DECLARATIVE INCORRECTNESS DIAGNOSIS IN CONSTRAINT LOGIC PROGRAMMING

FRANÇOIS LE BERRE AND ALEXANDRE TESSIER
LIFO – UNIVERSITÉ D'ORLÉANS – BP 6759 – 45067 ORLÉANS CEDEX 2 – FRANCE
{flb,tessier}@lifo.univ-orleans.fr, http://www.univ-orleans.fr/~tessier

Abstract. Our concern in this paper is the declarative incorrectness diagnosis of constraint logic programs. Many techniques have been developed for LP but cannot be merely adapted to CLP. Constraint logic program semantics is redefined, using a reject criterion, in term of skeletons. Skeletons give an intrinsic definition to the answers provided by a program. The reject criterion can take into account the behaviour of an incomplete constraint solver. The main contribution of this paper is to prove that: if there exists a wrong answer then there is an incorrect clause in the program, and this clause occurs in the answer skeleton. Moreover, we give an algorithm which, given an incorrectness symptom, localizes a faulty clause and the circumstances of its incorrectness. Above all, there are new notions adapted to CLP framework.

1 Introduction

Program debugging is known to be a time consuming task in the programming process, but, constraint logic program debugging is relatively unexplored. The proposed approach in this paper for incorrectness localisation is declarative diagnosis. Declarative error diagnosis in Logic Programming (LP) was introduced in [8] under the name of *algorithmic debugging*. In Constraint Logic Programming (CLP), the necessity of declarative diagnosis is as much significant as in LP. In this context, declarative means that there is no need for the programmer to understand the computational behaviour of the system. It is evident that a computer cannot diagnose errors in a program without being told a part of what should be computed. But, only intended declarative semantics of the program is required. Indeed, a great strength of CLP is its declarative nature, and, for a declarative language, it is essential to consider a declarative notion of error. It would be incoherent to use only low level tools. Tracing techniques are useful, but in addition to their direct link to the computational behaviour, they quickly become extraordinarily difficult (long and ineffective) to use. The success of a declarative debugging tool is directly related to the language declarativity level. From this viewpoint, CLP is used in a much more declarative way than LP. In particular, negation by failure, cut, is, var, etc. are useless because of the availability of global constraints, disequations, etc.

This paper is only devoted to errors which lead to wrong answers. In particular, errors leading to missing answers are not considered. The aim is to give definitions of incorrectness symptom and incorrectness in the CLP formalism and to prove constructively that if there exists a symptom then there exists an incorrectness.

It is not possible to merely adapt LP techniques to CLP. Herbrand interpretations do not represent program semantics any more. Some elements of the constraint domain are not finely expressible in the program language (e.g. π in CLP($\mathcal{R}$)). When we debug, we like to stay in the program language. With respect to classical theoretical frameworks [7], practical implementations use incomplete constraint solvers, that is to say solvers which do not end the computation while the current store is unsatisfiable (see Ex. 5). It is important, for practical purpose, to take into account this feature of real implementations. So, new theoretical foundations are necessary.

We start by reformulating completely the program semantics bases. Our approach of program semantics is based on an extension to CLP of the "grammatical view" of LP introduced by [2]. In fact, the basic notion is skeletons, which clearly express the relation between declarative and operational semantics. Moreover, we take into account the incompleteness of constraint solvers via a reject criterion. Classical results are found when the reject criterion is defined either by a domain or a theory.

The main contribution is that if there exists a wrong answer then there exists an incorrect clause in the program, and such a clause occurs in the answer skeleton.

The paper is organized as follows: Sect. 2 defines the language and notations, gives the motivations for the formal notions, and, formally defines them when interpretation of the constraints is based on a pre-interpretation. Sect. 3 abstracts the pre-interpretation to take into account incompleteness of constraint solvers. In this framework, we give a diagnosis algorithm. Sect. 4 concludes the paper.

2 Theoretical Viewpoint

2.1 Terminology and Notations

Let us consider once and for all four sets which define the program language: an infinite set of *variables* V; a set of *function symbols* Σ; a set of *constraint predicate symbols* Π_c; a set of *program predicate symbols* Π_p.

The set of *terms* is built, as usual, over (V, Σ). An *atom* is an atomic formula built over (V, Σ, Π_p). The constraint language CONST is a subset of the first order language built over (V, Σ, Π_c). We assume that it is closed by existential quantification, conjunction and contains the two logic constant *true* and *false*. A *constraint* is a formula of CONST.

A *clause* is a $n+2$-tuple $(0 \leq n)$ denoted by $a_0 \leftarrow c \,\Box\, a_1, \ldots, a_n$, where each a_i is an atom and c is a constraint. Given a clause R of the previous form, we define $head(R) = a_0$, $body(R) = c \,\Box\, a_1, \ldots, a_n$, $constraint(R) = c$ and $arity(R) = n$. A *goal* is a clause without head. A *program* is a set of clauses. A *constrained atom* is a pair denoted by $c \rightarrow a$ where a is an atom and c is a constraint.

Notations. $\tilde{x}$ denotes a sequence of distinct variables $x_1, \ldots, x_n$. If F is a formula built over $(V, \Sigma, \Pi_p \cup \Pi_c)$ then $var(F)$ denotes the free variable sequence of F. If c is a constraint, $\tilde{x}$ is $x_1, \ldots, x_n$, $\tilde{y}$ are the free variables of c which are not in $\tilde{x}$ and a is an atom then $\exists_{\tilde{x}}\, c$ denotes $\exists x_1 \cdots \exists x_n\, c$; $\tilde{\exists}\, c$ denotes $\exists_{var(c)}\, c$; $\exists_{-\tilde{x}}\, c$ denotes $\exists_{\tilde{y}}\, c$; $\exists_{-a}\, c$ denotes $\exists_{-var(a)}\, c$.

2.2 Motivations

To motivate the framework and the definitions, we first consider that intended signification of constraints is based on a pre-interpretation $\mathcal{D}$ with domain D.

Let P be a CLP program and $\leftarrow g$ be a goal. The answer constraint r to $\leftarrow g$ is considered abnormal if there exists a valuation v in the underlying pre-interpretation

$\mathcal{D}$ such that v satisfies r and $v(g)$ should not evaluate to $true$ with respect to the expected properties of P. We say that v is an *anomaly*.

