Integration of abductive reasoning and constraint optimization in SCIFF*

Marco Gavanelli*, Marco Alberti†, and Evelina Lamma*

*ENDIF, University of Ferrara,
†CENTRIA, Universidade Nova de Lisboa

Abstract. Abductive Logic Programming (ALP) and Constraint Logic Programming (CLP) share the feature to constrain the set of possible solutions to a program via integrity or CLP constraints. These two frameworks have been merged in works by various authors, who developed efficient abductive proof-procedures empowered with constraint satisfaction techniques. However, while almost all CLP languages provide optimization meta-predicates for finding an optimal solution with respect to some objective function (and not just *any* solution), the issue has received little attention in ALP.
In this paper we show how optimisation meta-predicates can be included in abductive proof-procedures, achieving in this way a significant improvement to research and practical applications of abductive reasoning.
In the paper, we give the declarative and operational semantics of an abductive proof-procedure that encloses constraint optimization meta-predicates, and we prove soundness in the three-valued completion semantics. In the proof-procedure, the abductive logic program can invoke optimisation meta-predicates, which can invoke abductive predicates, in a recursive way.

1 Introduction

Abductive Logic Programming (ALP) [1] is a set of programming languages deriving from Logic Programming. In an abductive logic program, a distinguished set of predicates, called *abducibles*, do not have a definition, but their truth value can be *assumed*. A set of formulae, called *Integrity Constraints* (IC, often in the form of implications) restrict the set of hypotheses that can be made, in order to avoid unrealistic hypotheses.

ALP is interesting as it supports hypothetical reasoning, and in the context of logic programming it supports a simple, sound implementation of negation by failure [2] (also called, in the context of ALP, negation by default). Operationally, various abductive proof-procedures have been proposed in the past, and they have recently gained significant efficiency [3–10].

One important milestone in the research on abductive proof procedures was the integration with constraint logic programming (CLP). After the first works that showed that ALP and CLP could be cast under a common umbrella [11], many abductive proof-procedures have been integrated with CLP [5, 7, 9]. However, while in CLP optimization meta-predicates are considered vital in order to address many real-life problems,

* This article appeared as a full paper in the proceedings of the 25th International Conference on Logic Programming (ICLP 2009).

the issue has received little attention in the ALP community. As a matter of fact, the user is not always satisfied in finding one solution, but among the many solutions that satisfy the constraints, (s)he wants to find the *optimal* one with respect to some objective function.

Indeed, some works introduce ad-hoc optimizations into ALP. A common approach is to assign costs to abduced literals [10]. But, as CLP languages support a full integration of optimization meta-predicates (including recursion on such meta-predicates), one would expect such integration also into constrained ALP.

Notably, van Nuffelen and Denecker [12] performed interesting experiments with ALP and aggregates (optimization is, in fact, an instance of aggregate meta-predicate), but, to the best of our knowledge, there is no proof of correctness for their procedure with aggregates. On the other hand the importance of aggregates in ALP is well motivated in [12], as the ALP solution is compared with a pure CLP solution: *"the CLP solution is a long program (400 lines) developed in some weeks of time [. . .], where as the above representation is a simple declarative representation of 11 logical formulae, written down after some hours of discussion."* The possibility of reducing the development time from weeks to hours is very attractive. On the other hand, a logic programming language without a convincing declarative semantics and a proof of soundness is incomplete.

In this paper, we define a declarative semantics for abduction with optimization meta-predicates. We first show that a naive integration of the ALP and CLP declarative semantics has a number of drawbacks, then we propose a new semantics. We propose an operational semantics, that extends the $\mathcal{S}$CIFF proof-procedure [13], and that recalls the branch-and-bound procedure used by CLP solvers. Thanks to the new declarative semantics, we are able to show that the new proof-procedure, called $\mathcal{S}$CIFF^{opt}, is sound with respect to the three-valued completion semantics, in practical cases (those without floundering). In particular, we are able to deal with recursion through the optimization predicates: the abductive proof-procedure can invoke optimization predicates, which in turn can perform abductive reasoning. Recursion through optimization lets us deal with problems of games, as we showed in a short version of this paper [14].

2 Preliminaries

$\mathcal{S}$CIFF is an abductive proof-procedure that follows the classical semantics of ALP with constraints. An ALP with constraints [5] is formally defined as a triple $\mathcal{P} \equiv \langle KB, \mathcal{A}, IC \rangle$, where KB is the knowledge base (a logic program), $\mathcal{A}$ is a distinguished set of predicates, called *abducibles*, and IC is a set of implications, called Integrity Constraints. With abuse of notation, we will use $\mathcal{A}$ also for the set of ground atoms built on the abducible predicates. Given a goal $\mathcal{G}$, the aim of abduction is to find an abductive answer, i.e., a pair (Δ, θ), where Δ is a set $\Delta \subseteq \mathcal{A}$ and θ is a substitution, such that

$$KB \cup \Delta \cup \mathcal{T} \models \mathcal{G}\theta \wedge IC \tag{1}$$

where $\mathcal{T}$ is the theory of constraints [15], and will be omitted for simplicity in the following. Although most of the results are general, in the examples we will use a constraint sort on finite domains (CLP(FD)). Abducibles are in **bold**.

The optimization meta-predicates are the main objective of this work. The following syntax will represent an atom with three arguments:

$$min(X : G) = V$$

meaning that we are looking for the solution to the goal G that gives the minimal value to variable X; such value is V. When the value of the optimal solution is not of interest, we adopt the simplified syntax $min(X : G)$ (one can think of this simplified syntax as if adding an unnamed variable, as in Prolog: $min(X : G) = _$). Of course, we have a symmetric meta-predicate max. We will use upper-case letters for variables and lower-case for predicates and constants (as in Prolog).

