phi \leftrightarrow Res(\phi)$$

Proof A.2 *The first three steps correspond to the first three step of the Maher's canonical form for the Herbrand Universe [17] and are been proved to preserve the equivalence.*
The fourth step correspond to the steps performed in the simplification algorithm in [4] to simplify inequations and are been proved to preserve the equivalence.
By definition, the fifth step preserves the $\mathcal{H}$-equivalence.

Lemma A.2 (4.2) *α is additive.*

Proof A.3 *First note that $[I^a]_{/\doteqdot} \bigvee_{\equiv} [J^a]_{/\doteqdot} = [I^a \tilde{\vee} J^a]_{/\doteqdot}$ where $\tilde{\vee}$ is the l.u.b on the $(I^a, \tilde{\leq})$.*
We need to show that

$[\tilde{\alpha}(\cup_i S_i)]_{/\tilde{\equiv}} = \bigvee_{\equiv}[\tilde{\alpha}(S_i)]_{/\tilde{\equiv}} = [\tilde{\vee}_i \tilde{\alpha}(S_i)]_{/\tilde{\equiv}}$.
To this aim we show that $\tilde{\alpha}(\cup_i S_i) \tilde{\equiv} \tilde{\vee}_i \tilde{\alpha}(S_i)$.
We want to show that $\downarrow^a \tilde{\alpha}(\cup_i S_i) = \downarrow^a \tilde{\alpha}(\cup_i S_i)$
First we show that $\downarrow^a \tilde{\alpha}(\cup_i S_i) \subseteq \downarrow^a (\cup_i \tilde{\alpha}(S_i))$
Consider $c \in \downarrow^a \tilde{\alpha}(\cup_i S_i)$*, then there exists a* c', $c \leq_a c'$ *such that* $c' \in \tilde{\alpha}(\cup_i S_i)$*, by definition of* $\tilde{\alpha}$*, which apply* α_c *to each element of* $\cup_i S_i$*, we have that* $\tilde{\alpha}(\cup_i S_i) = \cup_i \tilde{\alpha}(S_i)$ *then* $c' \in \cup_i \tilde{\alpha}(S_i)$. c*, which is* $c \leq c'$*, belongs to* $\downarrow^a (\cup_i \tilde{\alpha}(S_i))$.
Also $\downarrow^a (\cup_i \tilde{\alpha}(S_i)) \subseteq \downarrow^a \tilde{\alpha}(\cup_i S_i)$.
Consider $c \in \downarrow^a (\cup_i \tilde{\alpha}(S_i))$ *then there exists a* c', $c \leq_a c'$ *such that* $c' \in \cup_i \tilde{\alpha}(S_i)$ *then for some* j, $c' \in \tilde{\alpha}(S_j)$*, by definition of* $\tilde{\alpha}$*, which apply* α_c *to each element of* S_j, $c' \in \tilde{\alpha}(\cup_i S_i)$. c*, which is* $c \leq c'$*, belongs to* $\downarrow^a \tilde{\alpha}(\cup_i S_i)$.
Then $\downarrow^a \tilde{\alpha}(\cup_i S_i) = \downarrow^a (\cup_i \tilde{\alpha}(S_i))$.
But $\downarrow^a (\cup_i \tilde{\alpha}(S_i)) = \cup_i \downarrow^a \tilde{\alpha}(S_i) = \tilde{\vee}_i \tilde{\alpha}(S_i)$.
This completes the proof.

Theorem A.3 (4.5) *Let* $c, c' \in \mathcal{NC}$.
If $\mathcal{H} \models c \leftrightarrow c'$ *then* $\mathcal{H} \models \alpha_c(c) \leftrightarrow \alpha_c(c')$.

Proof A.4 *In the following by equivalent we mean* $\mathcal{H}$*-equivalent. Let us first note that*

a α_c *does not changes the structure of the connectives and quantifiers.*

b *if* $c \leftrightarrow true$ *then* $\alpha_c(c) \leftrightarrow \alpha_c(true) = true$.
If