We point out that anomalies cannot be caused by the possible incompleteness of the constraint solver. Indeed, if the solver provides an unsatisfiable answer constraint r (i.e. always wrong in $\mathcal{D}$) then, for each valuation v, $v(r) = false$, therefore v cannot be an anomaly for the goal.

Anomalies are due to P. P is wrong in the sense that P contains at least an incorrect clause.

Our concern is to provide an assistance to localize, as fast as possible, a faulty clause and the conditions of its abnormal behaviour. It is not necessary to review each clause of the program, but the matter is to examine clauses which have been used to construct the abnormal answer. Moreover, we try to make clear the circumstances under which the clause is faulty. At first, these circumstances are formalized by a valuation. Then, to take into account the fact that elements of the domain are only manipulated through constraints, they will be formalized by a constraint.

Example 1 Let FIB be the program:

$$fib(0,0) \leftarrow true$$
$$fib(1,1) \leftarrow true$$
$$fib(x+1, y_1+y_2) \leftarrow x > 0 \,\Box\, fib(x,y_1), fib(x,y_2)$$

The program language (Σ, Π_c, Π_p) is defined from symbols which occur in FIB. The underlying pre-interpretation for the constraints is $\mathcal{N}$, whose domain is $\mathbb{N}$, with the usual interpretation for function symbols and constraint predicate symbols.

The program predicate symbol fib is assumed to define the binary relation over $\mathbb{N}$ such that the second argument is the result of the Fibonacci mapping (called $fibo$) applied to the first argument.

The answer constraint $x = 1+1 \wedge y = 1$ to the goal $\leftarrow y = 1 \,\Box\, fib(x, y+1)$ is abnormal. A valuation v_0 such that $v_0(x) = 2$ and $v_0(y) = 1$ satisfies the answer constraint but should not satisfy the body of the goal ($fibo(2) = 1$).

The faulty clause is $fib(x+1, y_1+y_2) \leftarrow x > 0 \,\Box\, fib(x,y_1), fib(x,y_2)$ and a possible patching is $fib(x+1, y_1+y_2) \leftarrow x > 0 \,\Box\, fib(x,y_1), fib(x-1,y_2)$.

2.3 From $\mathcal{D}$-Definitions to Definitions

A *$\mathcal{D}$-atom* is a $(n+1)$-tuple denoted by $p(d_1, \ldots, d_n)$ where p is a n-ary program predicate symbol and $d_1, \ldots, d_n$ are elements of the domain D ($\mathcal{D}$-atoms are not elements of the language). The $\mathcal{D}$-base is the set of $\mathcal{D}$-atoms. A *$\mathcal{D}$-interpretation* $I^P_{\mathcal{D}}$ is a subset of the $\mathcal{D}$-base. It defines an interpretation. A *valuation* is a mapping from V to D. There is a natural extension of a valuation v denoted also by v which maps from terms to D, from constraints to $\{true, false\}$ and from atoms to $\mathcal{D}$-base. $I^P_{\mathcal{D}}$ is a *$\mathcal{D}$-model* of P if for each clause $(head \leftarrow c \,\Box\, body) \in P$, for every valuation v, $v(c) = true$ and $v(body) \subseteq I^P_{\mathcal{D}}$ implies $v(head) \in I^P_{\mathcal{D}}$. We say that a clause is not valid in $I^P_{\mathcal{D}}$ if there exists a valuation which satisfies (in $I^P_{\mathcal{D}}$) the body of the clause but does not satisfy the head. A program P has a least $\mathcal{D}$-model denoted by $M^P_{\mathcal{D}}$.

Let P be a program and $\leftarrow c_g \,\Box\, b_{g_1}, \ldots, b_{g_m}$ be a goal. Answer constraint r is regarded as abnormal because in the underlying pre-interpretation $\mathcal{D}$ there exists a valuation v which is an anomaly. $\{v(b_{g_i})\}_{i=1,\dots,m} \subseteq M^P_{\mathcal{D}}$ and v satisfies c_g (because r is $\exists_{-\tilde{x}}(r' \wedge c_g)$, where $\tilde{x}$ are the free variables of the goal). The anomaly lies in the fact that an atom b_{g_i} of the goal is such that $v(b_{g_i})$ should not be in $M^P_{\mathcal{D}}$.

When we wrote the program P we wanted that the relations defined by P be true in an intended $\mathcal{D}$-interpretation $I^P_{\mathcal{D}}$. This $\mathcal{D}$-interpretation formalizes the intended semantics of the program P.

The anomaly is that $r \to c_g \wedge b_{g_1} \wedge \cdots \wedge b_{g_m}$ is not valid in $I^P_{\mathcal{D}}$ because there exists b_{g_i} such that $v(b_{g_i}) \notin I^P_{\mathcal{D}}$ ($v(c_g)$ does not depend on $I^P_{\mathcal{D}}$). The anomaly exists because $M^P_{\mathcal{D}} \not\subseteq I^P_{\mathcal{D}}$. This first motivates the following definitions.

Definition 2.1 *An* incorrectness $\mathcal{D}$-symptom *of P wrt $I^P_{\mathcal{D}}$ is a $\mathcal{D}$-atom in $M^P_{\mathcal{D}} - I^P_{\mathcal{D}}$.*

A $\mathcal{D}$-incorrectness *of P wrt $I^P_{\mathcal{D}}$ is a pair $\langle h \leftarrow c \,\Box\, b_1, \dots, b_n; v\rangle$ such that $v(c) = true$, for $i = 1, \dots, n$, $v(b_i) \in I^P_{\mathcal{D}}$ and $v(h) \notin I^P_{\mathcal{D}}$.*

If there exists an incorrectness $\mathcal{D}$-symptom of P wrt $I^P_{\mathcal{D}}$ then $M^P_{\mathcal{D}} \not\subseteq I^P_{\mathcal{D}}$, thus $I^P_{\mathcal{D}}$ is not a $\mathcal{D}$-model of P ($M^P_{\mathcal{D}}$ is the least $\mathcal{D}$-model of P), thus there exists a clause in P which is not valid in $I^P_{\mathcal{D}}$ and thus there exists a $\mathcal{D}$-incorrectness of P wrt $I^P_{\mathcal{D}}$. This clause can be viewed as an error which causes the symptom.