3 A naive Declarative Semantics

The first intuition of a declarative semantics for abduction with optimization is to start from the declarative semantics of abduction itself (Eq 1) which states that, given an abductive program $\mathcal{P} \equiv \langle KB, \mathcal{A}, IC \rangle$, one's goal is to find a set $\Delta \subseteq \mathcal{A}$ of abducibles that (together with the knowledge base KB) entails both the goal G and the integrity constraints IC. I.e., we ask ourselves if there exists such a set Δ.

In the simplest possible situation, the optimization meta-predicate occurs only in the goal, e.g., $G \equiv min(X : p(X)) = V$, meaning that we want to find the minimal value V for variable X such that predicate $p(X)$ is true. The temptation is to adopt the same idea used to give semantics to optimization predicates in CLP, namely to rewrite $min(X : p(X)) = V$ as $p(V) \wedge not(\exists Y p(Y) \wedge Y < V)$, i.e., V is indeed the minimum if p is true and there is no smaller value Y that makes p true.

Now, combining abduction and optimization we would obtain:

$$KB \cup \Delta \models p(V) \wedge not(p(Y) \wedge Y < V) \wedge IC \tag{2}$$

This formalisation is very intuitive, but it does not provide a semantics usable in practical situations. The meaning of equation 2 is *"Do there exist a set Δ and a value V such that $p(V)$ is true and no other value Y smaller than V makes p true?"* Let us now consider a very simple abductive program:

$$p(X) \leftarrow \mathbf{a}(X) \wedge 1 \leq X \leq 2. \tag{3}$$

without integrity constraints. In this case, the declarative semantics in Eq. 2 would provide the following answers to the goal $min(X : p(X)) = V$:

$$\begin{array}{ll} \Delta_1 = \{\mathbf{a}(1)\} & V = 1 \\ \Delta_2 = \{\mathbf{a}(2)\} & V = 2 \\ \Delta_3 = \{\mathbf{a}(1), \mathbf{a}(2)\} & V = 1. \end{array}$$

We find such an answer counter intuitive, in particular Δ_2, and not in the direction of real-life applications. In the semantics of Eq. 2, for each set Δ that supports $p(X)$ we have a positive answer; the value V is simply the minimum value amongst the abduced

literals. Classical applications of ALP are diagnosis, and planning; by combining abductive reasoning with optimization, one would expect to be able to answer to questions like *"What is the plan of minimal cost?"* or *"What is the explanation of maximal likelihood?"*, which means that the user wants to find *the optimal set* Δ, not that she wants to find any explanation Δ, and then take the minimal value that makes true a predicate with such assumptions. More formally, the intended meaning is not

$$(\exists \Delta, V) \qquad KB \cup \Delta \models p(V) \wedge not(\exists Y.p(Y) \wedge Y < V) \wedge IC$$

which means that given a set Δ that satisfies p and IC, V is the optimal value in that particular Δ, but that the set Δ should be in the scope of optimization, as in:

$$\begin{aligned}(\exists V, \Delta^*)\ [\quad & KB \cup \Delta^* \models p(V) \wedge IC \\ & \wedge\ not(\exists Y, \Delta'.Y < V \wedge KB \cup \Delta' \models p(Y) \wedge IC)]\end{aligned} \tag{4}$$

Of course, Eq 4 is meaningful only for optimization atoms that occur in the goal, but it does not give a semantics to general ALPs that contain optimization atoms in the KB or in the IC.

Starting from the practical need to combine abduction and constraint optimization, we propose a new declarative semantics for ALP with optimization. We ground our semantics on the $\mathcal{S}$CIFF language and proof-procedure [13], but we believe that our results could be easily extended to other proof-procedures.

4 Declarative Semantics

The declarative semantics is given, as usual, with respect to the ground program. When there are optimization literals, defining the grounding of a program is not immediate; we adopt the same definitions by Faber et al. [16], adapted to the $\mathcal{S}$CIFF syntax.

Definition 1. *A* Set Term *is either a symbolic or a ground set. A* Symbolic Set *is a pair* $\{V : Conj\}$*, where* V *is a variable and* $Conj$ *is a conjunction of atoms. A* Ground Set *is a set of pairs* $\langle t : Conj\rangle$*, where* t *is a numeric constant and* $Conj$ *is a ground conjunction of atoms.*

Definition 2. *An* Optimization Atom *is either of the form* $min(S) = V$ *or* $max(S) = V$*, where* S *is a set term.*

We will suppose for simplicity that optimization atoms occur only in the body of clauses, and not in Integrity Constraints.

Definition 3. *Given a clause, a* local variable *is a variable that occurs only in an optimization atom. All other variables are* global.

Definition 4. *Given a symbolic set without global variables* $S = \{V : Conj\}$*, the* instantiation of S *is a ground set of pairs* $\{\langle\gamma(V) : \gamma(Conj)\rangle | \gamma$ *is a substitution for the local variables in* $S\}$.

A Ground Instance *of a clause* r *is obtained in two steps:*

1. *all global variables are grounded*
2. *every symbolic set is replaced by its instantiation*

After defining the grounding of a program, we can give it semantics. We will restrict ourselves to *locally stratified programs*. Note that local stratification does not prevent the user to use recursion through optimization.