- c *has the form* $\exists X\ Y \neq t[X]$ *for an arbitrary (and arbitrary quantified)* Y *and a generic* t *then, if* $|t| > k$ *then* $\alpha_c(c) = true$. *If* $|t| \leq k$ $\alpha_c(c) = c$ *and* $c \leftrightarrow \alpha_c(true) = true$.
- c *has the form* $Y = t \vee Y \neq t$ *for an arbitrary (and arbitrary quantified)* Y *and a generic* t *then if* $|t| > k$ $\alpha_c(c) = \alpha_c(Y = t) \vee \alpha_c(Y \neq t) = \alpha_c(Y = t) \vee true = true$. *In case* $|t| \leq k$ $\alpha_c(c) = c$ *and* $c \leftrightarrow \alpha_c(true) = true$. *Since* α_c *does not changes the structure of the connectives for all* $c \leftrightarrow true$ *then* $\alpha_c(c) \leftrightarrow \alpha_c(true)$.

c *if* $c \leftrightarrow false$ *then* $\alpha_c(c) \leftrightarrow \alpha_c(false) = false$.
This is true by definition of the Res form, recalling that c *is in a false simplified form.*

Let us now show that if $c \leftrightarrow c'$ *then* $\alpha_c(c) \leftrightarrow \alpha_c(c')$ *where* $c \not\leftrightarrow true$ *and* $c \not\leftrightarrow false$.
We show it by double structural induction on c *and* c'.
If c *and* c' *are basic quantified constraints (which are not equivalent to true or false) this is trivial.*
Suppose now that only c' *is a basic quantified constraint:*

- *$c' = (X = t)$ for an arbitrary (and arbitrary quantified) X and a generic (and arbitrary quantified) t. Since the equalities in the constraints are in solved form, by lemma 4.3, c is isomorphic to c', this means that c have the following syntactic form up to variables renaming,*
  $c = c' \bigwedge T_i \bigvee F_j$ where $T_i \leftrightarrow true$ or $T_i \equiv c'$ and $F_i \leftrightarrow false$ or $F_i \equiv c'$.
  By the observations a,b and c we still have that $\alpha_c(c) \leftrightarrow \alpha_c(c')$.

- *$c' = (X \neq t)$ for an arbitrary (and arbitrary quantified) X and a generic (not existentially quantified otherwise we are in the case a) t. By lemma 4.4, c has to have the following syntactic form up to variables renaming, $c = c' \bigwedge T_1 \bigvee F_j$ where $T_i \leftrightarrow true$ or $T_i \equiv c'$ and $F_i \leftrightarrow false$ or $F_i \equiv c'$.*
  By the observations a,b and c we still have that $\alpha_c(c) \leftrightarrow \alpha_c(c')$.

  We show it now for c and c' not basic constraints:

  - *if $c = \exists X\ d$ since $c \not\leftrightarrow true$ and $c \not\leftrightarrow false$ then also c' has to be equal $\exists y\ d'$ and $d \leftrightarrow d'$. By induction hypothesis $\alpha_c(d) \leftrightarrow \alpha_c(d')$. This implies $\alpha_c(\exists xd) = \exists x\alpha_c(d) \leftrightarrow \exists y\alpha_c(d') = \alpha_c(\exists yd')$.*

  - *if $c = \forall X\ d$ the case is similar to the one above.*
  - *if $c = A \wedge B$ then $c' = A' \wedge B'$ such that $A' \leftrightarrow A$ and $B' \leftrightarrow B$. By induction hypothesis $\alpha_c(A') \leftrightarrow \alpha_c(A)$ and $\alpha_c(B') \leftrightarrow \alpha_c(B)$. Then $\alpha_c(A' \wedge B') = \alpha_c(A') \wedge \alpha_c(B') \leftrightarrow \alpha_c(A) \wedge \alpha_c(B) = \alpha_c(A \wedge B)$.*
  - *if $c = A \vee B$ then the case is similar to the one above.*

Theorem A.4 (4.6) *Let $c_1, c_2 \in \mathcal{NC}$*

- $\alpha_c(c_1) \overset{a}{\tilde{\wedge}} \alpha_c(c_2) \geq_a \alpha_c(c_1 \wedge c_2)$,

- $\alpha_c(c_1) \overset{a}{\tilde{\vee}} \alpha_c(c_2) = \alpha_c(c_1 \vee c_2)$,

- $\tilde{\exists}^a x\ \alpha_c(c_1) = \alpha_c(\tilde{\exists} x\ c_1)$,

- $\tilde{\forall}^a x\ \alpha_c(c_1) = \alpha_c(\tilde{\forall} x\ c_1)$.

