The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments

Mancarella, P.; Terreni, G.; Sadri, F.; Toni, F.; Endriss, U.

DOI
10.1017/S1471068409990093

Publication date
2009

Document Version
Final published version

Published in
Theory and Practice of Logic Programming

TLP **9** (6): 691–750, 2009.
doi:10.1017/S1471068409990093 First published online 14 August 2009

The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments

PAOLO MANCARELLA and GIACOMO TERRENI
Dipartimento di Informatica, Università di Pisa, Pisa, Tuscany, Italy
(*e-mail:* paolo.mancarella@unipi.it, terreni@di.unipi.it)

FARIBA SADRI and FRANCESCA TONI
Department of Computing, Imperial College London, London, UK
(*e-mail:* {fs,ft}@doc.ic.ac.uk)

ULLE ENDRISS
Institute for Logic, Language and Computation (ILLC), University of Amsterdam, Amsterdam, The Netherlands
(*e-mail:* ulle.endriss@uva.nl)

submitted 23 January 2008; revised 23 December 2008; accepted 22 April 2009

Abstract

We present the CIFF proof procedure for abductive logic programming with constraints, and we prove its correctness. CIFF is an extension of the IFF proof procedure for abductive logic programming, relaxing the original restrictions over variable quantification (*allowedness conditions*) and incorporating a constraint solver to deal with numerical constraints as in constraint logic programming. Finally, we describe the CIFF system, comparing it with state-of-the-art abductive systems and answer set solvers and showing how to use it to program some applications.

KEYWORDS: abduction, constraints, proof procedures

1 Introduction

Abduction has found broad application as a powerful tool for hypothetical reasoning with incomplete knowledge. This form of reasoning is handled by labeling some pieces of information as abducibles, i.e., as possible hypotheses, that can be assumed to hold, provided that they are consistent with the rest of the given information in the knowledge base.

Attempts to make abductive reasoning an effective computational tool have given rise to *abductive logic programming* (ALP) that combines abduction with standard logic programming (LP). A number of *abductive proof procedures* have been proposed in the literature (e.g., Kakas and Mancarella 1990a, 1990b; Console *et al*. 1991; Fung and Kowalski 1997; Denecker and De Schreye 1998). These differ

in that they rely upon different semantics, the most common being the (generalized) stable models semantics (Kakas and Mancarella 1990a) and the (three-valued) completion semantics (Kunen 1987). Many of these proof procedures enrich the expressive power of the abductive framework by allowing the inclusion of *integrity constraints* to further restrict the range of possible hypotheses.

ALP has also been integrated with *constraint logic programming* (CLP; Jaffar and Maher 1994; Jaffar *et al*. 1998), in order to combine abductive reasoning with an arithmetic tool for *constraint solving* (Bressan *et al*. 1997; Kowalski *et al*. 1998; Kakas *et al*. 2000, 2001) – in the sense of CLP, not to be confused with integrity constraints. In recent years, several proof procedures for ALP with constraints (ALPC) have been proposed, including ACLP (Kakas *et al*. 2000) and the $\mathscr{A}$-system (Kakas *et al*. 2001).

Important applications of ALP and ALPC include agent programming (Sadri *et al*. 2002; Kakas *et al*. 2004, 2008), (semantic) Web management applications (Toni 2001), and planning and combinatorial problems (Wetzel *et al*. 1996; Kowalski *et al*. 1998).

Here we propose CIFF, another proof procedure for ALPC that extends the IFF procedure (Fung and Kowalski 1997) in two ways, namely, (1) by integrating abductive reasoning with constraint solving and (2) by relaxing the allowedness conditions on suitable inputs given in Fung and Kowalski (1997), in order to be able to handle a wider class of problems. The CIFF proof procedure has been implemented in Prolog in the CIFF system (Terreni 2008b).

CIFF features have been exploited in various application domains. In Kakas *et al*. (2004, 2008), CIFF has been used as the computational core for modeling an agent's planning, reactivity, and temporal reasoning capabilities based on a variant of the abductive event calculus (AEC; Kowalski and Sergot 1986; Shanahan 1989). Also, a (slightly modified) prototype version of CIFF for checking and repairing XML Web sites is currently under development (Mancarella *et al*. 2007, 2009; Terreni 2008a).

We have compared empirically the CIFF system to other related systems, namely, the $\mathscr{A}$-system (Kakas *et al*. 2001; Van Nuffelen 2004), which is the closest system from both a theoretical and an implementative viewpoint, and two state-of-the-art answer set solvers: SMODELS (Niemela and Simons 1997; Simons 2000) and DLV (Eiter *et al*. 1997; Leone *et al*. 2006). These solvers implement a different (answer set) semantics (Gelfond and Lifschitz 1991) but share with our approach the objective of modeling dynamic and non-monotonic settings in a declarative (and thus human-oriented) way. The results of our tests show (1) that the CIFF system and the other systems have comparable performances and (2) that the CIFF system is able to handle variables taking values in unbound domains.

The paper is organized as follows: In the next section we give background notions about ALPC. Section 3 specifies the CIFF proof procedure, while formal results are shown in Section 4. In Section 5 we briefly describe the CIFF system, and in Section 6 we discuss some related work together with some experimental results. Finally, Section 7 concludes the paper and proposes some future work.

This paper combines and extends a number of earlier papers: Endriss *et al*. (2004b), defining an earlier version of the CIFF proof procedure, and Endriss *et al*. (2004a, 2005), and (Mancarella *et al*. 2007), all defining earlier versions of the CIFF system.

2 Abductive logic programming with constraints (ALPC)

We present here some background on ALPC. We will assume familiarity with basic concepts of logic programming (atom, term, etc.) as found, e.g., in Lloyd (1987). We will frequently write $\vec{t}$ for a vector of terms such as $t_1, \ldots, t_k$. For instance, we are going to write $p(\vec{t})$ rather than $p(t_1, \ldots, t_k)$. Throughout the paper, to simplify the presentation, we will assume that predicates cannot have the same name but different arities. Moreover, with an abuse of notation, we will often use disjunctions and conjunctions as if they were sets, and similarly for substitutions. In particular, we will abstract away from the position of a conjunct (respectively disjunct) in a conjunction (respectively disjunction), and we will apply to disjunctions and conjunctions set-theoretic operations such as union, inclusion, difference, and so on.

An *abductive logic program* is a tuple $\langle P, A, IC \rangle$ in which:

- P is a *normal logic program*, namely a set of *clauses* of the form

$$p(\vec{s}) \leftarrow l_1(\vec{t}_1) \wedge \ldots \wedge l_n(\vec{t}_n), \qquad n \geqslant 0,$$

  where $p(\vec{s})$ is an atom and each $l_i(\vec{t}_i)$ is a literal, i.e., an atom $a(\vec{t})$ or the negation of an atom $a(\vec{t})$, represented as $\neg a(\vec{t})$. We refer to $p(\vec{s})$ as the *head* of the clause and to $l_1(\vec{t}_1) \wedge \ldots \wedge l_n(\vec{t}_n)$ as the *body* of the clause. A predicate p occurring in the head of at least one clause in P is called a *defined predicate*, and the set of clauses in P such that p occurs in their heads is called the *definition set* of p.
  Any variable in a clause is implicitly universally quantified with scope the entire clause.
- A is a set of predicates, referred to as *abducible predicates*. Atoms whose predicate is an abducible predicate are referred to as *abducible atoms* or simply as *abducibles*. Abducible atoms must not occur in the head of any clause of P (without loss of generality; see Kakas *et al*. 1998).
- IC is a set of *integrity constraints* that are *implications* of the form

$$l_1(\vec{t}_1) \wedge \ldots \wedge l_n(\vec{t}_n) \;\rightarrow\; a_1(\vec{s}_1) \vee \ldots \vee a_m(\vec{s}_m), \qquad n, m \geqslant 0, \quad n + m \geqslant 1.$$

  Each of the $l_i(\vec{t}_i)$ is a literal (as defined above), while each of the $a_i(\vec{s}_i)$ is an atom. We refer to $l_1(\vec{t}_1) \wedge \ldots \wedge l_n(\vec{t}_n)$ as the *body* and to $a_1(\vec{s}_1) \vee \ldots \vee a_m(\vec{s}_m)$ as the *head* of the integrity constraint.
  Any variable in an integrity constraint is implicitly universally quantified with scope the entire implication.

Given an abductive logic program $\langle P, A, IC \rangle$, we will refer to the set of all (defined and abducible) predicates occurring in $\langle P, A, IC \rangle$ as its *Herbrand signature*. Moreover, as is the convention in LP, we will assume as given a *Herbrand universe*,

namely, a set of ground terms. Further, we will refer to all ground atoms whose predicate belongs to the Herbrand signature of $\langle P, A, IC\rangle$ and that can be built using terms in the Herbrand universe as the *Herbrand base* of $\langle P, A, IC\rangle$. Finally, we will refer to *Herbrand terms* as (ground and nonground) terms whose instances belong to the Herbrand universe. Then, a *query* Q to an abductive logic program $\langle P, A, IC\rangle$ is a conjunction of literals whose predicate belongs to the Herbrand signature of $\langle P, A, IC\rangle$ and whose arguments are Herbrand terms. Any variable occurring in Q is implicitly existentially quantified with scope Q.