Remark. The converse is wrong. For example, let P be the program $\{p \leftarrow true \,\Box\, q\}$ and $I^P_{\mathcal{D}} = \{q\}$. $M^P_{\mathcal{D}} = \emptyset \subseteq I^P_{\mathcal{D}}$, thus there is no incorrectness $\mathcal{D}$-symptom of P wrt $I^P_{\mathcal{D}}$. But $I^P_{\mathcal{D}}$ is not a $\mathcal{D}$-model of P, and, for each valuation v, $\langle p \leftarrow true \,\Box\, q; v\rangle$ is a $\mathcal{D}$-incorrectness of P wrt $I^P_{\mathcal{D}}$.

According to the previous definitions, if there exists an incorrectness $\mathcal{D}$-symptom then there exists a $\mathcal{D}$-incorrectness. The clause of the $\mathcal{D}$-incorrectness is a faulty clause and the valuation explains why it is faulty.

Example 2 The intended semantics of FIB is formalized by the $\mathcal{N}$-interpretation $I^{FIB}_{\mathcal{N}} = \{fib(d_1, d_2) \mid fibo(d_1) = d_2\}$ The least $\mathcal{N}$-model of FIB is $M^{FIB}_{\mathcal{N}} = \{fib(d_1, d_2) \mid \text{if } d_1 = 0 \text{ then } d_2 = 0 \text{ else } d_2 = 2^{d_1 - 1}\}$
$M^{FIB}_{\mathcal{N}} \not\subseteq I^{FIB}_{\mathcal{N}}$, i.e. there exists an incorrectness $\mathcal{N}$-symptom of FIB wrt $I^{FIB}_{\mathcal{N}}$, thus there exists a $\mathcal{N}$-incorrectness of FIB wrt $I^{FIB}_{\mathcal{N}}$.
The $\mathcal{N}$-atom $fib(2, 2)$ is an incorrectness $\mathcal{N}$-symptom ($fibo(2) \neq 2$).
The pair $\langle fib(x+1, y_1+y_2) \leftarrow x > 0 \,\Box\, fib(x, y_1), fib(x, y_2); v_1\rangle$, is a $\mathcal{N}$-incorrectness of FIB wrt $I^{FIB}_{\mathcal{N}}$, where v_1 is such that $v_1(x) = 1$, $v_1(y_1) = 1$, $v_1(y_2) = 1$.

The point of interest in CLP is that, usually, v cannot be expressed in the language because there is no corresponding ground term for each element of D. When we debug, we would stay in the program language. To remain in the language we propose to change the valuation (which explains the incorrectness of a faulty clause) by a constraint which approximates the valuation in a sense. Indeed, valuations are only manipulated through constraints.

We say that a constraint r is a *witness* of the invalidity of $h \leftarrow c \,\Box\, b_1, \dots, b_n$ in the $\mathcal{D}$-interpretation $I^P_{\mathcal{D}}$ if there exists a valuation v solution of r such that $v(c) = true$, $\{v(b_i)\}_{i=1,\dots,n} \subseteq I^P_{\mathcal{D}}$, but $v(h) \notin I^P_{\mathcal{D}}$, i.e. $\models_{I^P_{\mathcal{D}}} \tilde{\exists}\,(r \wedge c \wedge b_1 \wedge \cdots \wedge b_n \wedge \neg h)$.
Remark. A clause is not valid in $I^P_{\mathcal{D}}$ iff there exists a witness of its invalidity (for example, the constraint $true$).

There is, of course, a commonplace algorithm which consists in verifying each clause of the program, that is, for each clause $head \leftarrow body$, to check if $\models_{I^P_{\mathcal{D}}} body \rightarrow head$. We can restrict the research by considering clauses which occur in the derivation which computes the abnormal answer constraint as we said before. But this is just a clause checking and it is too awkward. We would like to boil down to easier problems, no more focused on clauses but on *constrained atoms*.

Some witnesses can be more interesting than others: if $\models_{\mathcal{D}} c \rightarrow c'$ then c is a witness of the clause R implies that c' is a witness of the clause R; c provides more information than c' in the sense that c better approximates v than c'.

We say that r is a *strong witness* of the invalidity of the clause $h \leftarrow c \,\Box\, b_1, \dots, b_n$ in the $\mathcal{D}$-interpretation $I^P_{\mathcal{D}}$ if $\models_{I^P_{\mathcal{D}}} r \rightarrow c \wedge b_1 \wedge \cdots \wedge b_n$ and not($\models_{I^P_{\mathcal{D}}} r \rightarrow h$); i.e. $\models_{\mathcal{D}} r \rightarrow c$, for each $i = 1, \dots, n$, $\models_{I^P_{\mathcal{D}}} r \rightarrow b_i$ and not($\models_{I^P_{\mathcal{D}}} r \rightarrow h$). Strong witnesses are witnesses. The converse is wrong, as shown by the following example.

Example 3 The constraint $true$ is a witness of the invalidity of the clause

$$fib(x+1, y_1+y_2) \leftarrow x > 0 \,\Box\, fib(x, y_1), fib(x, y_2)$$

because of the valuation v_1 ($v_1(x) = 1, v_1(y_1) = 1, v_1(y_2) = 1$).
But $true$ is not a strong witness of its invalidity because

$$\text{not}(\models_{I_{FIB}} true \rightarrow x > 0 \wedge fib(x, y_1) \wedge fib(x, y_2)).$$

A strong witness of its invalidity is $x = 1 \wedge y_1 = 1 \wedge y_2 = 1$ (which is also a witness).

The strong witness notion motivates the following definitions of incorrectness symptom and incorrectness. We have next (in Sect. 3) to take into account incompleteness of constraint solvers.