Proof A.5 *By sake of simplicity in the following proof α_c will be called simply α*

- *$\alpha(c_1) \overset{a}{\tilde{\wedge}} \alpha(c_2) \geq_a c_1 \wedge c_2$.*
  We need to show that $\mathcal{M}(Res(\alpha(c_1) \wedge \alpha(c_2))) \geq_a \alpha(Res(c_1 \wedge c_2))$.
  We will first show that $\mathcal{M}(Res(\alpha(c_1) \wedge \alpha(c_2))) \geq_a \alpha(Res(c_1 \wedge c_2))$ holds for the first three phases of Res and then that it holds also for the fourth phase.
  As we already pointed out the function α does not change the structure

of the connectives and the quantifiers of a formula c.
α, in fact, leaves unchanged the quantifiers in front of a formula, cuts the equality, substitute with true *the inequality longer than k and makes explicit the existential quantifications of the new variables introduced by the cut but does not manipulate the connectives.*
The same result holds for the $\mathcal{M}$ operator.
This means that we can always think $\alpha(c)$ (or $\mathcal{M}(c)$) as having the same structure *than c, eventually with equality where the right part is cut at k in place of equalities "longer than k",and the logical constant* true *in place of inequalities"longer than k"(or containing variables in $\tilde{V}$).*
For similar reasons, we can think of the normal form of $(\alpha(c_1) \wedge \alpha(c_2))$ as having the same syntactic "structure" of $c_1 \wedge c_2$, namely if the prenex disjunctive-conjunctive form of $(\alpha(c_1) \wedge \alpha(c_2))$ is $\Delta(A_1 \vee \ldots \vee A_n)$ (where A_i is a conjunction of basic constraints), the prenex disjunctive-conjunctive form of $c_1 \wedge c_2$ is $\Delta(A'_1 \vee \ldots \vee A'_n)$ (where A_i is a conjunction of basic constraints), and the relation between the elements of the conjunction of each A_i w.r.t. the corresponding A'_i is as explained above.
This depends on the fact that also the prenex disjunctive conjunctive form is a syntactic manipulation which renames the variables and distributes the connectives.
Consider now the first tree phases of Res, they operate on each conjunction separately.
Calling, as before, the disjunctive form of $\alpha(c_1) \wedge \alpha(c_2)$ $\Delta(A_1 \vee \ldots \vee A_n)$ and the disjunctive form of $c_1 \wedge c_2$ $\Delta(A'_1 \vee \ldots \vee A'_n)$ we have
$\mathcal{M}(Res^{1-3}(\alpha(c_1) \wedge \alpha(c_2))) = \mathcal{M}(Res^{1-3}(\Delta(A_1 \wedge \ldots \wedge A_n))) =$ (by the observation on $\mathcal{M}$ and Res^{1-3} above) $\Delta(\mathcal{M}(Res^{1-3}(A_1)) \vee \ldots \vee \mathcal{M}(Res^{1-3}(A_n)))$.
In similar way
$\alpha(Res^{1-3}(\Delta(A'_1 \vee \ldots \vee A'_n))) = \Delta(\alpha(Res^{1-3}(A'_1)) \vee \ldots \vee \alpha(Res^{1-3}(A'_n)))$.
We first want to show that for each basic constraint in $\Delta(\mathcal{M}(Res^{1-3}(A_1)) \vee \ldots \vee \mathcal{M}(Res^{1-3}(A_n)))$, the corresponding basic constraint in $\Delta(\alpha(Res^{1-3}(A'_1)) \vee \ldots \vee \alpha(Res^{1-3}(A'_n)))$ implies it, under $\mathcal{H}$.
This would show that $\mathcal{M}(Res^{1-3}(\alpha(c_1) \wedge \alpha(c_2))) \geq_a \alpha(Res^{1-3}(c_1 \wedge c_2))$.
Suppose that this was not the case, namely there exists a basic constraint $\tilde{c}$ in the conjunction $\mathcal{M}(Res^{1-3}(A_i))$ which is not implied by the corresponding basic constraint in $\alpha(Res^{1-3}(A'_i))$.
Such basic constraint $\tilde{c}$ can be

- *an equality $X = t$ which belongs also to $\alpha(c_1)$ (or $\alpha(c_2)$) then there exists a t', $\alpha_k(t') = t$ such that $X = t' \in c_1$. Since $X = t' \in c_1$ $X = t'' \in Res^{1-3}(c_1 \wedge c_2)$ where $t'' \leq t'$. Then $X = t \in \alpha(Res^{1-3}(c_1 \wedge c_2))$.*
- *an equality $X = t$ such that there exists a t', $\alpha_k(t') = t$ and $X = t' \in Res^{1-3}(\alpha(c_1) \wedge \alpha(c_2))$, then there exists a t'', $t'' \leq t'$ such*

that $X = t'' \in Res^{1-3}(c_1 \wedge c_2)$. *Then* $X = t \in \alpha(Res^{1-3}(c_1 \wedge c_2))$.