Definition 5. *A ground program is* locally stratified with respect to optimization *if there exists a level mapping* $||\cdot|| : \mathcal{H} \mapsto \mathbb{N}$ *(where* $\mathcal{H}$ *is the Herbrand Base) such that for each pair of ground atoms* h *and* b *occurring, respectively, in the head and in the body of a clause:*

- *if* b *occurs in the clause in an optimization atom, then* $||b|| < ||h||$
- *otherwise,* $||b|| \leq ||h||$.

If such a mapping exists, then the set of ground atoms in the Herbrand base is partitioned into levels. Suppose for simplicity that the levels take the values of the first natural numbers (so the first level takes value 0).

4.1 3-valued completion for non-abductive programs

We extend the 3-valued completion semantics [17, 18] to the case with optimization meta-predicates. A (partial) interpretation I is a set of literals considered true. A literal p is false in I iff $\neg p \in I$. If $\{p, \neg p\} \cap I = \emptyset$, then p's truth value is *unknown* ($\perp$).

We report the extension of the T_P operator to the three-valued case:

Definition 6. *Consider an atom* p *of a defined predicate*

- $p \in T_P(I)$ *iff there is some instantiated clause* $R \in P$ *such that* R *has head* p*, and each subgoal literal in the body of* R *is true in* I.
- $p \in U_P(I)$ *iff for all clauses* $R \in P$ *that have head* p*, the body of the clause is false in* I.
- $W_P(I) = T_P(I) \cup \neg \cdot U_P(I)$*, where* $\neg \cdot U_P(I)$ *means the negation of all atoms in* $U_P(I)$ *(i.e., if atom* $a \in U_P(I)$*, then* $\neg a \in W_P(I)$*).*

Notice that Definition 6 gives a truth value only to literals defined in KB, other literals (abducibles, optimization atoms) have still unknown $\perp$ truth value.

T_P, U_P and W_P are monotonic transformations (i.e., $T_P(I) \subseteq T_P(J)$ whenever $I \subseteq J$), so considering the limit makes sense. For a transformation Φ, let [17]

- $\Phi \uparrow^0 (S) = S$,
- $\Phi \uparrow^{\alpha+1} (S) = \Phi(\Phi \uparrow^{\alpha} (S))$
- for limit ordinals λ, $\Phi \uparrow^{\lambda} (S) = \bigcup_{\alpha<\lambda} \Phi \uparrow^{\alpha} (S)$

Let $I_0 = \emptyset$, $I_\alpha = W_P \uparrow^{\alpha} (\emptyset)$, $I_\infty = W_P \uparrow^{\omega} (\emptyset)$, where ω is the first limit ordinal.

4.2 3-valued completion semantics for abductive programs

Since $\mathcal{P}$ is an ALP, the truth of an atom depends on the assumed hypotheses. We consider, in the declarative semantics, all the possible groundings of abducible literals that satisfy the integrity constraints. Let $I_0(\Delta)$ the 3-valued interpretation corresponding to the set of abduced atoms Δ, i.e., $\forall a \in \Delta$, $a \in I_0(\Delta)$ and $\forall a \in \mathcal{A} \setminus \Delta$, $\neg a \in I_0(\Delta)$.

Let $I_\infty(\Delta) = W_P \uparrow^\omega (I_0(\Delta))$. As stated earlier, $I_\infty(\Delta)$ assigns value $\perp$ to all optimization atoms. Let us suppose that the program is stratified also with respect to negation [19]; in such a case, the three-valued completion semantics gives values true-false to each atom (never unknown), so the only unknown atoms are the optimization atoms and the atoms that depend on them.

If the program is locally stratified with respect to optimization, there will be an optimization atom $min(S) = V_m$ such that $\forall \langle V : C \rangle \in S$, C is not $\perp$ (i.e., there will be an optimization atom of minimum level).

We now define a new operator that gives semantics to the optimization atoms.

4.3 Extension of 3-valued T_P for optimization atoms

Before defining the extension of the T_P operator to the optimization atoms, consider the following example, in CLP (without abduction):

$$min(X : max(Y : p(X, Y))) = V$$

Intuitively, this problem can be thought as a two player game: given that the possible solutions are those that satisfy predicate $p(X, Y)$, the first player tries to minimize X while the second maximizes Y. The point here is that the first player has control over the variable X, while the second instantiates variable Y: player 2 cannot choose the value of variable X. This means that we need to exclude some variables from the maximization; for this reason, various authors [20, 21] proposed to extend the syntax, in order to let the user choose which variables are subject to optimization and which are not, by providing *protected variables* [21] to the optimization meta-predicate.

In ALP, we have the same issue, and a further one: which of the two nested atoms is responsible for grounding the set Δ? We would like both invocations to be able to abduce literals, otherwise the expressivity of our language would be strongly compromised: in fact, we would boil down to a two step procedure, and lose the possibility of recursion through optimization predicates. We decided to explicitly communicate to the optimization atom those *literals* it is responsible to abduce. We show in the following of this section that there is a precise declarative semantics.

We extend the syntax of the optimization meta predicate:

$$min_{\mathcal{A}_m}(X : p(X)) = V \tag{5}$$

the intuitive meaning is that we are looking for the minimum value for X such that $p(X)$ is true, knowing that in such minimization we are entitled to abduce only the literals occurring in the set $\mathcal{A}_m \subseteq \mathcal{A}$. Since the set $\mathcal{A}_m$ could be infinite, we sometimes represent its content with non-ground atoms, meaning that all possible groundings belong to $\mathcal{A}_m$.