Definition 2.2 *An* incorrectness symptom *of P wrt the $\mathcal{D}$-interpretation $I^P_{\mathcal{D}}$ is a pair $\langle \leftarrow c \,\Box\, b_1, \dots, b_n; r\rangle$, such that $\models_{M^P_{\mathcal{D}}} r \rightarrow c \wedge b_1 \wedge \cdots \wedge b_n$ and not($\models_{I^P_{\mathcal{D}}} r \rightarrow c \wedge b_1 \wedge \cdots \wedge b_n$). Since $\models_{\mathcal{D}} r \rightarrow c$ then $\models_{I^P_{\mathcal{D}}} r \rightarrow c$, therefore there exists $i \in \{1, \dots, n\}$ such that not($\models_{I^P_{\mathcal{D}}} r \rightarrow b_i$).*

An incorrectness *of P wrt the $\mathcal{D}$-interpretation $I^P_{\mathcal{D}}$ is a pair $\langle h \leftarrow c \,\Box\, b_1, \dots, b_n; r\rangle$, where r is a strong witness for the clause.*

An atomic incorrectness symptom *of P wrt $I^P_{\mathcal{D}}$ is a constrained atom $r \rightarrow b$ such that $\models_{M^P_{\mathcal{D}}} r \rightarrow b$ and not($\models_{I^P_{\mathcal{D}}} r \rightarrow b$).*

Remark. There exists an incorrectness symptom of P wrt $I^P_{\mathcal{D}}$ iff there exists an atomic incorrectness symptom of P wrt $I^P_{\mathcal{D}}$.

We will show that if there exists an incorrectness symptom then there exists a strong witness for a clause used during the computation. But before, we have to define, more precisely, answers (skeletons) and answer constraints.

2.4 Skeletons

Definition 2.3 *Let G be the set of all goals. A* skeleton *is an oriented tree, labeled by $P \cup G$, such that the degree of a node is the number of atoms in the body of its label, and the root is the unique node labeled by an element of G.*

We want to associate a constraint system to a skeleton, and, as usual, we are confronted with the problem of variable renaming.

The relation "to be a variant" is an equivalence relation over the set of clauses. We assume that nodes of a skeleton (except the root) are labeled by an equivalence class of clauses denoted by a clause of P. We define a *renaming mapping* RM for a skeleton, in the following way, for each node N labeled by R: $RM(N) = R$ when N is the root; otherwise we choose a clause $RM(N)$ in the class of R, such that, for each pair of distinct nodes N_1 and N_2, $RM(N_1)$ and $RM(N_2)$ have no shared variables.

A skeleton S and a renaming mapping RM are assumed to be fixed.

For every node N of S, if b is the i^{th} atom of the label body of $RM(N)$ and N_i is the i^{th} child of N, then we define $atom(N_i) = b$.

Let $p(\tilde{s})$ and $q(\tilde{t})$ be two atoms, where $\tilde{s}$ and $\tilde{t}$ denote sequences of terms, we define the constraint $p(\tilde{s}) = q(\tilde{t})$ which denotes either the constraint $false$ when $p \neq q$, or the constraint $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$ when $p = q$.

We associate to each node N of S a constraint. The constraint associated to the root of S is the constraint of the goal. The constraint associated to another node N, such that $RM(N) = (h \leftarrow c \,\Box\, b_1, \ldots, b_n)$, is $c \wedge (h = atom(N))$. We associate to S the constraint system $const(S)$ defined by the collection of the constraints associated to each node of S. When S is finite, we denote by $C(S)$ the conjunction of the constraints of $const(S)$ and by $AC(S)$ the constraint $\exists_{-\tilde{x}} C(S)$, where $\tilde{x}$ are the free variables of the root of S. Note that $AC(S)$ does not depend on the renaming mapping RM.

Definition 2.4 *An* answer *to the goal* $\leftarrow g$ *is a finite skeleton* S *rooted by* $\leftarrow g$ *such that* $\models_{\mathcal{D}} \tilde{\exists} C(S)$. *Then,* $AC(S)$ *is an* answer constraint *to the goal* $\leftarrow g$.

Skeletons are the declarative notion of answer. They give a straightforward and intrinsic definition to the answers provided by a program. A skeleton puts together the clauses used along a derivation leaving aside the computation rule. Answers provided by a program are in fact independent of the computation rule. An answer is a finite skeleton, rooted by the goal and not rejected in some sense. From the answer skeleton we can infer the answer constraint, as well as, by giving a computation rule (i.e. a traversal of the skeleton), the success derivation. Skeletons are a good representation of the success derivations equivalence classes modulo the computation rule. They are a good tool to link easily operational semantics and declarative semantics. Moreover, skeletons are a constructive definition of *declarative answers*. This is particularly helpful in declarative debugging. SLD-resolution can be described in terms of skeletons, a node of an SLD-tree is a partial skeleton. But this is out of scope.

Example 4 S_2 is an answer of the program FIB to the goal $\leftarrow true \,\Box\, fib(x, y)$ (for further legibility, nodes are labeled by renamed clauses according to RM).

$S_2 =$ $\boxed{\leftarrow true \,\Box\, fib(x, y)}$

$\boxed{fib(x_1 + 1, y_1 + y_2) \leftarrow x_1 > 0 \,\Box\, fib(x_1, y_1), fib(x_1, y_2)}$

$\boxed{fib(1, 1) \leftarrow true}$ $\boxed{fib(1, 1) \leftarrow true}$

The constraint associated to S_2 is $C(S_2) = (true \wedge x_1 > 0 \wedge x_1 + 1 = x \wedge y_1 + y_2 = y \wedge true \wedge 1 = x_1 \wedge 1 = y_1 \wedge true \wedge 1 = x_1 \wedge 1 = y_2)$, which is satisfiable in $\mathcal{N}$. The answer constraint is $AC(S_2) = \exists x_1 \exists y_1 \exists y_2\, C(S_2)$, which is equivalent to $x = 1+1 \wedge y = 1+1$. The pair $\langle fib(x{+}1, y_1{+}y_2) \leftarrow x > 0 \,\square\, fib(x_1, y_1), fib(x_2, y_2); x = 1 \wedge y_1 = 1 \wedge y_2 = 1\rangle$ is an incorrectness of FIB wrt $I_{\mathcal{N}}^{FIB}$. The pair $\langle \leftarrow true \,\square\, fib(x,y); x = 1{+}1 \wedge y = 1{+}1\rangle$ is an incorrectness symptom of FIB wrt $I_{\mathcal{N}}^{FIB}$. The constrained atom $(x = 1{+}1 \wedge y = 1 + 1) \rightarrow fib(x,y)$ is an atomic incorrectness symptom of FIB wrt $I_{\mathcal{N}}^{FIB}$.
S_3 is not an answer to the goal $\leftarrow true \,\square\, fib(x,y)$. Indeed, $C(S_3)$ is such that not$(\models_{\mathcal{N}} \tilde{\exists}\, C(S_3))$ ($const(S_3)$ contains $1 = x_1$ and $0 = x_1$).