  - *an inequality* $X \neq t$ *which belongs also to* $\alpha(c_1)$ *(or* $\alpha(c_2)$*) then, by definition of* α, $|t| \leq k$, *and* $X \neq t \in c_1$. $X \neq t \in Res^{1-3}(c_1 \wedge c_2)$. *Then* $X \neq t \in \alpha(Res^{1-3}(c_1 \wedge c_2))$.
  - *an inequality* $X \neq t$ *which belongs to* $Res(\alpha(c_1) \wedge \alpha(c_2))$. *Note that if* $X \in \tilde{V}$ *then also* $t = U \in \tilde{V}$. *Since* $\mathcal{M}$ *did not substitute the inequality with true,* $X \notin \tilde{V}$ *and* $|t| \leq k$, $X \neq t \in Res^{1-3}(c_1 \wedge c_2)$. *Then* $X \neq t \in \alpha(Res^{1-3}(c_1 \wedge c_2))$.

For the first three phases of Res we have showed that
$\mathcal{M}(Res^{1-3}(\alpha(c_1) \wedge \alpha(c_2))) \geq_a \alpha(Res^{1-3}(c_1 \wedge c_2))$.

It is easy to note that for the fourth phase we have the following relation $\mathcal{M}(Res(\alpha(c_1) \wedge \alpha(c_2))) \geq_a \alpha(Res(c_1 \wedge c_2))$.
This comes from the observation that the conjunction of base concrete constraints in $\alpha(Res(c_1 \wedge c_2))$ *could be inconsistent for* $\mathcal{H}$ *while this is not true for the corresponding conjunction in* $\mathcal{M}(Res(\alpha(c_1) \wedge \alpha(c_2)))$, *while if a conjunction of base abstract constraints in* $\mathcal{M}(Res(\alpha(c_1) \wedge \alpha(c_2)))$ *is inconsistent for* $\mathcal{H}$, *then the corresponding conjunction in* $\alpha(Res(c_1 \wedge c_2))$ *is also inconsistent. Then* $\mathcal{M}(Res(\alpha(c_1) \wedge \alpha(c_2))) \geq_a \alpha(Res(c_1 \wedge c_2))$.

- $\alpha(c_1) \overset{a}{\tilde{\vee}} \alpha(c_2) = \alpha(c_1 \tilde{\vee} c_2)$.
  First, let us show how can simplify $\alpha(c_1 \tilde{\vee} c_2)$ *which is by definition equal to* $\alpha(Res(c_1 \vee c_2'))$. *We will show how we can simplify* $\alpha(c_1) \overset{a}{\tilde{\vee}} \alpha(c_2)$ *in a similar way.*
  Let $c_1 = \Delta_{c_1}(c_1)$ *and* $c_2' = \Delta_{c_2'}(c_2')$. *The procedure Res on* c_1 *and* c_2' *performs just the first step since* $c_1, c_2' \in \mathcal{NC}$ *and they are connected by an or.*
  Then $\alpha(Res(c_1 \vee c_2')) = \alpha(\Delta_{c_1} \cup \Delta_{c_2'}(c_1 \vee c_2'))$.
  Now applying α *we obtain:* $\Delta_{c_1} \cup \Delta_{c_2'} \cup \Delta_3(\alpha(c_1) \vee \alpha(c_2'))$ *where* $\Delta_3 = \exists d_1, \exists d_2, ..,$ *where* $d_i \in (\tilde{V} \cap Var(\alpha(c_1)) \cup (\tilde{V} \cap Var(\alpha(c_2)))$.
  On the other end, we simplify $\alpha(c_1) \overset{a}{\tilde{\vee}} \alpha(c_2)$ *which is by definition equal to* $Res(\alpha(c_1) \vee \alpha(c_2))$.
  By similar arguments, $\alpha(c_1) \vee \alpha(c_2') = (\Delta_{c_1} \cup \Delta_3')(\alpha(c_1)) \vee (\Delta_{c_2'} \cup \Delta_3'')(\alpha(c_2'))$, *where* $\Delta_3' = \exists d_1, \exists d_2, ..,$ *where* $d_i \in (\tilde{V} \cap Var(\alpha(c_1)))$ *and* $\Delta_3' = \exists e_1, \exists e_2, ..,$ *where* $e_i \in (\tilde{V} \cap Var(\alpha(c_2)))$.
  Then $Res(\alpha(c_1) \vee \alpha(c_2')) = \Delta_{c_1} \cup \Delta_{c_2} \cup \Delta_3' \cup \Delta_3''(\alpha(c_1) \vee \alpha(c_2'))$, *since all variables in* $\Delta_3' \cup \Delta_3''$ *are new distinct variables from the ones in* $\Delta_{c_1} \cup \Delta_{c_2}$.