A normal logic program P provides definitions for certain predicates, while abducibles can be used to extend these definitions to form possible *explanations* for queries, which can be regarded as *observations* against the background of the world knowledge encoded in the given abductive logic program. Integrity constraints, on the other hand, restrict the range of possible explanations. Note that, in general, the set of abducible predicates may not coincide with the set of all predicates without definitions in P (i.e., the set of *open* predicates).

Informally, given an abductive logic program $\langle P, A, IC\rangle$ and a query Q, an explanation for a query Q is a set of (ground) abducible atoms Δ that, together with P, both "entails" (an appropriate ground instantiation of) Q, with respect to some notion of "entailment," and "satisfies" the set of integrity constraints IC (see Kakas *et al*. 1998 for possible notions of integrity constraint "satisfaction"). The notion of "entailment" depends on the semantics associated with the logic program P (there are many different possible choices for such semantics; Kakas *et al*. 1998).

The following definition of abductive answer formalizes this informal notion of explanation.

Definition 2.1 (*Abductive answer*)
An *abductive answer* to a query Q with respect to an abductive logic program $\langle P, A, IC\rangle$ is a pair $\langle \Delta, \sigma\rangle$, where Δ is a finite set of ground abducible atoms and σ is a ground substitution for the (existentially quantified) variables occurring in Q, such that

- $P \cup \Delta \models_{LP} Q\sigma$ and
- $P \cup \Delta \models_{LP} IC$,

where $\models_{LP}$ stands for the chosen semantics for LP.

Given an abductive logic program $\langle P, A, IC\rangle$, an abductive answer to a query Q provides an explanation for Q, understood as an observation: the answer specifies which instances of the abducible predicates have to be assumed to hold for the (corresponding instances of the) observation Q to hold as well, and in addition, it forces such an explanation to validate the integrity constraints.

The framework of ALP can be usefully extended to handle constraint predicates in the same way CLP (Jaffar and Maher 1994) extends LP. The CLP framework is defined over a particular structure $\mathfrak{R}$ consisting of a domain $D(\mathfrak{R})$ and of a set of constraint predicates that includes equality ($\doteq$) and disequality ($\neq$), together with an assignment of relations on $D(\mathfrak{R})$ for each constraint predicate. We will refer to

the set of constraint predicates in $\mathfrak{R}$ as the *constraint signature* (of $\mathfrak{R}$) and to atoms of the constraint predicates as *constraint atoms* (over $\mathfrak{R}$).

The structure $\mathfrak{R}$ is equipped with a notion of $\mathfrak{R}$-satisfiability. Given a set of (possibly non-ground) constraint atoms C, the fact that C is $\mathfrak{R}$-satisfiable will be denoted as $\models_{\mathfrak{R}} C$. Moreover we denote as $\sigma \models_{\mathfrak{R}} C$ the fact that the grounding σ of the variables of C over $D(\mathfrak{R})$ satisfies C, i.e., C is $\mathfrak{R}$-satisfied.

An *abductive logic program with constraints* is a tuple $\langle P, A, IC\rangle_{\mathfrak{R}}$ with all components defined as above but where constraint atoms for $\mathfrak{R}$ might occur in the body of clauses of P and of integrity constraints of IC. Also, queries for abductive logic programs with constraints might include constraint atoms (over $\mathfrak{R}$). We keep the notion of Herbrand signature and Herbrand base as before.

The semantics of CLP is obtained by combining the LP semantics $\models_{LP}$ and the notion of $\mathfrak{R}$-satisfiability (Jaffar and Maher 1994). We denote this semantic notion as $\models_{LP(\mathfrak{R})}$, and we use it in the notion of abductive answer with respect to an abductive logic program with constraints.

Definition 2.2 (*Abductive answer with constraints*)
An *abductive answer with constraints* to a query Q with respect to an abductive logic program with constraints $\langle P, A, IC\rangle_{\mathfrak{R}}$ is a tuple $\langle \Delta, \sigma, \Gamma\rangle$, where Δ is a finite set of abducible atoms, σ is a ground substitution for the (existentially quantified) variables occurring in Q, and Γ is a set of constraint atoms such that

(1) there exists a ground substitution σ' for the variables occurring in $\Gamma\sigma$ so that $\sigma' \models_{\mathfrak{R}} \Gamma\sigma$, and
(2) for each ground substitution σ' for the variables occurring in $\Gamma\sigma$ so that $\sigma' \models_{\mathfrak{R}} \Gamma\sigma$, there exists a ground substitution σ'' for the variables occurring in $Q \cup \Delta \cup \Gamma$, with $\sigma\sigma' \subseteq \sigma''$, such that
   - $P \cup \Delta\sigma'' \models_{LP(\mathfrak{R})} Q\sigma''$ and
   - $P \cup \Delta\sigma'' \models_{LP(\mathfrak{R})} IC$.

Example 2.1
Consider the following abductive logic program with constraints (here we assume that $<$ is a constraint predicate of $\mathfrak{R}$ with the expected semantics):

$$\begin{array}{ll} P: & p(X) \leftarrow q(T_1, T_2) \wedge T_1 < X \wedge X < 8 \\ & q(X_1, X_2) \leftarrow s(X_1, a) \\ A: & \{r, s\} \\ IC: & r(Z) \rightarrow p(Z). \end{array}$$

An abductive answer with constraints for the query $Q = r(6)$ is

$$\langle \{r(6), s(T_1, a)\}, \oslash, \{T_1 < 6\}\rangle,$$

where $\oslash$ is the empty set.

Intuitively, given the query $r(6)$, the integrity constraint in IC would fire and force the atom $p(6)$ to hold, which in turn requires $s(T_1, a)$ for some $T_1 < 6$ to be true.

Considering a non-ground version of the query, for example, $Q = r(Y)$, the following is an abductive answer with constraints:

$$\langle \{r(Y), s(T_1, a)\}, \{Y/5\}, \{T_1 < Y, Y < 8\} \rangle.$$

3 The CIFF proof procedure

The language of CIFF is the same of an abductive logic program with constraints, but we assume to have the special symbols *false* and *true*. These will be used, in particular, to represent the empty body (*true*) and the empty head (*false*) of an integrity constraint.

The CIFF framework relies upon the availability of a concrete CLP structure $\mathfrak{R}$ over arithmetical domains equipped at least with the set $\{<, \leqslant, >, \geqslant, \doteq, \neq\}$ of constraint predicates whose intended semantics is the expected one[1]. The set of constraint predicates is assumed to be closed under complement[2]. When needed, we will denote by $\overline{Con}$ the complement of the constraint atom Con (e.g., $\overline{X < 3}$ is $X \geqslant 3$). We also assume that the constraint domain offers a set of functions like $+, -, *, \ldots$ whose semantics is again the expected one.

The structure $\mathfrak{R}$ is a *black box* component in the definition of the CIFF proof procedure: for handling constraint atoms and evaluating constraint functions, we rely upon an underlying *constraint solver* over $\mathfrak{R}$, which is assumed to be both sound and complete with respect to $\models_{\mathfrak{R}}$. In particular we will assume that given a constraint atom Con and its complement $\overline{Con}$, the formulae $Con \vee \overline{Con}$ and $Con \rightarrow Con$ are tautologies with respect to the constraint solver semantics. We do not commit to any concrete implementation of a constraint solver; hence the range of the admissible arguments to constraint predicates ($D(\mathfrak{R})$) depends on the specifics of the chosen constraint solver.

The semantics of the CIFF proof procedure is defined in terms of Definition 2.2 in which (1) the constraint structure $\mathfrak{R}$ is defined as above, and (2) the semantics of LP is the three-valued completion semantics (Kunen 1987) (we denote as $\models_{3(\mathfrak{R})}$ the notion of $\models_{LP(\mathfrak{R})}$ with respect to that semantics). We refer to an abductive answer with constraints as a *CIFF abductive answer*. Recall that the three-valued completion semantics embeds the Clark equality theory (CET; Clark 1978), which handles equalities over Herbrand terms.

The CIFF proof procedure operates on a set of *iff-definitions* obtained from the *completion* (Clark 1978) of the defined predicates $p_1, \ldots, p_n$ in the Herbrand signature of $\langle P, A, IC \rangle_{\mathfrak{R}}$.