We can now define precisely the declarative semantics of the optimization meta-predicate in the 3-valued completion semantics, by extending the T_P operator:

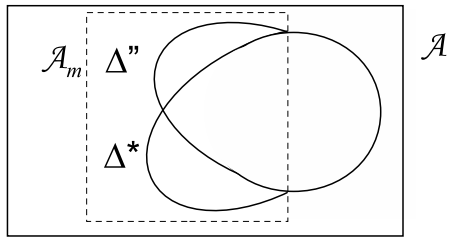

Fig. 1. Relation of sets in Definition 7

Definition 7. *A non-optimization literal $l \in M_P(I)$ iff $l \in W_P(I)$.*

Otherwise, consider the set of all the possible groundings of the abducible literals that satisfies the integrity constraints: $Cons = \{\Delta \in 2^{\mathcal{A}} | KB \cup \Delta \models IC\}$. Consider one specific set Δ^* (intuitively, the candidate set of abduced literals).

The atom $min_{\mathcal{A}_m}(S) = V_m$ is true, i.e., $min_{\mathcal{A}_m}(S) = V_m \in M_P(I(\Delta^*))$, iff all the following conditions hold:

1. *$\forall \Delta \in Cons$, $\forall \langle V : C \rangle \in S$, either $I(\Delta) \models C$ or $I(\Delta) \models \neg C$ (intuitively, C is not $\perp$ in all the possible Δ)*
2. *there exists some $\langle V : C \rangle \in S$ such that $V = V_m$ and*
   - *(a) $I(\Delta^*) \models C$*
   - *(b) $\nexists \langle V' : C' \rangle \in S$ s.t. $I(\Delta^*) \models C'$ and $V' < V$*
   - *(c) $\nexists (\langle V'' : C'' \rangle \in S$ and $\Delta'' \in Cons)$ such that*
     - *i. $\Delta'' \cap (\mathcal{A} \setminus \Delta^*) \subseteq \mathcal{A}_m$*
     - *ii. $\Delta'' \cap (\mathcal{A} \setminus \mathcal{A}_m) = \Delta^* \cap (\mathcal{A} \setminus \mathcal{A}_m)$*
     - *iii. $I(\Delta'') \models C''$ and $V'' < V$*

The atom $min_{\mathcal{A}_m}(S) = V_m$ is false, i.e., $\neg(min_{\mathcal{A}_m}(S) = V_m) \in M_P(I(\Delta^*))$ iff condition 1 holds, but at least one of the other conditions does not hold.

Intuitively, condition 2b imposes that no better solution exists in the same Δ^*, while condition 2c requires that a better solution does not exist in a different set Δ''. Such set Δ'' cannot be completely different from Δ^*, as in this specific optimization we are supposed to optimize only with respect to $\mathcal{A}_m$, and not with respect to the whole set of abducibles $\mathcal{A}$. In the current optimization, we will only abduce literals in $\mathcal{A}_m$, while the other literals (in $\mathcal{A} \setminus \mathcal{A}_m$) can be abduced externally. For this reason, Δ'' coincides with Δ^* for the part in $\mathcal{A} \setminus \mathcal{A}_m$, and differs only for the part in $\mathcal{A}_m$ (Figure 1).

The operator M_P is monotonic, and gives a truth value to optimization predicates that contain only conditions whose truth value is known. In other words, if one knows the truth value of the argument of min, he can define the truth of min through the M_P operator. Otherwise, the min literal remains unknown.[1] Note that we require such knowledge for all the possible Δ that satisfy the integrity constraints (set $Cons$).

[1] This condition could be relaxed; e.g., $min(\langle 1, p \rangle) = 2$ is obviously false even if we do not know the truth of atom p.

Given a set Δ, we can apply the operator M_P up to its fix-point; if a ground literal $a \in M_P \uparrow^{\omega} (I_0(\Delta))$, we write

$$\mathcal{P} \models^{opt}_{\Delta} a.$$

5 $\mathcal{S}$CIFF^{opt} operational semantics

The $\mathcal{S}$CIFF proof-procedure consists of a set of transitions that rewrite a node into one or more children nodes. It encloses the transitions of the IFF proof-procedure [6], and extends it in various directions. We recall the basics of $\mathcal{S}$CIFF; a complete description is in [13], with proofs of soundness, completeness, and termination.

Each node of the proof is a tuple $T \equiv \langle R, CS, PSIC, \Delta \rangle$, where R is the resolvent, CS is the CLP constraint store, PSIC is a set of implications derived from propagation of integrity constraints, and Δ is the current set of abduced literals. The main transitions, inherited from the IFF are:

Unfolding replaces a (non abducible) atom with its definitions;
Propagation if an abduced atom $a(X)$ occurs in the condition of an IC (e.g., $a(Y) \rightarrow p$), the atom is removed from the condition (generating $X = Y \rightarrow p$);
Case Analysis given an implication containing an equality in the condition (e.g., $X = Y \rightarrow p$), generates two children in logical or (in the example, either $X = Y$ and p, or $X \neq Y$);
Equality rewriting rewrites equalities as in the Clark's equality theory;
Logic simplifications other simplifications like $true \rightarrow A \Leftrightarrow A$, etc.

$\mathcal{S}$CIFF includes also the transitions of CLP [15] for constraint solving.

We introduce a new transition to process optimisation meta-predicates.

In order to simplify the exposition, we assume that the goal argument of min constrains the value of the objective function to become ground in each leaf node, as in many practical implementations of CLP (see, e.g., SICStus manual). We plan to remove such assumption in future work, by considering *bounds* as in the operational semantics of optimization in CLP [21].