$$S_3 = \begin{array}{c} \boxed{\leftarrow true \,\square\, fib(x,y)} \\ | \\ \boxed{fib(x_1+1, y_1+y_2) \leftarrow x_1 > 0 \,\square\, fib(x_1,y_1), fib(x_1,y_2)} \\ \diagup \qquad\qquad\qquad \diagdown \\ \boxed{fib(1,1) \leftarrow true} \qquad\qquad \boxed{fib(0,0) \leftarrow true} \end{array}$$

Our definition of answer constraint is equivalent to classical one's [7, 5]: if S is an answer to the goal $\leftarrow g$ then $P \models_{\mathcal{D}} AC(S) \rightarrow g$; and, if $P \models_{\mathcal{D}} c \rightarrow g$ then $\models_{\mathcal{D}} c \rightarrow \bigvee_{S \in R} AC(S)$, where R is the set of answers to the goal $\leftarrow g$.
Remark. R may be infinite. It is not always possible to replace it by a finite subset. This is possible if $\mathcal{D}$ is replaced by a theory $\mathcal{T}$, a model of which is $\mathcal{D}$, by using the finiteness theorem of the first order logic.

2.5 Symptom Implies Error

Diagnosis algorithm is invoked when, during a computation, appears a computed incorrectness symptom. Computed incorrectness symptoms are a particular case of incorrectness symptoms.

Definition 2.5 *A* computed incorrectness symptom *of P wrt the $\mathcal{D}$-interpretation $I_{\mathcal{D}}^P$ is a pair $\langle \leftarrow c_g \,\square\, b_{g_1}, \ldots, b_{g_n}; r\rangle$, such that r is an answer constraint to the goal $\leftarrow c_g \,\square\, b_{g_1}, \ldots, b_{g_n}$ and not($\models_{I_{\mathcal{D}}^P} r \rightarrow c_g \,\square\, b_{g_1}, \ldots, b_{g_n}$).*

Lemma 2.6 *There exists a computed incorrectness symptom iff there exists an incorrectness symptom.*
Proof. $\Rightarrow$: If $\langle \leftarrow c \,\square\, b_1, \ldots, b_n; r\rangle$ is a computed incorrectness symptom of P wrt $I_{\mathcal{D}}^P$ then r is an answer constraint to the goal $\leftarrow c \,\square\, b_1, \ldots, b_n$. The correctness of answer constraint shows that $P \models_{\mathcal{D}} r \rightarrow c \wedge b_1 \wedge \cdots \wedge b_n$, thus $\models_{M_{\mathcal{D}}^P} r \rightarrow c \wedge b_1 \wedge \cdots \wedge b_n$ (because $M_{\mathcal{D}}^P$ is the least $\mathcal{D}$-model of P), therefore $\langle \leftarrow c \,\square\, b_1, \ldots, b_n; r\rangle$ is an incorrectness symptom of P wrt $I_{\mathcal{D}}^P$.
$\Leftarrow$: If $\langle \leftarrow c \,\square\, b_1, \ldots, b_n; r\rangle$ is an incorrectness symptom of P wrt $I_{\mathcal{D}}^P$ then $\models_{M_{\mathcal{D}}^P} r \rightarrow c \wedge b_1 \wedge \cdots \wedge b_n$, i.e. $r \rightarrow c \wedge b_1 \wedge \cdots \wedge b_n$ evaluates to $true$ in each $\mathcal{D}$-model of P, thus $P \models_{\mathcal{D}} r \rightarrow c \wedge b_1 \wedge \cdots \wedge b_n$. The completeness of answer constraint shows that $\models_{\mathcal{D}} r \rightarrow \bigvee_{S \in R} AC(S)$, where R is the set of answers to the goal $\leftarrow c \,\square\, b_1, \ldots, b_n$. This means that for each valuation v such that $v(c) = true$ there exists $S \in R$ such that $v(AC(S)) = true$. We know that not($\models_{I_{\mathcal{D}}^P} r \rightarrow c \wedge b_1 \wedge \cdots \wedge b_n$), thus there exists a valuation v such that $v(r) = true$, therefore $v(c) = true$, and

$\{v(b_1), \dots, v(b_n)\} \not\subseteq I^P_{\mathcal{D}}$. Let $S \in R$ be a finite skeleton such that $v(AC(S)) = true$ then $\langle \leftarrow c \,\Box\, b_1, \dots, b_n; AC(S) \rangle$ is a computed incorrectness symptom of P wrt $I^P_{\mathcal{D}}$. ∎

Lemma 2.7 *If there exists a computed incorrectness symptom of P wrt $I^P_{\mathcal{D}}$ then there exists an incorrectness of P wrt $I^P_{\mathcal{D}}$.*

Proof. Let $\langle \leftarrow c_g \,\Box\, b_{g_1}, \dots, b_{g_m}; AC(S) \rangle$ be a computed incorrectness symptom of P wrt $I^P_{\mathcal{D}}$. We will show that $C(S)$ is a strong witness for a clause which labels a node of S. Note that S is a finite skeleton and $\langle \leftarrow c_g \,\Box\, b_{g_1}, \dots, b_{g_m}; C(S) \rangle$ is an incorrectness symptom of P wrt $I^P_{\mathcal{D}}$ (Lemma 2.6 $\Rightarrow$).