The completion of a predicate p with respect to $\langle P, A, IC \rangle_{\mathfrak{R}}$ is defined as follows. Assume that the following set of clauses is the definition set of p in $\langle P, A, IC \rangle_{\mathfrak{R}}$:

$$\begin{array}{lcl} p(\vec{t}_1) & \leftarrow & D_1 \\ & \vdots & \\ p(\vec{t}_k) & \leftarrow & D_k, \end{array}$$

[1] Here $\doteq$ is used for equality instead of $=$, the latter being used for Clark's equality as shown later.
[2] Clearly, $\neq$ is the complement of $\doteq$, and vice versa.

where each D_i is a conjunction of literals and constraint atoms. The *iff-definition* of p is of the form

$$p(\vec{X}) \leftrightarrow [\vec{X} = \vec{t}_1 \wedge D_1] \vee \ldots \vee [\vec{X} = \vec{t}_k \wedge D_k],$$

where $\vec{X}$ is a vector of *fresh* variables (not occurring in any D_i or t_i) implicitly *universally* quantified with scope the entire iff-definition, and all other variables are implicitly *existentially* quantified with scope the right-hand side disjunct in which it occurs.

Note that the equality symbol $=$ is used to represent Clark's equality in iff-definitions. In the sequel, we will refer to $=$ as the *equality predicate* and to atoms containing it as *equality atoms*[3]. Note also that input programs can not include $=$ explicitly, $=$ being reserved for Clark's equality in iff-definitions.

If p is a non-abducible, non-constraint, non-equality atom and does not occur in the head of any clause of P, its iff-definition is of the form

$$p(\vec{X}) \leftrightarrow \mathit{false}.$$

Definition 3.1 (*CIFF theory and framework*)
Let $\langle P, A, IC\rangle_{\mathfrak{R}}$ be an abductive logic program with constraints. The *CIFF theory* Th relative to $\langle P, A, IC\rangle_{\mathfrak{R}}$ is the set of all the iff-definitions of each non-abducible, non-constraint predicate in the language of $\langle P, A, IC\rangle_{\mathfrak{R}}$. Moreover we say that a *CIFF framework* is the tuple $\langle Th, A, IC\rangle_{\mathfrak{R}}$.

Example 3.1
Let us consider the following abductive logic program with constraints $\langle P, A, IC\rangle_{\mathfrak{R}}$:

$$\begin{array}{ll} P: & p(T) \leftarrow s(T) \\ & p(W) \leftarrow W < 8 \\ A: & \{s\} \\ IC: & r(T) \wedge s(T) \rightarrow p(T). \end{array}$$

The resulting CIFF theory Th is

$$\begin{array}{ll} p(X) & \leftrightarrow [X = T \wedge s(T)] \vee [X = W \wedge W < 8] \\ r(Y) & \leftrightarrow \mathit{false}. \end{array}$$

With explicit quantification, the theory Th would be

$$\begin{array}{lll} \forall X & (p(X) & \leftrightarrow [\exists T(X = T \wedge s(T)] \vee [\exists W(X = W \wedge W < 8)]) \\ \forall Y & (r(Y) & \leftrightarrow \mathit{false}). \end{array}$$

Note that Th includes an iff-definition for r even though r occurs only in the integrity constraints IC. Moreover, there is no iff-definition for the abducible predicate s. To improve readability and unless otherwise stated, in the remainder we will write CIFF theories with implicit variable quantification.

Definition 3.2 (*CIFF query*)
A *CIFF query* Q is a conjunction of literals, possibly including constraint literals. All the variables in a CIFF query Q are implicitly existentially quantified with scope Q.

[3] In particular, constraints of the form $A \doteq B$ are *not* equality atoms but (equality) constraint atoms.

Allowedness. Fung and Kowalski (1997) require frameworks for their IFF proof procedure to meet a number of so-called allowedness conditions to be able to guarantee the correct operation of their proof procedure. These conditions are designed to avoid problematic patterns of quantification that can lead to problems analogous to *floundering* in LP with negation (Lloyd 1987). These allowedness conditions are primarily needed to avoid dealing with atomic conjuncts that may contain *universally* quantified variables and also to avoid keeping explicit quantifiers for the variables that are introduced during an IFF computation.

Informally, the problem arises when a universally quantified variable occurring in a clause occurs nowhere else in the body except, possibly, in a negative literal or in an abducible atom.

The IFF proof procedure for ALP (without constraints) has the following allowedness conditions:

- an integrity constraint $A \rightarrow B$ is allowed iff every variable in it also occurs in an atomic conjunct within its body A;
- an iff-definition $p(\vec{X}) \leftrightarrow D_1 \vee \ldots \vee D_n$ is allowed iff every variable, other than those in $\vec{X}$, occurring in a disjunct D_i, also occurs inside a non-equality atomic conjunct within the same D_i;
- a query is allowed iff every variable in it also occurs in an atomic conjunct within the query itself.

As stated in Fung and Kowalski (1997), the above allowedness conditions ensure statically that floundering is avoided. We will refer to a CIFF framework arising from an abductive logic program without constraints and to a query Q such that they are allowed as above as *IFF allowed*.

Also our CIFF frameworks $\langle Th, A, IC \rangle_{\mathfrak{R}}$ must be *allowed* in order to guarantee the correct operation of CIFF. Unfortunately, it is difficult to formulate appropriate allowedness conditions that guarantee correct execution of the proof procedure without imposing too many unnecessary restrictions. This is a well-known problem, which is further aggravated for languages that include constraint predicates. In particular, adapting the IFF approach, the allowedness condition for an iff-definition would be defined as follows.

Definition 3.3 (*CIFF static allowedness*)
A CIFF framework $\langle Th, A, IC \rangle_{\mathfrak{R}}$ is *CIFF statically allowed* iff it satisfies the following conditions:

- each integrity constraint $A \rightarrow B \in IC$ is such that every variable in it also occurs in a non-constraint atomic conjunct within its body A;
- each iff-definition $p(\vec{X}) \leftrightarrow D_1 \vee \ldots \vee D_n \in Th$ is such that every variable, other than those in $\vec{X}$, occurring in a disjunct D_i, also occurs in a non-equality, non-constraint atomic conjunct within the same D_i.

A CIFF query Q is CIFF statically allowed iff every variable in Q also occurs in a non-constraint atomic conjunct within the query itself.

Our proposal is to relax the above allowedness conditions and to check *dynamically*, i.e., at runtime, the risk of floundering. Some restrictions are still needed in order to ensure that the quantification of variables during a CIFF computation can be kept implicit, both for simplicity and for keeping the IFF style of behavior.

The new allowedness conditions for CIFF are defined as follows.

Definition 3.4 (*CIFF allowedness*)
A CIFF framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$ is *CIFF allowed* iff every iff-definition in Th is allowed. An iff-definition $p(\vec{X}) \leftrightarrow D_1 \vee \ldots \vee D_n$ is allowed iff every variable, other than those in $\vec{X}$, occurring in a disjunct D_i, also occurs inside an atomic conjunct within the same D_i.

A CIFF query Q is *CIFF allowed* iff every variable in it also occurs in an atomic conjunct within the query itself.

Note that in this definition there are no restrictions concerning the integrity constraints.

Example 3.2
The following CIFF framework is CIFF allowed (P_1 is the original normal logic program):

$$
\begin{array}{ll}
P_1: & p(Z) \\
 & p(Y) \leftarrow \neg q(Y) \\
Th_1: & p(X) \leftrightarrow [X = Z] \vee [X = Y \wedge \neg q(Y)] \\
 & q(X) \leftrightarrow \mathit{false} \\
 & s(X) \leftrightarrow \mathit{false} \\
A_1: & \oslash \\
IC_1: & Z \doteq W \ \rightarrow \ s(Z, W).
\end{array}
$$

It is worth noting that the above CIFF framework is neither CIFF statically allowed nor IFF allowed (note that there are no constraints in it). Indeed, in Th_1, the variable Z occurs only in an *equality atomic* conjunct and the variable Y occurs only in an *equality atomic* conjunct and in a negative literal. The following CIFF framework, instead, is not CIFF allowed (P_2 is the original normal logic program):

$$
\begin{array}{ll}
P_2: & p(Z) \leftarrow \neg q(Z, Y) \\
Th_2: & p(X) \leftrightarrow [X = Z \wedge \neg q(Z, Y)] \\
 & q(X, Y) \leftrightarrow \mathit{false} \\
 & s(X, Y) \leftrightarrow \mathit{false} \\
A_2: & \oslash \\
IC_2: & q(Z, W) \ \rightarrow \ s(Z, W).
\end{array}
$$

The non-allowedness is due to the variable Y in Th_2 that occurs only in a negative literal.

The query $Q = \neg q(V, a)$ is not CIFF allowed (and neither CIFF statically allowed nor IFF allowed) due to the variable V that occurs only in a negative literal.

Note that in some cases a CIFF framework that is not CIFF allowed can be turned into a CIFF-allowed framework by adding explicit, though useless since trivially satisfied, constraints over the *critical* variables (e.g., Y in Th_2 above). For instance, the above non-CIFF-allowed framework can be modified by changing the first clause as follows:

$$P_2: \quad p(Z) \leftarrow \neg q(Z, Y) \wedge Y \doteq Y.$$

Note however that this can be done only if the critical variables such as Y above are meant to be variables ranging over the domain $D(\Re)$; i.e., they are constraint variables.