Definition 8 (Transition Optimize). *Given a node*

$$T' \equiv \langle R', CS', PSIC', \Delta' \rangle,$$

such that the resolvent R' contains exactly an optimization literal, i.e.,

$$R' = \{min_{\mathcal{A}_m}(F : G) = V\},$$

transition Optimize *opens a new $\mathcal{S}$CIFF derivation tree with starting node*

$$T^{opt} \equiv \langle G, CS', PSIC', \Delta' \rangle.$$

We call the derivations spawning from node T^{opt} sub-derivations. *If all sub-derivations finitely fail, then the successor of T' is the special node $false$.*

Otherwise, let S the set of leaf nodes of the sub-derivations starting from node T^{opt}. For each leaf node $N_j \in S$ (Figure 2) one can compute the value of F for the node.

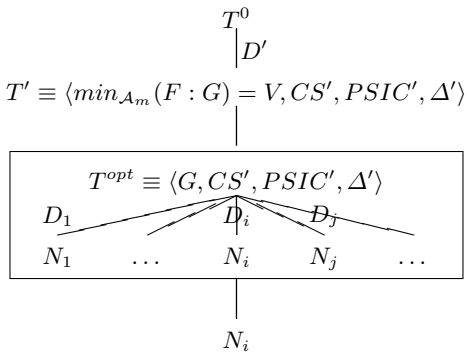

Fig. 2. $\mathcal{S}$CIFF^{opt} derivation.

As in CLP, we can call F the objective function *and write, with an abuse of notation, $F(N_j)$ to indicate its value in a node.*

If there exists a node $N_k \in S$ in which the set of abduced literals Δ_k contains new literals not included in $\mathcal{A}_m$ (i.e., $\Delta_k \not\subseteq \mathcal{A}_m \cup \Delta'$), the derivation flounders.

Note also that transition Optimize *is not applicable if R' contains more than one literal. In case no transitions are applicable and in the last node the resolvent is not empty, the derivation flounders (this can happen, e.g., in case the goal contains a conjunction of minimization literals).*

If there is a successful sub-derivation D_i starting from node T^{opt} with final node $N_i \in S$, and value $F^* \equiv F(N_i)$, then to the constraint store of each node $N_j \in S$ a new constraint $F(N_j) \leq F^*$ is added. To such nodes, constraint propagation is applied, and it can possibly fail. In case of success in some node N_k, again the new value $F_2^* \equiv F(N_k)$ is computed, and the new constraint $F(N_l) \leq F_2^*$ is added to the store of all the remaining nodes N_l. This process continues until the fix point. The final nodes of the successful sub-derivations are then generated as children of the node T'.

If the $\mathcal{S}$CIFF^{opt} operational semantics (including transition *Optimize*) has a successful (non floundering) derivation with abductive answer (Δ, σ) for a goal G and a program $\mathcal{P}$ we write

$$\mathcal{P} \vdash^{opt}_{\Delta\sigma} G\sigma.$$

Example 1. Consider the program in Eq. 3, with the goal $min_{\{\mathbf{a}(_)\}}(X : p(X)) = V$. The only applicable transition is *Optimize*, which opens a new derivation for the goal $p(X)$. The $\mathcal{S}$CIFF proof-procedure has two possible derivations, with nodes (we report for simplicity only the sets of abduced literals): $\Delta_1 = \{\mathbf{a}(1)\}$, $\Delta_2 = \{\mathbf{a}(2)\}$. *Optimize* chooses one of them: suppose Δ_2. $F(\Delta_2) = 2$, so the new constraint $X \leq 2$ is added to all nodes; constraint propagation does not exclude any value nor causes failure. Another node is selected: Δ_1. $F(\Delta_1) = 1$, and the constraint $X \leq 1$ is added to all nodes. This causes a failure in Δ_2, since $2 \not\leq 1$, so the only viable solution is the node containing Δ_1. Such node is reported as child of the initial goal, with $V = 1$. No other transition is applicable, so we have success with $V = 1$

The optimization predicates are postponed after the others, because they can be safely applied at the end of a derivation. This is not an issue in many practical applications: for example, when solving constrained optimization problems in CLP, one first imposes constraints and then performs a search. Of course, in some cases postponing the optimization goal can lower efficiency. Another way to deal with such problem is using protected variables [21]; in this paper we do not use them to simplify the exposition, and leave for future work the integration of protected variables into our syntax and semantics.

Note that this limitation does not prevent recursion through optimization: when transition *Optimize* is applied, a new derivation starts, and optimisation can be applied inside it (it is a different derivation).

Example 2. Consider the goal $G_1 \equiv min_{\{\mathbf{a}(_)\}}\big(X : min_{\{\mathbf{a}(_)\}}(Y : p(X,Y))\big)$, with:

$$p(X,Y) \leftarrow 0 < X < Y, \mathbf{a}(X,Y).$$

Start D^1**:** Transition *Optimize* is applicable, and a new derivation tree is generated, rooted in $G_2 \equiv min_{\{\mathbf{a}(_)\}}(Y : p(X,Y))$.

Start D^2**:** *Optimize* is applicable in the new derivation, and generates a new tree rooted in $G_3 \equiv p(X,Y)$.

Start D^3**:** G_3 is solved with the $\mathcal{S}$CIFF transitions, that in particular abduce $\mathbf{a}(X,Y)$. The best solution in this subtree is node $Y = 1$, $X = 0$, $\mathbf{a}(0,1)$.

End D^3**:** Derivation D^3 terminates, so there is no floundering in it.

End D^2**:** No transitions after *Optimize*, so no floundering.