Let us assume that $C(S)$ is not a strong witness for any clause of S, then we will show by induction on the subtrees height of S that for each node N of S, except the root, $\models_{I^P_{\mathcal{D}}} C(S) \to atom(N)$, i.e. $C(S) \to atom(N)$ is not an atomic incorrectness symptom of P wrt $I^P_{\mathcal{D}}$, thus $\langle \leftarrow c_g \,\Box\, b_{g_1}, \dots, b_{g_m}; C(S) \rangle$ is not an incorrectness symptom of P wrt $I^P_{\mathcal{D}}$. We recall that, because of the definition of $C(S)$: $\models_{\mathcal{D}} C(S) \to c_g$, and for each node N of S, except the root, labeled by a clause $h_N \leftarrow c_N \,\Box\, b_{N_1}, \dots, b_{N_k}$ we have: $\models_{\mathcal{D}} C(S) \to c_N$; and $\models_{I^P_{\mathcal{D}}} C(S) \to h_N$ iff $\models_{I^P_{\mathcal{D}}} C(S) \to atom(N)$, because $const(S)$ contains $h_N = atom(N)$. We denote by $S(N)$ the subtree of S rooted by N. If $S(N)$ is 1 high then N is a leaf labeled by $h_N \leftarrow c_N$ and $\models_{I^P_{\mathcal{D}}} C(S) \to h_N$, therefore $\models_{I^P_{\mathcal{D}}} C(S) \to atom(N)$. If $S(N)$ is a high, $a > 1$, and $\models_{I^P_{\mathcal{D}}} C(S) \to atom(N')$, for each child N' of N. Let $h_N \leftarrow c_N \,\Box\, b_{N_1}, \dots, b_{N_k}$ be the label of N, then $\models_{I^P_{\mathcal{D}}} C(S) \to c_N \wedge b_{N_1} \wedge \cdots \wedge b_{N_k}$, thus $\models_{I^P_{\mathcal{D}}} C(S) \to h_N$, therefore $\models_{I^P_{\mathcal{D}}} C(S) \to atom(N)$. Consequently, each child N' of the root is such that $\models_{I^P_{\mathcal{D}}} C(S) \to atom(N')$, thus $\models_{I^P_{\mathcal{D}}} C(S) \to c_g \wedge b_{g_1} \wedge \cdots \wedge b_{g_m}$, therefore $\langle \leftarrow c_g \,\Box\, b_{g_1}, \dots, b_{g_m}; C(S) \rangle$ is not an incorrectness symptom of P wrt $I^P_{\mathcal{D}}$.

We have shown that if $\langle \leftarrow c_g \,\Box\, b_{g_1}, \dots, b_{g_m}; C(S) \rangle$ is an incorrectness symptom of P wrt $I^P_{\mathcal{D}}$ then there exists a clause $h_N \leftarrow c_N \,\Box\, b_{N_1}, \dots, b_{N_k}$ which labels a node of S such that $\langle h_N \leftarrow c_N \,\Box\, b_{N_1}, \dots, b_{N_k}; C(S) \rangle$ is an incorrectness of P wrt $I^P_{\mathcal{D}}$. ∎

The conclusion of this section is that if S is an answer to the goal $\leftarrow g$ and $\langle \leftarrow g; AC(S) \rangle$ is an incorrectness symptom of P wrt $I^P_{\mathcal{D}}$ then there exists a clause $head \leftarrow body$, which labels a node of S, whose free variables are $\tilde{y}$, such that $\langle head \leftarrow body; \exists_{-\tilde{y}} C(S) \rangle$ is an incorrectness of P wrt $I^P_{\mathcal{D}}$. The proof gives a (non optimized) algorithm which consists in verifying the nodes of the skeleton from the root to the leaves by looking if the constrained atoms constituted with $C(S)$ and atoms of the node label body are atomic incorrectness symptoms. Next section gives this diagnosis algorithm, but previously introduces a novel semantics which takes into account incompleteness of constraint solvers.

3 Incorrectness Diagnosis Algorithm

This section introduces a novel program semantics which abstracts the underlying pre-interpretation. It is well-known that, for practical purpose, some constraint solvers are not satisfaction complete. Furthermore, they do not have the behaviour of a theory. We cannot take into account, in theoretical works, this feature of practical implementation by changing the constraint solver by a theory.

Example 5 The CLP($\mathcal{R}$) constraint solver provides three kinds of answer: `yes` (satisfiable constraint), `no` (unsatisfiable constraint) and `maybe` (it cannot decide). It answers `yes` to $x * x = 1 \wedge x = 1$, but `maybe` to $x * x = 1$, nevertheless $\models \exists x(x * x = 1 \wedge x = 1) \rightarrow \exists x(x * x = 1)$. It answers `no` to $x = 1 \wedge x = 0$, but `maybe` to $x * x = 1 \wedge x * x = 0$, nevertheless $\models \neg \exists x(x = 1 \wedge x = 0) \rightarrow \neg \exists x(x * x = 1 \wedge x * x = 0)$.

3.1 Reject Criterion

In Sect. 2, program semantics was formalized by the least $\mathcal{D}$-model of P. A finite skeleton S was rejected if $\tilde{\exists} C(S)$ was unsatisfiable in $\mathcal{D}$. For the system, a finite skeleton S is rejected if the constraint solver answers `no` to $C(S)$. We want to abstract the pre-interpretation $\mathcal{D}$ by a *Reject Criterion* RC. A reject criterion RC is a relation over CONST. It is, in general, defined from: a pre-interpretation $\mathcal{D}$, denoted by $RC(\mathcal{D})$, c is rejected if c is unsatisfiable in $\mathcal{D}$; a theory $\mathcal{T}$ (not necessarily satisfaction complete as generally assumed [7]), denoted by $RC(\mathcal{T})$, c is rejected if for each $\mathcal{D}$-model of $\mathcal{T}$ c is rejected by $RC(\mathcal{D})$ (i.e. $\mathcal{T} \models \neg c$); a constraint solver $\mathcal{A}$, denoted by $RC(\mathcal{A})$, c is rejected if $\mathcal{A}$ answers `no` to c.

In order to simplify, we assume that if c_1 and c_2 are two constraint then: $c_1 \wedge c_2$ and $c_2 \wedge c_1$ represent the same constraint; if $\tilde{x}$ are some free variables of c_1 which have no free occurrence in c_2 then $\exists_{\tilde{x}} c_1 \wedge c_2$ and $\exists_{\tilde{x}}(c_1 \wedge c_2)$ are the same; if x and y are two variables then $\exists x \exists y\, c_1$ and $\exists y \exists x\, c_1$ are the same, $\exists x\, c_1$ and $\exists x \exists x\, c_1$ are the same.

Reject criterions verify, for each rejected constraint c, the three following properties: for each constraint c', $c \wedge c'$ is rejected; for each variable x, $\exists x\, c$ is rejected; for each variable renaming θ, $c\theta$ is rejected. These three properties are verified when RC is defined from a pre-interpretation, from a theory, from usual constraint solvers.