The following example shows how the IFF allowedness requirement forbids the use of the IFF proof procedure even for simple abductive frameworks in which IFF could compute correct abductive answers.

Example 3.3

Consider the following CIFF framework:

$$\begin{array}{ll} P_3: & p(Y) \\ & q(Z) \leftrightarrow r(Z) \wedge p(a) \\ Th_3: & p(X) \leftrightarrow [X = Y] \\ & q(X) \leftrightarrow [X = Z \wedge r(Z) \wedge p(a)] \\ A_3: & \{r\} \\ IC_3: & \oslash. \end{array}$$

The above framework is not IFF allowed due to the variable Y. Consider the query $q(b)$. Intuitively there is a simple and sound abductive answer for $q(b)$, i.e., $r(b)$, and this could be computed by IFF, were it not for the allowedness restrictions it imposes on its inputs. Instead, the above framework is CIFF allowed, and as will become clear, the CIFF proof procedure returns exactly the correct answer.

Until now we have shown only "artificial" examples, but the IFF allowedness restrictions limit the application of the IFF proof procedure in many realistic settings.

Example 3.4

Abduction is a very interesting solution for modeling agent systems and agent capabilities. In particular the AEC language (Kowalski and Sergot 1986; Miller and Shanahan 2002) is a popular framework for modeling (among others) planning capabilities of an agent through abductive reasoning. The following CIFF framework models a fragment of the AEC (definitions for *init* and *term* are omitted for

simplicity):

$$
\begin{array}{lll}
AEC: & holds(G,T) \leftarrow & happens(A,T_1) \wedge init(A,G) \wedge \\
 & & \neg clip(T_1,G,T) \wedge T_1 < T \\
 & clip(T_1,G,T_2) \leftarrow & happens(A,T) \wedge term(A,G) \wedge T_1 \leqslant T \wedge T < T_2 \\
Th_{AEC}: & holds(X_1,X_2) \leftrightarrow & [X_1 = G \wedge X_2 = T \wedge happens(A,T_1) \wedge \\
 & & init(A,G) \wedge \neg clip(T_1,G,T) \wedge T_1 < T] \\
 & clip(X_1,X_2,X_3) \leftrightarrow & [X_1 = T_1 \wedge X_2 = G \wedge X_3 = T_2 \wedge \\
 & & happens(A,T) \wedge term(A,G) \wedge T_1 \leqslant T \wedge T < T_2] \\
A_{AEC}: & \{happens\} & \\
IC_{AEC}: & \oslash. &
\end{array}
$$

The above framework is neither an IFF framework due to the presence of constraint atoms nor CIFF statically allowed due to the variable T in the first iff-definition and the variables T_1 and T_2 in the second iff-definition, violating the allowedness restrictions stated in Definition 3.3. This is because these variables occur only in equality and/or constraint atomic conjuncts in the respective disjuncts. However the framework is CIFF allowed, and CIFF can be used for reasoning with it, as done, e.g., in the KGP model (Kakas *et al*. 2008).

In the remainder of the paper, we will always assume that CIFF frameworks and CIFF queries are CIFF allowed. For simplicity, from here onward, with the word *allowed* we mean *CIFF allowed*, unless otherwise explicitly stated.

3.1 CIFF proof rules

The CIFF proof procedure is a rewriting procedure, consisting of a number of *CIFF proof rules*, each of which replaces a *CIFF formula* by another one.

In the remainder, a negative literal $L = \neg A$, everywhere in a CIFF framework, in a CIFF query, or in a CIFF formula, will be written in implicative form, i.e., $\neg A$ is written as $A \rightarrow false$.

Hence, in this context a literal is either an atom A or an implication $A \rightarrow false$.

A special case of such implication is given by the next definition.

Definition 3.5 (*CIFF disequality*)
A *CIFF disequality* is an implication of the form

$$X = t \rightarrow false,$$

where X is an *existentially quantified variable* and t is a term *not in the form of a universally quantified variable* and such that X does not occur in t.

Definition 3.6 (*CIFF formula, CIFF node, and CIFF conjunct*)
A *CIFF formula* F is a disjunction

$$N_1 \vee \ldots \vee N_n, \qquad n \geqslant 0.$$

If $n = 0$, the disjunction is equivalent to *false*.

Each disjunct N_i is a *CIFF node* that is of the form

$$C_1 \wedge \ldots \wedge C_m, \qquad m \geqslant 0.$$

If $m = 0$, the conjunction is equivalent to *true*. Each conjunct C_i is a *CIFF conjunct*, and it can be of the form of

- an atom (*atomic CIFF conjunct*),
- an implication (*implicative CIFF conjunct*, including negative literals), or
- a disjunction of conjunctions of literals (*disjunctive CIFF conjunct*),

where implications are of the form

$$L_1 \wedge \ldots \wedge L_t \rightarrow A_1 \vee \ldots \vee A_s, \qquad s, t \geqslant 1,$$

in which each L_i is a literal (possibly *false* or *true*) and each A_i is an atom (possibly *false* or *true*).

In the sequel we will refer to $L_1 \wedge \ldots \wedge L_t$ as the *body* of the implication and to $A_1 \vee \ldots \vee A_s$ as the *head* of the implication.

In a *CIFF node* N, variables that appear either in an atomic CIFF conjunct or in a disjunctive CIFF conjunct are implicitly existentially quantified with scope N. All the remaining variables, i.e., variables occurring only in implicative CIFF conjuncts, are implicitly universally quantified with the scope being the implication in which they appear.

Finally a CIFF node N can have an associated *label* λ. We will denote a node N labeled by λ as $\lambda : N$.

We are now going to present the *CIFF proof rules*. In doing that, we treat a CIFF node as a *(multi)set* of CIFF conjuncts and a CIFF formula as a *(multi)set* of CIFF nodes. That is, we represent a CIFF formula $F = N_1 \vee \ldots \vee N_n$ as

$$\{N_1, \ldots, N_n\},$$

where each N_i is a CIFF node, of the form $C_1 \wedge \ldots \wedge C_m$ represented by

$$\{C_1, \ldots, C_m\},$$

in which each C_j is a CIFF conjunct.

Example 3.5

Let us consider the following abductive logic program with constraints $\langle P, A, IC \rangle_{\mathfrak{R}}$:

$$\begin{array}{ll} P: & p \leftarrow a \\ & p \leftarrow b \\ A: & \{a, b, c\} \\ IC: & a \rightarrow c. \end{array}$$

The CIFF formula $p \wedge (a \rightarrow c)$ (composed of a single node) is represented by

$$\{\{p, (a \rightarrow c)\}\}.$$

The CIFF formula $[a \wedge (a \rightarrow c)] \vee [b \wedge (a \rightarrow c)]$, composed of two CIFF nodes (obtained in CIFF from the earlier nodes as will be seen later) $N_1 = a \wedge (a \rightarrow c)$ and $N_2 = b \wedge (a \rightarrow c)$, is represented by

$$\{\{a, (a \rightarrow c)\}, \{b, (a \rightarrow c)\}\}.$$

Each *CIFF proof rule*[4] operates over a node N within a formula F, and it will result in a new formula F'. A rule is presented in the following form:

Rule name ϕ **Input:** F, N **Output** F'

Given:	a set of CIFF conjuncts χ in N
Conditions:	a set of conditions over χ and N
Action:	{**replace**, **replace_all**, **add**, **delete**} Ψ; **mark** λ

The **Given** part identifies a (possibly empty) set of conjuncts χ in N within F. A rule ϕ can be applied on a set χ of conjuncts of N satisfying the stated **Conditions**. We say ϕ is *applicable* to F, and we call the set χ a *rule input* for ϕ. Finally, the **Action** part defines both a new set of conjuncts Ψ and an action (**replace**, **replace_all**, **add**, **delete**, or **mark**) that states, as described below, how F' is obtained from F through Ψ. In the remainder we will omit to specify the **Input** part and the **Output** part.

Given a rule ϕ as above, we denote by

$$F \xrightarrow[\phi]{N, \chi} F'$$

the *application* of rule ϕ with **Input** F, N, **Given** χ, and **Output** F'.

Abstracting from the particular action, F' is always derived from F replacing the node N by a set of nodes $\mathcal{N}$, i.e.,

$$F' = F - \{N\} \cup \mathcal{N}.$$

We refer to $\mathcal{N}$ as the *CIFF successor nodes* of N, and we refer to each node $N' \in \mathcal{N}$ as a *CIFF successor node* of N. Each type of action defines $\mathcal{N}$ as follows:

replace:	$\mathcal{N} = \{(N - \chi) \cup \Psi\}$
replace_all:	$\mathcal{N} = \{[(N - \chi) \cup \{D_1\}], \ldots, [(N - \chi) \cup \{D_k\}]\}$ where $\Psi = \{D_1 \vee \ldots \vee D_k\}$
add:	$\mathcal{N} = \{N \cup \Psi\}$
delete:	$\mathcal{N} = \{N - \Psi\}$
mark:	$\mathcal{N} = \{\lambda : N\}$

The **mark** action does not change the elements in N, but it marks the node N with the label λ[5]. All the actions, apart from the **replace_all** action, replace N by a single successor node.