- $\tilde{\exists}^a X\ \alpha(c_1) = \alpha(\tilde{\exists} X\ c_1)$.
  $\tilde{\exists}^a X \alpha(c_1)$ *is equal, by definition to* $\exists X\ \alpha(c_1)$.
  Let $c_1 = \Delta_{c_1}(c_1)$. *Then* $\exists X\ \alpha(c_1) = \exists X \cup \Delta_{c_1} \cup \Delta_{\alpha(c_1)}\ \alpha(c_1)$.

We obtain an equivalent formula if we consider $\alpha(\tilde{\exists} X\ c_1)$ which is, by definition, equal to $\alpha(\exists X\ c_1)$.
By definition of α we obtain $\exists X \cup \Delta_{c_1} \cup \Delta_{\alpha(c_1)}$, as we wanted.

- $\tilde{\forall}^a X\ \alpha(c_1) = \tilde{\forall} X\ c_1$.
  The proof is similar to the previous one.

Lemma A.5 (4.7) *Consider $c_1 \in \mathcal{NC}$ such that $c_1 = (\bigwedge_i X_i = t_i)$. $\tilde{\neg}^a \alpha_c(c_1) = \alpha_c(\tilde{\neg}(c_1))$.*

Proof A.6 *We have to show that $\mathcal{M}(Res(\neg(\alpha_c(c_1)))) = \alpha_c(Res(\neg c_1))$.*
Consider the particular form of the constraint c_1, by the definition of α, we have that $(\neg(\alpha_c(c_1)))$ can be expressed as a disjunction $\bigvee_i A_i$ of inequalities $A_i = X_i \neq t_i$. In a similar way $\neg c_1$ can be expressed as a disjunction $\bigvee_i A'_i$ of corresponding inequalities $A'_i = X_i \neq t'_i$, where t'_i is the term obtained cutting the term t at depth k.
Since Res does not have any effect on a disjunction of inequalities, we need to show $\mathcal{M}(\bigvee_i A_i) = \alpha_c(\bigvee_i A'_i)$, only. Let us show that

- *for each inequality $X_i \neq t_i \in \mathcal{M}(A_i)$, $X_i \neq t_i \in \alpha_c(A'_i)$.*
  Suppose $X_i \neq t_i \in \mathcal{M}(A_i)$, then $X_i = t_i$ was one of the equality in the conjunction c_1 with $|t_i| \leq k$. Then $X_i \neq t_i$ belongs to the disjunction in $\neg c_1$. Since $|t_i| \leq k$, $X_i \neq t_i$ belongs to the disjunction in $\alpha_c(\neg c_1)$.

- *for each inequality $X_i \neq t_i \in \alpha_c(A'_i)$, $X_i \neq t_i \in \mathcal{M}(A_i)$.*
  Suppose $X_i \neq t_i \in \alpha_c(A_i)$ then the depth of t_i has to be less or equal k. Then $X_i = t_i \in \alpha_c(A_i)$, and $X_i \neq t_i \in \neg(\alpha_c(c_1))$.
  Since t_i has not been cut by α ($|t_i| \leq k$) $X_i \neq t_i \in \mathcal{M}(A_i)$.
  This completes the proof.

Theorem A.6 *[4.9] $\alpha(T^{\mathcal{B}}_{S,P}(\gamma([I^a]_{/\equiv}))) \leq T^{\mathcal{B}^a}_{S,P}([I^a]_{/\equiv})$.*