We say that a finite skeleton S is rejected by the reject criterion RC if $C(S)$ is rejected by RC. Definitions of answer and answer constraint are adapted to this new framework: an *answer*, according to RC, to the goal $\leftarrow g$ is a finite skeleton S, rooted by $\leftarrow g$, which is not rejected by RC; then $AC(S)$ is an *answer constraint*, according to RC, to the goal $\leftarrow g$.

The reject criterion is not supposed to have a logical behaviour. But assume that the reject criterion is deduced from an incomplete[1] constraint solvers, then: if S is an answer to the goal $\leftarrow g$ then $P, \mathcal{T} \models AC(S) \rightarrow g$; if $P \models c \rightarrow g$ then $\mathcal{T} \models c \rightarrow \bigvee_{S \in R} AC(S)$, where R is a finite subset of the answers to the goal $\leftarrow g$.

3.2 Abstraction of the Intended Semantics

As we abstract the underlying pre-interpretation $\mathcal{D}$, we abstract the intended $\mathcal{D}$-interpretation $I^P_{\mathcal{D}}$. Indeed, the notions of incorrectness symptom and incorrectness are defined from the validity of a constrained atom in $I^P_{\mathcal{D}}$. That is, the interaction with the intended semantics is only based on constrained atoms. Therefore, the intended semantics can be formalized by a set of constrained atoms. We propose to use the classical notion of oracle $\mathcal{O}$ to formalize the intended semantics. $\mathcal{O}$ accepts or rejects constrained atoms in accordance with the intended semantics. An oracle $\mathcal{O}$ must have the following natural property: $\mathcal{O}$ rejects $c \rightarrow b$ iff $\mathcal{O}$ rejects $(\exists_{\tilde{y}} c) \rightarrow b$, where $\tilde{y}$ are some variables not in $var(b)$.

[1] An incomplete constraint solver, based on a theory $\mathcal{T}$, answers `no` or `maybe` to c when $\mathcal{T} \models \neg c$, and it answers `yes` or `maybe` to c when $\mathcal{T} \models \tilde{\exists} c$

Remark. We assume that symptoms are not due to the system, thus the intended semantics of a program must be in accordance with the reject criterion. For example, if $c \to a$ is expected and $c \wedge c'$ is not rejected then $c \wedge c' \to a$ is also expected. But, the fact that $c \to a$ is expected while c is rejected is not a problem in the incorrectness framework.

This section is more general than Sect. 2, in the sense that we make two abstractions: $\mathcal{D}$ is abstracted by RC and $I^P_{\mathcal{D}}$ is abstracted by $\mathcal{O}$. Now, we lift definitions:

Definition 3.1 *A* computed incorrectness RC-symptom *of P wrt $\mathcal{O}$ is a pair $\langle \leftarrow c_g \,\square\, b_{g_1}, \ldots, b_{g_n}; r\rangle$ such that r is an answer constraint according to RC to the goal $\leftarrow c_g \,\square\, b_{g_1}, \ldots, b_{g_n}$ and there exists $i \in \{1, \ldots, n\}$ such that $r \to b_{g_i}$ is rejected by $\mathcal{O}$.*

An incorrectness *of P wrt $\mathcal{O}$ is a pair $\langle h \leftarrow c \,\square\, b_1, \ldots, b_n; r\rangle$ such that for each $i \in \{1, \ldots, n\}$, $(r \wedge c) \to b_i$ is not rejected by $\mathcal{O}$ and $(r \wedge c) \to h$ is rejected by $\mathcal{O}$.*

Lemma 3.2 *If there exists a computed incorrectness RC-symptom of P wrt $\mathcal{O}$ then there exists an incorrectness of P wrt $\mathcal{O}$.*

Proof. There is just to lift the proof of Lemma 2.7. If $\langle \leftarrow c_g \,\square\, b_{g_1}, \ldots, b_{g_n}; r\rangle$ is a computed incorrectness RC-symptom of P wrt $\mathcal{O}$ then there exists a finite skeleton S not rejected by RC, rooted by $\leftarrow c_g \,\square\, b_{g_1}, \ldots, b_{g_n}$, such that $AC(S) = r$ and there exists a renamed clause *head* $\leftarrow$ *body* according to RM labelling a node of S such that $\langle head \leftarrow body; r\rangle$ is an incorrectness of P wrt $\mathcal{O}$. ∎

3.3 The Algorithm

The proposed algorithm consists on a top-down traversal of the skeleton along a branch from the goal to an incorrect clause, asking questions to the oracle. The oracle can be a human, but also an automatic system based on a partial specification of the intended semantics.

We emphasize that our algorithm uses the constraint solver of the system. Usually, the CLP system presents answer constraints into a simplified form. We want to question the oracle with constrained atoms whose constraint is simplified as well as answer constraints provided by the system. We question oracle on $(\exists_{-var(b)} r) \to b$, where $\exists_{-var(b)} r$ is simplified as well as answer constraints by the constraint solver, rather than to question it on $r \to b$. Then determining if r is a correct answer constraint for the goal $\leftarrow c \,\square\, b_1, \ldots, b_n$ is not easier than determining if $(\exists_{-var(b)} r) \to b$ is expected (especially if b has few variables or is ground).

Algorithm uses the following functions: `C(S)`: constraint associated to the skeleton `S`; `root(S)`: root of `S`; `child(S,N,I)`: `I`th child of the node `N` in `S`; `label(S,N)`: label of the node `N` in `S`; `arity(S,N)`: number of children of the node `N` in `S`; `i-atom(S,N,I)`: `I`th atom in the body of `label(S,N)`; `simplify(C,V)`: simplification of $\exists$`_V`C (it calls the constraint solver of the system); `var(X)`: free variables of `X`; `ask(C --> B)`: questions the oracle for the constrained atom `C` $\to$ `B`.

Declarative incorrectness diagnosis algorithm, given a computed answer S which provides an incorrect RC-symptom, is:

```
N := IncorrectNode(S)
write(   <   label(S,N)   ;   simplify(C(S),var(label(S,N)))   >    )
```

Where `IncorrectNode` is defined by:

```
function IncorrectNode(S) return node
begin
    N := root(S)
    loop
       I := 1
       loop
          if I > arity(S,N) then return N
          B := i-atom(S,N,I)
          ask(   simplify(C(S),var(B))    -->    B   )
          exit when answer is NO
          I := I + 1
       endloop
       N := child(S,N,I)
    endloop
end
```

Proof of correctness and completeness of the algorithm is a direct consequence of the proof of Lemma 3.2 and the fact that S is finite.