[4] In the remainder, when we want to refer to a CIFF framework, a CIFF node, a CIFF formula, and so on; we drop the prefix "CIFF" if it is clear from the context.

[5] As we will see later, λ can only be the label *undefined*. When clear from the context, we will represent a CIFF node omitting its label.

In the **replace_all** action, Ψ consists of a single conjunct in disjunctive form, i.e., $\Psi = \{D_1 \vee \ldots \vee D_k\}$. This action adds to F a set $\mathcal{N}$ of k successor nodes, each of them obtained from N by deleting χ and by adding a single disjunct D_i.

We are now ready to specify the proof rules in detail.

In the presentation we are going to write $\vec{t} = \vec{s}$ as a shorthand for $t_1 = s_1 \wedge \ldots \wedge t_k = s_k$ (with the implicit assumption that the two vectors have the same length) and $[\vec{X}/\vec{t}]$ for the substitution $[X_1/t_1, \ldots, X_k/t_k]$. Note that X and Y will always represent variables.

Furthermore, in our presentation of the proof rules, we abstract away from the order of conjuncts in the body of an implication by writing the body of implications with the "critical" conjunct in the first position.

Recall that in writing the proof rules, we use implicit variable quantification described in Definition 3.6.

The first proof rule replaces an atomic conjunct in a node N by its iff-definition:

R1 - Unfolding atoms

Given: $\{\ p(\vec{t})\ \}$
Conditions: $\{\ [p(\vec{X}) \leftrightarrow D_1 \vee \ldots \vee D_n] \in Th\ \}$
Action: **replace** $\{\ (D_1 \vee \ldots \vee D_n)[\vec{X}/\vec{t}]\ \}$

Note that any variable in $D_1 \vee \ldots \vee D_n$ is implicitly existentially quantified in the resulting formula F'.

We assume that variable renaming may be applied so that all existential variables have distinct names in the resulting CIFF node.

Unfolding can be applied also to atoms occurring in the body of an implication, yielding one new implication for every disjunct in the corresponding iff-definition:

R2 - Unfolding within implications

Given: $\{\ (p(\vec{t}) \wedge B) \rightarrow H\ \}$
Conditions: $\{\ [p(\vec{X}) \leftrightarrow D_1 \vee \ldots \vee D_n] \in Th\ \}$
Action: **replace** $\{\ [(D_1[\vec{X}/\vec{t}] \wedge B) \rightarrow H], \ldots, [(D_n[\vec{X}/\vec{t}] \wedge B) \rightarrow H]\ \}$

Observe that within F', any variable in any D_i becomes universally quantified with scope the implication in which it occurs. Also in rule **R2** *renaming* of variables is assumed, as discussed for **R1**.

The next rule is the *propagation* rule, which allows us to resolve an atom in the body of an implication in N with a matching atomic conjunct also in N:

R3 - Propagation

Given: $\{\ [(p(\vec{t}) \wedge B) \rightarrow H], \quad p(\vec{s})\ \}$
Conditions: $\{\ \}$
Action: **add** $\{\ (\vec{t} = \vec{s} \wedge B) \rightarrow H\ \}$

Note that if p has no arguments, $(\vec{t} = \vec{s} \wedge B) \rightarrow H$ should be read as $(true \wedge B) \rightarrow H$.

The *splitting* rule is the only rule performing a **replace_all** action. Roughly speaking it distributes a disjunction over a conjunction:

R4 - Splitting

Given:	$\{ D_1 \vee \ldots \vee D_n \}$
Conditions:	$\{ \}$
Action:	**replace_all** $\{ D_1 \vee \ldots \vee D_n \}$

The following *factoring* rule can be used to generate two cases, one in which the given abducible atoms unify and one in which they do not:

R5 - Factoring

Given:	$\{ p(\vec{t}), \quad p(\vec{s}) \}$
Conditions:	$\{ p$ abducible $\}$
Action:	**replace** $\{ [p(\vec{t}) \wedge p(\vec{s}) \wedge (\vec{t} = \vec{s} \rightarrow \mathit{false})] \vee [p(\vec{t}) \wedge \vec{t} = \vec{s}] \}$

The next set of CIFF proof rules are the *constraint* rules. They manage constraint atoms and are, in a sense, the interface to the constraint solver. They also deal with equalities and CIFF disequalities (see Definition 3.5) that can be delegated to the constraint solver if their arguments are in the constraint domain $D(\mathfrak{R})$. The formal definition of the proof rules is quite complex; hence we first introduce some useful definitions.

Definition 3.7 (*Basic c-atom*)
A *basic c-atom* is either a constraint atom or an equality atom of the form $A = B$, where A and B are not both variables, and each is either a variable or a term ranging over the chosen constraint domain $D(\mathfrak{R})$.

As an example, $X > 3$ and $X = 2$ are both basic c-atoms, whereas $X = Y$ and $X = a$ are not (where $a \notin D(\mathfrak{R})$).

Definition 3.8 (*Basic c-conjunct and constraint variable*)
A *basic c-conjunct* is a basic c-atom which occurs as a CIFF conjunct in a node.
A *constraint variable* is a variable occurring in a basic c-conjunct.

Note that a constraint variable is always an existentially quantified variable with its scope the entire CIFF node in which it occurs. This is because it must appear in a basic c-conjunct (i.e., outside an implication).

Definition 3.9 (*c-atom and c-conjunct*)
A *c-atom* is either a basic c-atom or a non-ground equality atom of the form $A = B$ such that all the variables occurring in it are constraint variables.
A *c-conjunct* is a c-atom that occurs as a CIFF conjunct in a node.

We are now ready to present the first *constraint* proof rule.

R6 - Case analysis for constraints

Given:	$\{ (Con \wedge A) \rightarrow B \}$
Conditions:	$\{ Con$ is a c-atom $\}$
Action:	**replace** $\{ [Con' \wedge (A \rightarrow B)] \vee \overline{Con'} \}$

where Con' is $A \doteq B$ if Con is $A = B$, and Con' is Con otherwise.

Observe that as Con is a c-atom, all the variables occurring in it are constraint variables; thus they are existentially quantified.

The next rule provides the actual *constraint solving* step itself. It may be applied to any set of c-conjuncts in a node, but to guarantee soundness, eventually, it has to be applied to the set of *all* c-conjuncts in a node. To simplify presentation, we assume that the constraint solver will fail whenever it is presented with an ill-defined constraint such as $bob \leqslant 5$ (in the case of a numerical solver). For inputs that are "well typed," however, such a situation never arises.

R7 - Constraint solving

Given:	$\{ Con_1, \ldots, Con_n \}$
Conditions:	$\{$ each Con_i is a c-conjunct; $\{Con'_1, \ldots, Con'_n\}$ is not $\Re$-satisfiable $\}$
Action:	**replace** $\{ false \}$

As in the case of the previous rule, Con'_i is obtained from Con_i by replacing all occurrences of $=$ with $\doteq$.

The next proof rules deal with equalities (which are not constraint atoms to be handled by the constraint solver) and rely upon the following rewrite rules that essentially implement the term reduction part of the unification algorithm of Martelli and Montanari (1982):

(1) Replace $f(t_1, \ldots, t_k) = f(s_1, \ldots, s_k)$ by $t_1 = s_1 \wedge \ldots \wedge t_k = s_k$.

(2) Replace $f(t_1, \ldots, t_k) = g(s_1, \ldots, s_l)$ by *false* if f and g are distinct or $k \neq l$.

(3) Replace $t = t$ by *true*.

(4) Replace $X = t$ by *false* if t contains X.

(5) Replace $t = X$ by $X = t$ if X is a variable and t is not.

(6) Replace $Y = X$ by $X = Y$ if X is a universally quantified variable and Y is not.

In the following *equality rewriting* rules, we denote as $\mathscr{E}(e)$ the result of applying the above rewrite rules (1)–(6) to the equality e. If no rewrite rule can be applied, then $\mathscr{E}(e) = e$.

R8 - Equality rewriting in atoms

Given:	$\{\ t_1 = t_2\ \}$
Conditions:	$\{\ \}$
Action:	**replace** $\{\ \mathscr{E}(t_1 = t_2)\ \}$

R9 - Equality rewriting in implications

Given:	$\{\ (t_1 = t_2 \wedge B) \rightarrow H\ \}$
Conditions:	$\{\ \}$
Action:	**replace** $\{\ (\mathscr{E}(t_1 = t_2) \wedge B) \rightarrow H\ \}$

The following two *substitution rules* propagate equalities to the rest of the node. In the first case we assume that $N = (X = t \wedge Rest)$.