End D^1**:** no floundering.

We are not currently dealing with the case in which there is an optimisation sub-goal in the condition of an integrity constraint. We leave this issue for future research; if at the end of a derivation there is an implication with a minimization sub-goal in the condition, the derivation *flounders*.

6 Soundness

We will rely on the following theorems, proved in [13]:

Theorem 1 (Soundness of $\mathcal{S}$CIFF). *Given an abductive logic program $\mathcal{P}$, if $\mathcal{P} \vdash_{\Delta} \mathcal{G}$ with abductive answer (Δ, σ), then $\mathcal{P} \models_{\Delta\sigma} \mathcal{G}\sigma$*

Theorem 2 (Completeness of $\mathcal{S}$CIFF). *Given an abductive logic program $\mathcal{P}$, a (ground) goal G, for any ground set Δ such that $\mathcal{P} \models_{\Delta} \mathcal{G}$ then $\exists\Delta'$ such that $\mathcal{P} \vdash_{\Delta'} G$ with an abductive answer (Δ', σ) such that $\Delta'\sigma \subseteq \Delta$.*

We can now give the main result

Theorem 3 (Soundness of $\mathcal{S}$CIFF^{opt}). *Given an abductive logic program $\mathcal{P}$ with optimization predicates, that is locally stratified both with respect to negation and to optimization, the following results hold:*

1. **(Soundness of success)** *if*

$$\mathcal{P} \vdash^{opt}_{\Delta} \mathcal{G}$$

with abductive answer (Δ, σ)*, then*

$$\mathcal{P} \models^{opt}_{\Delta\sigma} \mathcal{G}\sigma$$

2. **(Soundness of failure)** *if the* $\mathcal{S}CIFF^{opt}$ *derivation for a goal G finitely fails, then*

$$\mathcal{P} \not\models^{opt} \mathcal{G}$$

Proof. Suppose that the derivation does not contain applications of the *Optimize* transition. In this case, the thesis follows immediately from the soundness and completeness of the $\mathcal{S}$CIFF proof-procedure (Theorems 1 and 2).

Note that a non-floundering $\mathcal{S}$CIFF^{opt} derivation contains at most one application of the *Optimize* transition, otherwise the second application would make the derivation flounder. However, each $\mathcal{S}$CIFF^{opt} derivation can generate new sub-derivations (each possibly containing an application of *Optimize*). We call *full derivation* the forest of derivations triggered by a goal, including all sub-derivations for optimisation sub-goals.

By induction, suppose that all $\mathcal{S}$CIFF^{opt} full derivations of depth n (i.e., we have n levels of application of the *Optimize* transition) are sound (satisfy conditions 1 and 2). Consider a $\mathcal{S}$CIFF^{opt} derivation D that generates sub-derivations of depth up to n. The sub-derivations are sound by inductive hypothesis. The derivation D consists of a $\mathcal{S}$CIFF derivation D' followed by the application of the *Optimize* transition (Figure 2). Derivation D' is a $\mathcal{S}$CIFF derivation, that is sound and complete [13]. The *Optimize* transition is applied only to a node with a min goal, that must be the only element in the resolvent (otherwise other transitions would be applicable after *Optimize*). Let $T' \equiv \langle min_{\mathcal{A}_m}(F : G) = V, CS', PSIC', \Delta' \rangle$ the final node of D' (Figure 2).

Consider a sub-derivation D_i (with goal G); let N_i the final node of D_i. D_i is sound by inductive hypothesis.

Suppose that transition *Optimize* generates node N_i as successor of T'; we prove that all the conditions in Definition 7 hold.

Condition 1 holds because the program is stratified with respect to both negation and optimization. Since the program is stratified with respect to negation, the tree-valued completion semantics has a unique model, and no literal has unknown truth value.

Condition 2a requires the derivation D_i to be sound: this holds because of the inductive hypothesis.

Condition 2b requires that there is no other substitution θ' that, together with the same set of hypotheses Δ^*, provides a better value for V. In fact, if there existed a substitution θ' supporting a value $V' < V$, then (Δ^*, θ') would be an abductive answer to the goal $G \wedge F < V$ (i.e., $\mathcal{P} \models_{\Delta^* \ \theta'} (G \wedge F < V)\theta'$). But the goal $G \wedge F \leq V$ is the initial node of another sub-derivation, call it D_j. If D_j succeeded with a value $V' < V$, then transition *Optimize* would have added the constraint $F(N_i) \leq V'$ to the node N_i, which would have failed (contradicting the hypothesis that N_i is the successor of T'). If D_j failed, then, since for inductive hypothesis the soundness of failure holds for the sub-derivations, there is no abductive answer that supports the goal $G \wedge F \leq V$. Otherwise, D' may succeed with $V' = V$, but this does not contradict the assumption that V is one of the optima.

Condition 2c holds again due to the soundness of failure. It requires that there is no other substitution θ'' that, together with a different set of hypotheses Δ'' provides a better value V''. Moreover, the candidate set Δ'' can differ from Δ^* only for the literals in $\mathcal{A}_m$ (see Definition 7).

Suppose, by contradiction, that there is a set Δ'' satisfying the conditions 2c ($i-iii$). Since $V'' < V$, (Δ'', θ'') is an abductive answer to the goal $G \wedge F \leq V$, that is the initial node of another derivation D^{over}. If D^{over} succeeds with a value $V'' < V$, then transition *Optimize* would not return the value V: it would impose the constraint $F(N_i) \leq V''$ to the node N_i, which would obviously fail (contradicting a previous hypothesis). If D^{over} fails, this failure would be unsound, contradicting the inductive hypothesis. If D^{over} succeeds with $V'' = V$, it contradicts the assumption that $V'' < V$.