This algorithm provides only one incorrectness. More may exists in the skeleton. It can be adapted to provide several incorrect clauses which occurs in the skeleton.

Example 6 We recall actual semantics and intended semantics of FIB:
$M_{\mathcal{N}}^{FIB} = \{fib(0,0), fib(1,1), fib(2,2), fib(3,4), fib(4,8), fib(5,16), fib(6,32), \ldots\}$
$I_{\mathcal{N}}^{FIB} = \{fib(0,0), fib(1,1), fib(2,1), fib(3,2), fib(4,3), fib(5,5), fib(6,8), \ldots\}$
For the goal $\leftarrow x = y \,\Box\, fib(x,y)$, the answer $S4$ (clauses are renamed according to RM) provides the incorrectness symptom: $x = 2 \wedge y = 2 \rightarrow fib(x,y)$.

$$S4 = \quad \boxed{\leftarrow x = y \,\Box\, fib(x,y)}$$

$$\boxed{fib(x_1+1, y_1+y_2) \leftarrow x_1 > 0 \,\Box\, fib(x_1,y_1), fib(x_1,y_2)}$$

$$\boxed{fib(1,1) \leftarrow true} \qquad \boxed{fib(1,1) \leftarrow true}$$

$C(S)$ is $x = y \wedge x = x_1 + 1 \wedge y = y_1 + y_2 \wedge x_1 > 0 \wedge x_1 = 1 \wedge y_1 = 1 \wedge true \wedge x_1 = 1 \wedge y_2 = 1 \wedge true$, and $AC(S) = \exists x_1 \exists y_1, \exists y_2\, C(S)$ can be simplified into $x = 2 \wedge y = 2$. Now we trace the diagnosis session for skeleton $S4$.

$x = 2 \wedge y = 2 \rightarrow fib(x,y)$ expected? `NO`
$x_1 = 1 \wedge y_1 = 1 \rightarrow fib(x_1,y_1)$ expected? `YES`
$x_1 = 1 \wedge y_2 = 1 \rightarrow fib(x_1,y_2)$ expected? `YES`
Incorrectness is:
$\langle fib(x_1+1, y_1+y_2) \leftarrow x_1 > 0 \,\Box\, fib(x_1,y_1), fib(x_1,y_2); x_1 = 1 \wedge y_1 = 1 \wedge y_2 = 1\rangle$

We can better simplify interaction with oracle. When a variable is fixed (its possible value domain is a singleton) and its value can be expressed, we can change it by its value in the atom of the questions. In our example, each variable is fixed.
The fact that the SLD-tree is not finite is not a problem for incorrectness diagnosis. The fact that $x = 5 \rightarrow fib(x,x)$ misses is the problem of insufficiency.

4 Conclusion

We have formally defined the notions of incorrectness symptom and incorrectness for constraint logic programs in terms of constrained atoms. In LP, abnormal valuations are always expressible in the program language. It is not true in CLP, where valuated atoms are replaced by constrained atoms (elements of the domain are only manipulated through constraints).

We have proved that the existence of an incorrectness symptom implies that of an incorrectness. This incorrectness can be localized in the skeleton associated to the incorrectness symptom. Definition of incorrectness contains an incorrect clause but also the conditions of its incorrectness (abnormal behaviour) expressed by a constraint.

We give a diagnosis algorithm. It remains to find efficient ones, in particular with respect to the number of questions to the oracle and to implement these algorithms.

Some programs have the form: $go(\tilde{x}) \leftarrow big\_constraint$. Then we observe an incorrectness symptom and call our diagnosis which provides the incorrect clause $go(\tilde{x}) \leftarrow big\_constraint$. It is not very interesting! We can discuss here the programming methodology and prefer a better structured program: in general, it is more straightforward to cut a problem in easier subproblem... but it is not the matter. Debugging the *big_constraint* has no mean in our framework: constraint predicates are assumed to be correct. Concerning this program, there exists a symptom because the programmer made an error when he wrote the *big_constraint*. It is certainly possible to say more than $go(\tilde{x}) \leftarrow big\_constraint$ is incorrect. The aim is to elaborate appropriate formal ways based on a semantics of the variables of the constraint.

The problem of wrong answers is easier than the problem of missing ones. Indeed, in CLP a declarative answer constraint is not covered by a single computed answer constraint. Thus, for insufficiency diagnosis, a computed answer cannot be considered alone. We have to consider the set (or a subset) of computed answers. We cannot limit exploration to a single skeleton. An error is rather a non completely covered constrained atom [3] than a non covered constrained atoms [4] as shown in [9].

References

[1] Comini, M., Levi, G., and Vitiello, G., Declarative Diagnosis Revisited. In John Lloyd, editor, *International Logic Programming Symposium*, pages 275–287. MIT Press, 1995.

[2] Deransart, P., Małuszyński, J., *A Grammatical View of Logic Programming*. MIT Press, 1993.

[3] Drabent, W., Nadjm-Tehrani, S., and Małuszyński, J., Algorithmic Debugging with Assertions. In *Meta-Programming in Logic Programming*, pages 501–522. MIT Press, 1989.

[4] Ferrand, G., Error Diagnosis in Logic Programming: an adaptation of E. Y. Shapiro's method. *Journal of Logic Programming*, 4:177–198, 1987.

[5] Gabbrielli, M., and Levi, G., Modeling answer constraints in Constraint Logic Programs. In *International Conference on Logic Programming*, pages 238–252. MIT Press, 1991.

[6] Giacobazzi, R., Debray, S. K., and Levi, G., Generalized Semantics and Abstract Interpretation for Constraint Logic Programs. *Journal of Logic Programming*, 25(3):191–248, 1995.

[7] Jaffar, J., and Maher, M. J., Constraint Logic Programming: a survey. *Journal of Logic Programming*, 19-20:503–581, 1994.

[8] Shapiro, E. Y., *Algorithmic Program Debugging*. MIT Press, 1982.

[9] Tessier, A., Declarative Debugging in Constraint Logic Programming: the Cover Relation. Technical Report 96/09, LIFO, University of Orléans, 1996.