R10 - Substitution in atoms

Given:	$\{\ X = t, \quad Rest\ \}$
Conditions:	$\{\ X \notin t$; t is a Herbrand term $\}$
Action:	**replace** $\{\ X = t, \quad (Rest[X/t])\ \}$

R11 - Substitution in implications

Given:	$\{\ (X = t \wedge B) \rightarrow H\ \}$
Conditions:	$\{\ X$ universally quantified; $X \notin t$; t is a Herbrand term $\}$
Action:	**replace** $\{\ (B \rightarrow H)[X/t]\ \}$

Note that if B is empty, then $(B \rightarrow H)[X/t]$ should be read as $(true \rightarrow H)[X/t]$.

If none of the *equality rewriting* or *substitution* rules is applicable, then an equality in the body of an implication may give rise to a *case analysis:*

R12 - Case analysis for equalities

Given:	$\{\ (X = t \wedge B) \rightarrow H\ \}$
Conditions:	$\{\ (X = t \wedge B) \rightarrow H$ is not of the form $X = t \rightarrow false$; $X \notin t$;
	X is existentially quantified; $X = t$ is not a c-atom;
	t is not a universally quantified variable;
	t is a Herbrand term $\}$
Action:	**replace** $\{\ [X = t \wedge (B \rightarrow H)] \vee [X = t \rightarrow false]\ \}$

Note that the variables which occur in t become existentially quantified in the first disjunct, while in the second disjunct each variable in t maintains its original quantification.

The first condition of the rule avoids applying *case analysis* if the implication $(X = t \wedge B) \rightarrow H$ is of the form $X = t \rightarrow false$. This is because if it were applied, the resulting first disjunct would become $[X = t \wedge (true \rightarrow false)]$ which is trivially *false*, while the second disjunct would become $X = t \rightarrow false$ itself. The other conditions guarantee that none of the earlier rules is applicable.

The next rule moves negative literals in the body of an implication to the head of that implication:

R13 - Negation rewriting

Given:	$\{\ ((A \rightarrow false) \wedge B) \rightarrow H\ \}$
Conditions:	$\{\ \}$
Action:	**replace** $\{\ B \rightarrow (A \vee H)\ \}$

Note that if B is empty, then $B \rightarrow (A \vee H)$ should be read as $true \rightarrow (A \vee H)$.

The following are *logical simplification* rules:

R14 - Logical simplification #1

Given:	$\{\ true\ \}$
Conditions:	$\{\ \}$
Action:	**delete** $\{\ true\ \}$

R15 - Logical simplification #2

Given:	$\{\ (true \wedge B) \rightarrow H\ \}$
Conditions:	$\{\ B$ is not empty $\}$
Action:	**replace** $\{\ B \rightarrow H\ \}$

R16 - Logical simplification #3

Given:	$\{\ false \rightarrow H\ \}$
Conditions:	$\{\ \}$
Action:	**delete** $\{\ false \rightarrow H\ \}$

R17 - Logical simplification #4

Given:	$\{\ true \rightarrow H\ \}$
Conditions:	$\{\ H$ does not contain any universally quantified variable $\}$
Action:	**replace** $\{\ H\ \}$

Note that the last simplification rule replaces an implication with an empty body with its head as a CIFF conjunct. This is done only if no universally quantified variables occur in the head; otherwise we would have some universally quantified

variables outside implications in a node. For example, suppose we applied the rule on $true \rightarrow a(f(Y))$, where Y is universally quantified and a is abducible. We would obtain $a(f(Y))$ as a conjunct in a node, thus leading to two main problems: (1) the variable quantification cannot be implicit; (2) even worse, the semantics should be extended to the case of *infinitely* many instantiations of abducible atoms in an abductive answer.

The case in which H does have a universally quantified variable is dealt with by the *dynamic allowedness* rule, which is used to identify nodes with problematic quantification patterns, which could lead to floundering:

R18 - Dynamic allowedness (DA)

Given:	$\{ B \rightarrow H \}$
Conditions:	$\{$ either $B = true$ or B consists of constraint atoms alone; no other rule applies to the implication $\}$
Action:	**mark** *undefined*

Due to the definition of the other CIFF proof rules, the implication $B \rightarrow H$ to which **DA** is applied to falls in one of the following cases:

(1) $B = true$ and there is a universally quantified variable in H;
(2) there is a constraint atom in B with an universally quantified variable occurring in it.

DA allows us to avoid obtaining infinitely many abducible atoms in an abductive answer. For example, let us consider an implication of the form $X > Y \rightarrow H$ such that X is universally quantified. Depending on $D(\Re)$, there could be infinitely many instances of X satisfying the c-atom, and CIFF should handle *all* those cases. However, we believe that **DA** could be relaxed, in particular for those implications that fall in case (2) above. Consider, for example, the following implication:

$$X > 3 \wedge X < 100 \rightarrow a(X),$$

where X is universally quantified and a is an abducible predicate. If $D(\Re)$ is the set of all integers, there is a finite set of abducible atoms satisfying the implication, i.e., the set $\{a(4), a(5), \ldots, a(99)\}$. However, **DA** marks a node with this implication as undefined due to the presence of X. The relaxation of **DA** is not in the scope of this paper.

The CIFF proof rules are summarized in Table 1 in which the rules drawn from the IFF procedure are indicated by "IFF" on the right-hand side. It is worth noting that the four **Logical simplification** rules are a reformulation of the corresponding IFF rules in which **Logical simplification #4**, in particular, checks for the quantification of the variables in the head of an implication for managing correctly the floundering problem. Moreover, **Case analysis for equalities** is a slight extension of the corresponding IFF rule for handling c-atoms.

Table 1. *CIFF proof rules.*

R1	Unfolding atoms	IFF
R2	Unfolding in implications	IFF
R3	Propagation	IFF
R4	Splitting	IFF
R5	Factoring	IFF
R6	Case analysis for constraints	
R7	Constraint solving	
R8	Equality rewriting in atoms	IFF
R9	Equality rewriting in implications	IFF
R10	Substitution in atoms	IFF
R11	Substitution in implications	IFF
R12	Case analysis for equalities	IFF
R13	Negation rewriting	IFF
R14	Logical simplification #1	IFF
R15	Logical simplification #2	IFF
R16	Logical simplification #3	IFF
R17	Logical simplification #4	IFF
R18	Dynamic allowedness	

3.2 CIFF derivation and answer extraction

The CIFF proof rules are the building blocks of a *CIFF derivation* that defines the process of computing answers with respect to a framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and a query Q.

Prior to defining a CIFF derivation formally, we introduce some useful definitions.

Definition 3.10 (*Failure and undefined CIFF nodes*)
A CIFF node N which contains *false* as an atomic CIFF conjunct is called a *failure CIFF node*. A CIFF node N marked as *undefined* is called an *undefined CIFF node*.

Definition 3.11 (*CIFF selection function*)
Let F be a CIFF formula. We define a *CIFF selection function* $\mathscr{S}$ as a function such that

$$\mathscr{S}(F) = \langle N, \phi, \chi\rangle,$$

where N is a CIFF node in F, ϕ is a CIFF proof rule, and χ is a set of CIFF conjuncts in N such that χ is a *rule input* for ϕ.

In the sequel we assume that selection functions, given a CIFF formula F, always select a triple $\langle N, \phi, \chi\rangle$ whenever a rule is applicable to F.

We are now ready to define a *CIFF prederivation* and a *CIFF branch*.

Definition 3.12 (*CIFF prederivation and initial formula*)
Let $\langle Th, A, IC\rangle_{\mathfrak{R}}$ be a CIFF framework, Q be a query, and $\mathscr{S}$ be a *CIFF selection function*. A *CIFF prederivation* for Q with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and $\mathscr{S}$ is a (finite or infinite) sequence of CIFF formulae $F_1, F_2, \ldots, F_i, F_{i+1}, \ldots$ such that each F_{i+1} is obtained from F_i through $\mathscr{S}$ as follows:

- $F_1 = \{N_1\} = \{Q \cup IC\}$, where Q and IC are treated as sets of CIFF conjuncts, (we will refer to F_1 as the *initial formula* of a CIFF prederivation);
- $\mathscr{S}(F_i) = \langle N_i, \phi_i, \chi_i\rangle$ such that N_i is neither an undefined CIFF node nor a failure CIFF node; and
- $F_i \xrightarrow[\phi_i]{N_i, \chi_i} F_{i+1}$.

The construction of a prederivation can be interpreted as the construction of an or-tree rooted at N_1 and whose nodes are CIFF nodes. Roughly speaking, the whole or-tree can be seen as a search tree for answers to the query. Note that all the variables in the query are existentially quantified in N_1 because the allowedness conditions of Definition 3.4 impose that each variable in Q occurs in an atomic conjunct of Q.

CIFF formulae F_i in a prederivation correspond to successive frontiers of the search tree. Each derivation step is done by applying (through $\mathscr{S}$) the *selected* proof rule on a set χ of CIFF conjuncts within a node N in a frontier. The resulting frontier is obtained by replacing N by the set of *successor nodes* $\mathscr{N}$.

Definition 3.13 (*Successor nodes in a CIFF prederivation*)
Let $\mathscr{D}$ be a CIFF prederivation for a query Q with respect to a CIFF framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and a selection function $\mathscr{S}$.