Proof A.7 *In this proof, for a sake of simplicity, let us abbreviate $T^{\mathcal{B}}_{S,P}$ with T^c_P, and $T^{\mathcal{B}^a}_{S,P}$ with T^a_P.*
We need to show that $[\tilde{\alpha}(T^c_P(\gamma([I^a]_{/\equiv})))]_{/\equiv} \leq [T^a_P([I^a]_{/\equiv})]_{/\equiv}$.
To this aim we will show that for all $c|A \in \tilde{\alpha}(T^c_P(\gamma([I^a]_{/\equiv})))$ there exists a $\overline{c}|A \in T^a([I^a]_{/\equiv})$ and $c \leq_a \overline{c}$.
Suppose $c|A \in \tilde{\alpha}^+ T^c_P \gamma([I^a]_{/\equiv})$ there exists a $c'|A \in T^{c\,+}_P(\gamma([I^a]_{/\equiv}))$ such that $c = \alpha_c(c')$.
There exists then a clause $C = A : -d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n \in P$ with local variables Y, such that there exists
$c_1|A_1, ..., c_m|A_m \in \gamma^+([I^a]_{/\equiv})$,
$c_{m+1}|A_{m+1}, ..., c_n|A_n \in \gamma^-([I^a]_{/\equiv})$ and $c' = \tilde{\exists} Y\ (d \tilde{\wedge} c_1 \tilde{\wedge} \ldots \tilde{\wedge} c_n)$.
By definition of γ,
there exist $c'_1|A_1, ..., c'_m|A_m \in \downarrow^a I^{a+}$, $c'_{m+1}|A_{m+1}, ..., c'_n|A_n \in \downarrow^a I^{a-}$, such that $\alpha_c(c_i) = c'_i$.

Now since it exists the clause $C \in P$ with local variables Y,
there exists a corresponding clause in C in $\alpha(P)$ with local variables Y', and
$c'_1|A_1, ..., c'_m|A_m \in \downarrow^a I^a+$, $c'_{m+1}|A_{m+1}, ..., c'_n|A_n \in \downarrow^a I^a-$,
we can derive $\overline{c}|A \in T_P^{a+}([I^a]_{/\tilde{\equiv}})$,
where $\overline{c} = \tilde{\exists}^a Y' \; (\alpha_c(d) \overset{a}{\tilde{\wedge}} c'_1 \overset{a}{\tilde{\wedge}} \ldots \overset{a}{\tilde{\wedge}} c'_n)$.
By correctness of the abstract operations and by definition of α_c, we have that $c = \alpha_c(c') = \alpha_c(\tilde{\exists} Y \; (d \tilde{\wedge} c_1 \tilde{\wedge} \ldots \tilde{\wedge} c_n)) \leq_a$
$\tilde{\exists}^a Y' \; (\alpha_c(d) \overset{a}{\tilde{\wedge}} \alpha_c(c_1) \overset{a}{\tilde{\wedge}} \ldots \overset{a}{\tilde{\wedge}} \alpha_c(c_n)) = \tilde{\exists}^a Y' \; (\alpha_c(d) \overset{a}{\tilde{\wedge}} c'_1 \overset{a}{\tilde{\wedge}} \ldots \overset{a}{\tilde{\wedge}} c'_n) = \overline{c}$.
In a analogous way we prove that for each $c|A \in \tilde{\alpha}^- T_P^c \gamma([I^a]_{/\tilde{\equiv}})$ *there exists a* $\overline{c}|A \in T_P^{a-}([I^a]_{/\tilde{\equiv}})$ *such that* $c \leq_a \overline{c}$.