7 Example

Consider a two player game, where each of the players A and B can play one move. The result of the two moves is a configuration with an associated value: one player's aim is to maximize the value, the other player's is to minimize it. Player A's move is represented by the abducible $\mathbf{a}(M_a, X)$, where M_a is the possible move and X is the obtained value. Analogously, player B abduces $\mathbf{b}(M_b, X)$. The obtained value is defined with a predicate $f(M_a, M_b, X)$ that gives the obtained value X corresponding to moves M_a and M_b. It can be defined (Figure 3) as a set of facts $f(0, 0, 5)$, $f(0, 1, 10)$, $f(1, 0, 4)$, $f(1, 1, 3)$. To compute the obtained value, we can define a predicate or an IC as the following:

$$\mathbf{a}(M_a, X_a), \mathbf{b}(M_b, X_b) \rightarrow X_a = X_b, f(M_a, M_b, X_a). \tag{6}$$

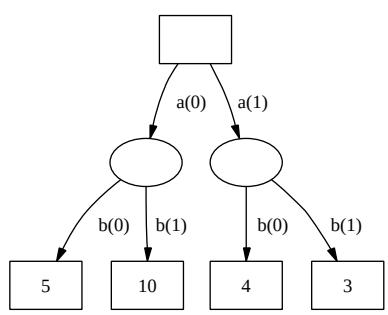

Fig. 3. min-max

As player A moves first, and wants to maximize the value X, while player B moves next and his goal is to minimize X, the $\mathcal{S}$CIFF^{opt} goal will be

$$max_{\{\mathbf{a}(_,_),\mathbf{b}(_,_)\}}(V_b : \mathbf{a}(M_a, X_a) \wedge (M_a = 0 \vee M_a = 1) \wedge \\ min_{\{\mathbf{b}(_,_)\}}(X_b : \mathbf{b}(M_b, X_b) \wedge (M_b = 0 \vee M_b = 1)) = V_b)$$

We have four possible sets Δ satisfying the integrity constraint: $\Delta_0^0 = \{\mathbf{a}(0,5), \mathbf{b}(0,5)\}$, $\Delta_1^0 = \{\mathbf{a}(0,10), \mathbf{b}(1,10)\}$, $\Delta_0^1 = \{\mathbf{a}(1,4), \mathbf{b}(0,4)\}$, and $\Delta_1^1 = \{\mathbf{a}(1,3), \mathbf{b}(1,3)\}$. Declaratively, the internal goal $min_{\{\mathbf{b}(_,_)\}}$ is true in Δ_0^0 and Δ_1^1: Δ_0^1 is ruled out by Δ_0^0 (see condition 2c in Definition 7) and Δ_1^0 by Δ_1^1. The external $max_{\{\mathbf{a}(_,_),\mathbf{b}(_,_)\}}$ goal, thus, chooses from these two sets the Δ with maximum value of X_b, namely Δ_0^0; this output is the same as a min-max algorithm.

From an operational viewpoint, transition *Optimize* generates two nodes: one in which abduces $\mathbf{a}(0, X_a)$, and one with $\mathbf{a}(1, X_a)$.

$M_a = 0$ Transition *Optimize* is applied to min: it opens two nodes, one abducing $\mathbf{b}(0, X_b)$, the other with $\mathbf{b}(1, X_b)$.

$M_b = 0$ In the first, propagation of the integrity constraint (Eq. 6) imposes $X_a = X_b = 5$. Now transition *Optimize* of the internal min imposes the constraint $X_b \leq 5$ to all the open nodes in its scope, i.e., the node with value 10 in Figure 3

$M_b = 1$ In the second node, the propagation of the IC imposes $X_a = X_b = 10$, which conflicts with the constraint $X_b \leq 5$; CLP propagation results in a failure.

Now *Optimize* applied to min provides value 5 as optimum, and generates the node with Δ_0^0 as successor. The external *Optimize* (applied to max) adds the new constraint $X_a \geq 5$ to all open nodes, in particular to the open choice point.

$M_a = 1$ In this node, the external *Optimize* has imposed $V_b \geq 5$. Again, transition *Optimize* is applied to the min literal, and it opens two nodes. The minimum value computed for V_b is 3, and it does not satisfy $V_b \geq 5$, so the result is indeed $X_a = X_b = 5$.

This example shows how min-max problems can be easily encoded in $\mathcal{S}$CIFF^{opt}. In this simple example, we impose the optimization directly in the goal for ease of presentation, but it can be simply extended to other example with recursion through minimization, to solve problems in PSPACE. In [14] we showed one such example.

8 Related work

Since abduction is one of the basic inferences in human reasoning [22], countless works address it in many different contexts. In many works, abduction is associated to optimization in the sense that a minimal number of hypotheses should be assumed, or that there is a cost associated to each assumed literal. The work [23] addresses the problem in the propositional case, with a limited syntax for integrity constraints (that can contain only abducible literals), and without CLP constraints; the authors use neural networks, that are an incomplete method, meaning that they cannot prove optimality.