We say that $\mathscr{N}$ is the set of *successor nodes of N in* $\mathscr{D}$ iff

- $\mathscr{S}(F_i) = \langle N, \phi_i, \chi_i\rangle$,
- $F_i \xrightarrow[\phi_i]{N, \chi_i} F_{i+1}$, and
- for each $N' \in F_{i+1}$ such that $N' \notin F_i \backslash \{N\}$, then $N' \in \mathscr{N}$.

Moreover we say that a node N' in $\mathscr{N}$ is a *successor node of N in* $\mathscr{D}$.

Definition 3.14 (*CIFF branch*)
Given a CIFF prederivation $\mathscr{D} = F_1, F_2, \ldots, F_i, F_{i+1}, \ldots$, a *CIFF branch* $\mathscr{B}$ in $\mathscr{D}$ is a (finite or infinite) sequence of CIFF nodes $N_1, N_2, \ldots, N_i, N_{i+1}, \ldots$ such that each $N_i \in F_i$ and each N_{i+1} is a CIFF successor node of N_i in $\mathscr{D}$.

The next step, finally, is the definition of a CIFF derivation.

Definition 3.15 (*CIFF derivation*)
Let $\langle Th, A, IC\rangle_{\mathfrak{R}}$ be a CIFF framework, Q be a query, and $\mathscr{S}$ be a *CIFF selection function*. A *CIFF derivation* $\mathscr{D}$ for Q with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and $\mathscr{S}$ is a CIFF prederivation $F_1, F_2, \ldots$ such that for each CIFF branch $\mathscr{B}$ in $\mathscr{D}$ if

- $\mathscr{S}(F_i) = \langle N_i, \phi, \chi\rangle$,
- $\mathscr{S}(F_j) = \langle N_j, \phi, \chi\rangle$,

- $N_i \in \mathscr{B}$,
- $N_j \in \mathscr{B}$, and
- $i \neq j$.

Then $\phi \notin \{$**Propagation**, **Factoring**, **Equality rewriting in atoms**, **Equality rewriting in implications**, **Substitution in atoms**$\}$.

Informally, a derivation is a prederivation such that in each branch certain proof rules can be applied only once to a given set of selected CIFF conjuncts. This is because those rules can produce loops if they are applied repeatedly to the same set of conjuncts[6]. The concept of successor nodes in a prederivation is valid also for a derivation. Where it has no impact, we will omit the selection function when we refer to a derivation.

Example 3.6
Consider the following framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$:

$$\begin{array}{ll} Th: & p \leftrightarrow true \\ A: & \{a\} \\ IC: & p \rightarrow a. \end{array}$$

The following is a prederivation $\mathscr{D}$ for the query $Q = p$:

$$\begin{array}{lll} F_1 = & \{\{p, [p \rightarrow a]\}\} & \textbf{[Init]} \\ F_2 = & \{\{p, [p \rightarrow a], [true \rightarrow a]\}\} & \textbf{[R3]} \\ F_3 = & \{\{p, [p \rightarrow a], [true \rightarrow a], [true \rightarrow a]\}\} & \textbf{[R3]} \\ & \vdots & \end{array}$$

The **Propagation** rule **R3** can be applied repeatedly to the integrity constraint, giving rise to an infinite prederivation that should be avoided in a derivation[7].

Definition 3.16 (*Successor CIFF derivation*)
Let $\mathscr{D} = F_1, \ldots, F_i$ be a CIFF derivation, $\mathscr{S}$ be a CIFF selection function, and $N \in F_i$. We say that $\mathscr{D}' = F_1, \ldots, F_{i+1}$ is a *successor CIFF derivation via N of* $\mathscr{D}$ iff

- $\mathscr{S}(F_i) = \langle N, \phi_i, \chi_i\rangle$,
- $F_i \xrightarrow[\phi_i]{N, \chi_i} F_{i+1}$, and
- $\mathscr{D}'$ is a CIFF derivation.

Definition 3.17 (*Leaf and successful CIFF nodes*)
Let $\mathscr{D} = F_1, \ldots, F_i$ be a CIFF derivation. A CIFF node N in F_i is a *leaf CIFF node* iff

- it is a *failure CIFF node*;
- it is an *undefined CIFF node*; or
- there exists no successor CIFF derivation via N of $\mathscr{D}$.

[6] Note, however, that they could be applied to different *copies* of a set of conjuncts.
[7] The example shows the need of *multisets* for representing correctly CIFF formulae and CIFF nodes.

A *leaf node* which is neither a failure CIFF node nor an undefined CIFF node is called a *successful CIFF node.*

We are now ready to introduce the following classifications of *CIFF branches* and *CIFF derivations.*

Definition 3.18 (*Failure, undefined, and successful CIFF branches*)
Let $\mathscr{D}$ be a CIFF derivation, and let $\mathscr{B} = N_1, \ldots, N_k$ be a CIFF branch in $\mathscr{D}$. We say that $\mathscr{B}$ is

- a *successful CIFF branch* if N_k is a successful CIFF node;
- a *failure CIFF branch* if N_k is a failure CIFF node;
- an *undefined CIFF branch* if N_k is an undefined CIFF node.

Definition 3.19 (*Failure and successful CIFF derivations*)
Let $\mathscr{D}$ be a CIFF derivation; $\mathscr{D}$ is called a *successful CIFF derivation* iff it contains at least one *successful CIFF branch*; $\mathscr{D}$ is called a *failure CIFF derivation* iff all its branches are *failure CIFF branches.*

Intuitively, an abductive answer to a query Q can be extracted from a successful node of a *successful derivation.* Formally, the following can be defined.

Definition 3.20 (*CIFF extracted answer*)
Let $\langle Th, A, IC\rangle_{\mathfrak{R}}$ be a CIFF framework, and let Q be a CIFF query. Let $\mathscr{D}$ be a successful CIFF derivation for Q with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$. A *CIFF extracted answer* from a successful node N of $\mathscr{D}$ is a pair

$$\langle \Delta, C\rangle,$$

where Δ is the set of abducible atomic conjuncts in N and $C = \langle \Gamma, E, DE\rangle$ in which

- Γ is the set of all the c-conjuncts in N,
- E is the set of all the *equality atoms* (i.e., equalities over Herbrand terms) in N, and
- DE is the set of all the CIFF disequalities in N.

The soundness of the CIFF proof procedure with respect to the notion of $\mathfrak{R}$-satisfiability and the three-valued completion semantics is the subject of the next section. The idea is to show that CIFF extracted answers correspond to abductive answers with constraints in the sense of Definition 2.2.

Example 3.7
Consider the following framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$, obtained from the abductive logic program with constraints of Example 2.1, and the following query Q:

$$\begin{array}{ll} Th: & p(T) \leftrightarrow T = X \wedge q(T_1, T_2) \wedge T_1 < X \wedge X < 8 \\ & q(X, Y) \leftrightarrow X = X_1 \wedge Y = X_2 \wedge s(X_1, a) \\ A: & \{r, s\} \\ IC: & r(Z) \rightarrow p(Z) \\ Q: & r(Y). \end{array}$$

The following is a CIFF derivation $\mathscr{D}$ for Q with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$:

$F_1 = \{\{r(Y), [r(Z) \rightarrow p(Z)]\}\}$ **[Init]**
$F_2 = \{\{r(Y), [Z = Y \rightarrow p(Z)], [r(Z) \rightarrow p(Z)]\}\}$ **[R3]**
$F_3 = \{\{r(Y), [true \rightarrow p(Y)], [r(Z) \rightarrow p(Z)]\}\}$ **[R11]**
$F_4 = \{\{r(Y), p(Y), [r(Z) \rightarrow p(Z)]\}\}$ **[R17]**
$F_5 = \{\{r(Y), Y = X, q(T_1, T_2), T_1 < X, X < 8, [r(Z) \rightarrow p(Z)]\}\}$ **[R1]**
$F_6 = \{\{r(X), Y = X, q(T_1, T_2), T_1 < X, X < 8, [r(Z) \rightarrow p(Z)]\}\}$ **[R10]**
$F_7 = \{\{r(X), Y = X, T_1 = V, T_2 = W, s(V, a), T_1 < X, X < 8, [r(Z) \rightarrow p(Z)]\}\}$ **[R1]**
$F_8 = \{\{r(X), Y = X, T_1 = V, T_2 = W, s(V, a), V < X, X < 8, [r(Z) \rightarrow p(Z)]\}\}$ **[R10]**

No more new rules can be applied to the only node in F_8, and this is neither a failure node nor an undefined node. Hence, it is a *successful node* from which we extract the following answer:

$$\langle \{r(X), s(V, a)\}, C\rangle,$$

where $C = \langle \Gamma, E, DE\rangle$ is

$$\begin{array}{ll} \Gamma: & \{Y = X, T_1 = V, V < X, X < 8\} \\ E: & \{T_2 = W\} \\ DE: & \oslash. \end{array}$$

Indeed, note that the *abductive answers with constraints* given in Example 2.1 are instances of the above extracted answer.