Suppose we have $c|A \in \alpha^- T_P^c \gamma([I^a]_{/\tilde{\equiv}})$, *then there exists* $c'|A \in T_P^{c-} \gamma([I^a]_{/\tilde{\equiv}})$ *such that* $\alpha_c(c') = c$.
Then for each clause $C_k = A \leftarrow d_k|A_{k,1}, ..., A_{k,m_k}, \beta_k$.
defining A in P with local variables Y_k, there exist $e_{k,1}|A_{k,1}, ..., e_{k,m_k}|A_{k,m_k} \in \gamma^-([I]_{/\tilde{\equiv}})$,
$e_{k,m_k+1}|A_{k,m_k+1}, ..., e_{k,n_k}|A_{k,n_k} \in \gamma^+([I]_{/\tilde{\equiv}})$, $n_k \geq m_k$
where for $m_k + 1 \leq j \leq n_k$, $\neg A_{k,j}$ *occurs in* β_k,
and $c'' = \tilde{\wedge} \; \tilde{\forall} Y_k(\neg d_k \tilde{\vee} e_{k,1} \ldots \tilde{\vee} e_{k,n_k})$.
By definition of γ there exist
$e'_{k,1}|A_{k,1}, ..., e'_{k,m_k}|A_{k,m_k} \in \downarrow^a I^{a-}$ *and* $e'_{k,m_k+1}|A_{k,m_k+1}, ..., e'_{k,n_k}|A_{k,n_k} \in \downarrow^a I^{a+}$,
$\alpha_c(e_{k,j}) = e'_{k,j}$.
Now C_ks are the only clauses in P which define A, by construction, the corresponding clauses of C_k's are the only clause defining A in $\alpha(P)$.
Moreover Y'_k are the local variables of the corresponding clause of C_k in $\alpha(P)$, there exist $e'_{k,1}|A_{k,1}, ..., e'_{k,m_k}|A_{k,m_k} \in \downarrow^a I^a-$,
$e'_{k,m_k+1}|A_{k,m_k+1}, ..., e'_{k,n_k}|A_{k,n_k} \in \downarrow^a I^{a+}$, $n_k \geq m_k$
where for $m_k + 1 \leq j \leq n_k$, $\neg A_{k,j}$ *occurs in* β_k,
then $\overline{c}|A \in T_P^{a-}([I^a]_{/\tilde{\equiv}})$,
where $\overline{c} = \overset{a}{\tilde{\bigwedge}} \tilde{\forall}^a Y_k(\tilde{\neg}^a \alpha_c(d_k) \overset{a}{\tilde{\vee}} e'_{k,1} \ldots \overset{a}{\tilde{\vee}} e'_{k,n_k})$.
By correctness of the abstract operations and by definition of α_c, we have that $c = \alpha_c(c'') = \alpha_c(\overset{a}{\tilde{\bigwedge}} \tilde{\forall} Y_k(\tilde{\neg} d_k \tilde{\vee} e_{k,1} \ldots \tilde{\vee} e_{k,n_k}) \leq_a$
$\overset{a}{\tilde{\bigwedge}} \; \tilde{\forall}^a Y'_k(\tilde{\neg}^a \alpha_c(d_k) \overset{a}{\tilde{\vee}} \alpha_c(e_{k,1}) \ldots \overset{a}{\tilde{\vee}} \alpha_c(e_{k,n_k})) = \overset{a}{\tilde{\bigwedge}} \; \tilde{\forall}^a Y'_k(\tilde{\neg}^a \alpha_c(d_k) \overset{a}{\tilde{\vee}} e'_{k,1} \ldots \overset{a}{\tilde{\vee}} e'_{k,n_k}) = \overline{c}$.