This case can be easily cast in $\mathcal{S}$CIFF^{opt}, by transforming the original program. For each abducible literal $\mathbf{a}$ we have another abducible $\mathbf{c_a}(C_a)$, with the integrity constraint

$$\mathbf{a}, \mathbf{c_a}(C_a) \rightarrow C_a = \gamma_a \tag{7}$$

where γ_a is a real number representing the cost of assuming $\mathbf{a}$. Then, we modify the goal $\mathcal{G}$ as follows: the new goal $\mathcal{G}'$ abduces all the new $\mathbf{c_x}$ literals, where $\mathbf{x}$ is a literal in the original program; then minimizes the sum of the costs:

$$\mathcal{G}' = min((\mathcal{G}, \mathbf{c_a}(C_a), \mathbf{c_b}(C_b), \mathbf{c_c}(C_c), \ldots, F = C_a + C_b + C_c + \ldots), F)$$

The same problem has been addressed also in the context of the stable model semantics [10], for normal logic programs without function symbols. The non-propositional case without function symbols can be addressed in $\mathcal{S}$CIFF^{opt} by extending the encoding in Eq. (7).

9 Conclusions

Integration of abductive reasoning and constraint satisfaction has been vastly investigated in the recent years, and efficient proof-procedures have been developed [5, 7, 9]. Surprisingly, constraint optimization, one of the main topics in Constraint Programming, has been often left out of abductive proof-procedures, except for the interesting experiments reported (without proofs) in [12].

We extended the declarative and operational semantics of the $\mathcal{S}$CIFF proof-procedure to support this type of reasoning, resulting in the $\mathcal{S}$CIFF^{opt} framework. For $\mathcal{S}$CIFF^{opt} we proved a soundness result, that, to the best of our knowledge, is the first in the literature on abductive logic programming with constraint optimization. The soundness result holds when there is no floundering, a common issue in many logic programming languages. In future work, we plan to study the floundering issue in more detail, and extend the applicability of $\mathcal{S}$CIFF^{opt} to other problems, like those in which a conjunction of optimization atoms is required.

References

1. Kakas, A.C., Kowalski, R.A., Toni, F.: The role of abduction in logic programming. In Gabbay, D.M., Hogger, C.J., Robinson, J.A., eds.: Handbook of Logic in Artificial Intelligence and Logic Programming. Volume 5., Oxford University Press (1998) 235–324
2. Eshghi, K., Kowalski, R.A.: Abduction compared with negation by failure. In Levi, G., Martelli, M., eds.: ICLP'89. (1989) 234–255
3. Kakas, A.C., Mancarella, P.: On the relation between Truth Maintenance and Abduction. In Fukumura, T., ed.: Proc. 1st Pacific Rim Int. Conf. on Artificial Intelligence, PRICAI. (1990)
4. Denecker, M., De Schreye, D.: SLDNFA: an abductive procedure for abductive logic programs. Journal of Logic Programming **34** (1998) 111–167
5. Kakas, A.C., Michael, A., Mourlas, C.: ACLP: Abductive Constraint Logic Programming. Journal of Logic Programming **44** (2000) 129–177
6. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming **33** (1997) 151–165
7. Kakas, A.C., van Nuffelen, B., Denecker, M.: $\mathcal{A}$-System: Problem solving through abduction. In Nebel, B., ed.: Proc. of IJCAI-01. (2001) 591–596
8. Alferes, J., Pereira, L., Swift, T.: Abduction in well-founded semantics and generalized stable models via tabled dual programs. Theory and Practice of Logic Programming **4** (2004)
9. Endriss, U., Mancarella, P., Sadri, F., Terreni, G., Toni, F.: The CIFF proof procedure for abductive logic programming with constraints. In Alferes, J.J., Leite, J.A., eds.: Logics in Artificial Intelligence, 9th European Conference, JELIA 2004. (2004)
10. Perri, S., Scarcello, F., Leone, N.: Abductive logic programs with penalization: Semantics, complexity and implementation. TPLP **5** (2005) 123–159
11. Kowalski, R., Toni, F., Wetzel, G.: Executing suspended logic programs. Fundamenta Informaticae **34** (1998) 203–224

12. van Nuffelen, B., Denecker, M.: Problem solving in ID-logic with aggregates. In: Proc. 8th Int. Workshop on Non-Monotonic Reasoning, NMR'00. (2000) 1–9
13. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The SCIFF framework. ACM Transactions on Computational Logic **9** (2008)
14. Gavanelli, M., Alberti, M., Lamma, E.: Integrating abduction and constraint optimization in constraint handling rules. In Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N., eds.: ECAI 2008: 18th European Conference on Artificial Intelligence. (2008) 903–904
15. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming **19-20** (1994) 503–582
16. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In Alferes, J., Leite, J., eds.: JELIA 2004, Springer Verlag (2004)
17. Fitting, M.: A Kripke-Kleene semantics for logic programs. Journal of Logic Programming **2** (1985) 295–312
18. Kunen, K.: Negation in logic programming. In: Journal of Logic Programming. Volume 4. (1987) 289–308
19. Apt, K.R., Bol, R.N.: Logic programming and negation: a survey. Journal of Logic Programming **19/20** (1994) 9–71
20. Fages, F.: From constraint minimization to goal optimization in CLP languages. In Freuder, E., ed.: CP'96. Volume 1118 of LNCS. (1996)
21. Marriott, K., Stuckey, P.: Semantics of constraint logic programs with optimization. In Aït-Kaci, H., Hanus, M., Moreno-Navarro, J., eds.: ICLP Workshop: Integration of Declarative Paradigms. (1994) 23–35
22. Hartshorne, C., Weiss, P., eds.: Collected Papers of Charles Sanders Peirce, 1931–1958. Volume 2. Harvards University Press (1965)
23. Abdelbar, A., Andrews, E., Wunsch II, D.: Abductive reasoning with recurrent neural networks. Neural Networks **16** (2003) 665–673