Lemma A.7 *[4.10] Consider* $c_1, c_2 \in \mathcal{ANC}$ *then* $\exists c'_1, c'_2 \mathcal{NC}$ *such that* $\alpha_c(c'_1) = c_1$, $\alpha_c(c'_2) = c_2$ *and* $c_1 \overset{a}{\tilde{\wedge}} c_2 = \alpha_c(c'_1 \tilde{\wedge} c'_2)$, *and* $c_1 \overset{a}{\tilde{\vee}} c_2 = \alpha_c(c'_1 \tilde{\vee} c'_2)$.

Proof A.8 *Consider $Res(c_1 \wedge \tilde{c}_2)$, where $\tilde{c}_2$ is the renamed apart version (w.r.t. c_1, x, y) of c_2.*
Consider c'_1 as c_1, where all the variables in $\tilde{V}$ (which appears only in the right part of equalities because $c_1 \in \mathcal{ANC}$) are substituted with terms in the following matter: if $V \in Var(Res(c_1 \wedge \tilde{c}_2))$ then $V = a$ otherwise $v = t$

where t is the term assigned to V by the solution of $Res(c_1 \wedge \tilde{c}_2)$.
It is worth noting that, by construction, $\alpha_c(c_1') = c_1$ and $\alpha_c(c_2') = c_2$.
We show now that $c_1 \overset{a}{\tilde{\wedge}} c_2 = \alpha_c(c_1' \tilde{\wedge} c_2')$.
$c_1 \overset{a}{\tilde{\wedge}} c_2 = \mathcal{M}(Res(c_1 \wedge \tilde{c}_2))$, while $\alpha_c(c_1' \tilde{\wedge} c_2') = \alpha_c(Res(c_1' \wedge \tilde{c}_2'))$, where $\tilde{c}_2'$ is the renamed apart version (w.r.t. c_1', x, y) of c_2.
By construction, $Res(c_1' \wedge \tilde{c}_2')$ is equal to $Res(c_1 \wedge \tilde{c}_2)$ where the variables in $\tilde{V}$ are replaced by a. It is immediate to see that for every equality that $\mathcal{M}$ cuts in $Res(c_1 \wedge \tilde{c}_2)$, α cuts it in the same way in $Res(c_1' \wedge \tilde{c}_2')$ and that for every equality that α cuts in $Res(c_1' \wedge \tilde{c}_2')$ $\mathcal{M}$ cuts it or leave unchanged in $Res(c_1 \wedge \tilde{c}_2)$ giving the same result.
Moreover for each inequality (longer than k) that $\mathcal{M}$ substitute with true in $Res(c_1 \wedge \tilde{c}_2)$, α does the same in $Res(c_1' \wedge \tilde{c}_2')$ because that inequality is still deeper than k, and also for each inequality (which contain variables in $\tilde{V}$) that $\mathcal{M}$ substitute with true in $Res(c_1 \wedge \tilde{c}_2)$, α does the same in $Res(c_1' \wedge \tilde{c}_2')$, because, by construction of c_1' and c_2, that inequality in $Res(c_1' \wedge \tilde{c}_2')$ is longer than k.
On the other hand α does not cut any other inequalities in $Res(c_1' \wedge \tilde{c}_2')$ because, by construction of c_1' and c_2, we have substitute just the variables in $\tilde{V}$ with terms longer than 0.
By construction of c_1' and c_2', it is easy to see that $c_1 \overset{a}{\tilde{\vee}} c_2 = \alpha(c_1' \tilde{\vee} c_2')$.

Theorem A.8 (4.11) $[T^{\mathcal{B}^a}_{S,P}([I^a]_{/\tilde{\equiv}})]_{/\tilde{\equiv}} \leq \alpha(T^{\mathcal{B}^a}_{S,P}(\gamma([I^a]_{/\tilde{\equiv}})))$

Proof A.9 *We have to show that $[T^a_P([I^a]_{/\tilde{\equiv}})]_{/\tilde{\equiv}} \leq [\tilde{\alpha}(T^c_P(\gamma([I^a]_{/\tilde{\equiv}})))]_{/\tilde{\equiv}}$, where T^a_P and T^c_P are as in the proof of theorem A.6.*
We show that for each $c|A \in T^a_P([I^a]_{/\tilde{\equiv}})$, $c|A$ also belongs to $\tilde{\alpha} T^c_P \gamma([I^a]_{/\tilde{\equiv}})$.
Suppose $c|A \in T^{a+}_P(I^a)$ then there exists a clause
$C = A : -d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n \in P$ with local variables Y,
by hypothesis $P = \alpha(P)$ that $C \in \alpha(P)$
and $c_1|A_1, ..., c_m|A_m \in \downarrow^a I^a+$, $c_{m+1}|A_{m+1}, ..., c_n|A_n \in \downarrow^a I^a-$,
$c = \tilde{\exists}^a Y\ (\alpha_c(d) \overset{a}{\tilde{\wedge}} c_1 \overset{a}{\tilde{\wedge}} \ldots \overset{a}{\tilde{\wedge}} c_n)$.
Consider now $c_1', ..., c_m', c_{m+1}', ..., c_n'$, such that $\alpha_c(c_i') = c_i$ and
$\alpha(c_1' \tilde{\wedge} \ldots \tilde{\wedge} c_n') = c_1 \overset{a}{\tilde{\wedge}} \ldots \overset{a}{\tilde{\wedge}} c_n$. Such c_i' exist by the lemma A.7. By definition of γ, $c_1'|A_1, ..., c_m'|A_m \in \gamma^+(I^a)$, and $c_{m+1}'|A_{m+1}, ..., c_n'|A_n \in \gamma^-(I^a)$.
Then considering $C = A : -d|A_1, ..., A_m, \neg A_{m+1}, ..., \neg A_n \in P = \alpha(P)$ with local variables Y,
we obtain that $\overline{c} = \tilde{\exists} Y\ (d \tilde{\wedge} c_1' \tilde{\wedge} \ldots \tilde{\wedge} c_n') \in T^{c+}_P \gamma(I^a)$.
By hypothesis $d = \alpha(d)$ and by the lemma A.7, we obtain that $\alpha_c(\overline{c}) = c$ then $c|A \in \alpha^+ T^c_P \gamma(I^a)$
Using also the second part of lemma A.7, we also obtain that for each $c|A \in T^{a-}_P(I^a)$, $c|A \in \alpha^- T^c_P \gamma(I^a)$.