Example 3.8
Consider the following framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$ (where we assume a constraint structure $\mathfrak{R}$ over integers with the usual relations and functions) and the following query Q:

$$\begin{array}{ll} Th: & p(X) \leftrightarrow X = Z \wedge a(Z) \wedge Z < 5 \\ A: & \{a\} \\ IC: & a(2) \rightarrow false \\ Q: & p(Y). \end{array}$$

The following is a CIFF derivation $\mathscr{D}$ for Q with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$:

$F_1 = \{\{p(Y), [a(2) \rightarrow false]\}\}$ **[Init]**
$F_2 = \{\{Y = Z, a(Z), Z < 5, [a(2) \rightarrow false]\}\}$ **[R1]**
$F_3 = \{\{Y = Z, a(Z), Z < 5, [a(2) \rightarrow false], [2 = Z \rightarrow false]\}\}$ **[R3]**
$F_4 = \{\{Y = Z, a(Z), Z < 5, [a(2) \rightarrow false], [Z = 2 \rightarrow false]\}\}$ **[R9]**
$F_5 = \{\{Y = Z, a(Z), Z < 5, [a(2) \rightarrow false], [Z \neq 2 \vee [Z \doteq 2, (true \rightarrow false)]]\}\}$ **[R6]**
$F_6 = \{\{Y = Z, a(Z), Z < 5, Z \neq 2, [a(2) \rightarrow false]\},$
$\{Y = Z, a(Z), Z < 5, Z \doteq 2, [a(2) \rightarrow false], (true \rightarrow false)\}\}$ **[R4]**
$F_7 = \{\{Y = Z, a(Z), Z < 5, Z \neq 2, [a(2) \rightarrow false]\},$
$\{Y = Z, a(Z), Z < 5, Z \doteq 2, [a(2) \rightarrow false], false\}\}$ **[R17]**

Note that only the **Case analysis for constraints** rule (**R6**) can be applied to F_4 because the variable Z is a constraint variable. Hence $Z = 2$ is a c-atom (see Definition 3.9), and thus the **Case analysis for equalities** rule (**R12**) cannot be applied to F_4.

No more rules can be applied to both nodes in F_7. The first node is neither a failure node nor an undefined node. Hence, it is a *successful node* from which we

extract the following answer:

$$\langle\{a(Z)\},\langle\{Y = Z, Z < 5, Z \neq 2\},\varnothing,\varnothing\rangle\rangle.$$

4 Correctness of the CIFF proof procedure

As anticipated in the previous section, the CIFF proof procedure is sound with respect to the three-valued completion semantics; i.e., each CIFF extracted answer is indeed a CIFF correct answer in the sense of definition 2.2. All the results stated in this section (and whose proofs are given in Appendix) are based upon the results given in Fung (1996) for the IFF proof procedure.

Theorem 4.1 (*CIFF soundness*)
Let $\langle P, A, IC\rangle_{\mathfrak{R}}$ be an abductive logic program with constraints such that the corresponding CIFF framework is $\langle Th, A, IC\rangle_{\mathfrak{R}}$. Let $\langle\Delta, C\rangle$, where $C = \langle\Gamma, E, DE\rangle$, be a CIFF extracted answer from a successful CIFF node in a CIFF derivation with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and a CIFF query Q. Then there exists a ground substitution σ such that $\langle\Delta, \sigma, \Gamma\rangle$ is an abductive answer with constraints to Q with respect to $\langle P, A, IC\rangle_{\mathfrak{R}}$.

The proof of the theorem relies upon the following propositions. The first proposition shows that given a CIFF extracted answer $\langle\Delta, C\rangle$ there exists a substitution satisfying all the constraint atoms, equality atoms, and CIFF disequalities in C.

Proposition 4.1
Let $\langle\Delta, C\rangle$ be a CIFF extracted answer from a successful CIFF node N, where $C = \langle\Gamma, E, DE\rangle$. Then

(1) there exists a ground substitution θ such that $\theta \models_{3(\mathfrak{R})} \Gamma$, and
(2) for each such ground substitution θ, there exists a ground substitution σ such that

$$\theta\sigma \models_{3(\mathfrak{R})} \Gamma \cup E \cup DE.$$

Example 4.1
Given $\Gamma = \{2 \leqslant T, T < 4\}$, $E = \{X = f(Y), Z = g(V)\}$, and $DE = \{(Y = h(W, V)) \rightarrow false\}$, we have that both $\theta_1 = \{T/2\}$ and $\theta_2 = \{T/3\}$ satisfy Γ and contain all the possible assignments for T (given that $D(\mathfrak{R})$ is the set of all integers). We can obtain a ground substitution $\theta_1\sigma$ (with $\sigma = \sigma_{DE} \cup \sigma_E$) as follows:

(1) $\sigma_{DE} = \{Y/r(c)\}$ obtaining $S_1 = ((E \cup DE)\theta)\sigma_{DE} = \{X = f(r(c)), Z = g(V), (r(c) = h(W, V)) \rightarrow false\}$;

(2) the second step is to assign the corresponding terms to X and Z obtaining

$$S_2 = \{f(r(c)) = f(r(c)), g(V) = g(V), (r(c) = h(W, V)) \rightarrow false\};$$

(3) finally we assign new terms with fresh functions to the remaining existentially quantified variable V, e.g., $\sigma_E = \{V/t(c)\}$, obtaining

$$S_3 = \{f(r(c)) = f(r(c)), g(t(c)) = g(t(c)), (r(c) = h(W, t(c))) \rightarrow false\}.$$

The set S_3 is clearly entailed by CET. Note that we do not care about the universally quantified variable W in S_3. This is because

$$(r(c) = h(W, t(c))) \rightarrow false$$

is entailed by CET for any assignment to W, due to the fact that r and h are distinct function symbols.

Similarly, we can obtain another ground substitution using θ_2.

The next proposition directly extends the above result to the set Δ of a CIFF extracted answer.

Proposition 4.2
Let $\langle \Delta, C \rangle$ be a CIFF extracted answer from a successful CIFF node N, where $C = \langle \Gamma, E, DE \rangle$. For each ground substitution σ' such that $\sigma' \models_{3(\mathfrak{R})} \Gamma \cup E \cup DE$, there exists a ground substitution σ that extends σ' for the variables that are in Δ but not in C such that

(1) $\sigma' \subseteq \sigma$ and
(2) $\Delta\sigma \models_{3(\mathfrak{R})} \Delta \cup \Gamma \cup E \cup DE$.

The third proposition shows that the CIFF proof rules are indeed equivalence-preserving rules with respect to the three-valued completion semantics. This a basic requirement to prove the soundness of CIFF.

Proposition 4.3 (*Equivalence preservation*)
Given an abductive logic program with constraints $\langle P, A, IC \rangle_{\mathfrak{R}}$, a CIFF node N, and a set of CIFF successor nodes $\mathcal{N}$ obtained by applying a CIFF proof rule ϕ to N, it holds that

$$P \models_{3(\mathfrak{R})} N \quad \text{iff} \quad P \models_{3(\mathfrak{R})} \mathcal{N}^{\vee},$$

where $\mathcal{N}^{\vee}$ is the disjunction of the nodes in $\mathcal{N}$.

Corollary 4.1 (*Equivalence preservation of CIFF formulae*)
Let $\langle P, A, IC \rangle_{\mathfrak{R}}$ be an abductive logic program with constraints, F a CIFF formula, and $\mathcal{S}$ any CIFF selection function. Let $\mathcal{S}(F) = \langle N, \phi, \chi \rangle$ and F' the result of applying ϕ to N in F. Then

$$P \cup IC \models_{3(\mathfrak{R})} F \quad \text{iff} \quad P \cup IC \models_{3(\mathfrak{R})} F',$$

$$\text{i.e., } P \cup IC \models_{3(\mathfrak{R})} (F \leftrightarrow F').$$

The CIFF soundness in Theorem 4.1 concerns only those branches of a CIFF successful derivations whose leaf node is a CIFF successful node. It implies that abductive answers with constraints can be obtained also by those derivations that contain failure and undefined branches but have at least a successful branch. We also prove the following notion of soundness regarding failure CIFF derivations.

Theorem 4.2 (*Soundness of failure*)
Let $\langle P, A, IC\rangle_{\mathfrak{R}}$ be an abductive logic program with constraints such that the corresponding CIFF framework is $\langle Th, A, IC\rangle_{\mathfrak{R}}$. Let $\mathscr{D}$ be a failure CIFF derivation with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and a query Q. Then

$$P \cup IC \models_{3(\mathfrak{R})} \neg Q.$$

Note that there is a class of CIFF derivations for which a soundness result cannot be stated; i.e., all the derivations containing only undefined and failure branches. The meaning of such CIFF derivations is that for each branch, no CIFF answer can be extracted, but there are some branches (undefined branches) for which neither failure nor success is ensured. The presence of an undefined branch is due to the application of the **DA** rule, and as we have seen at the end of Section 3.1, this could lead to infinite sets of abducibles in the answers.

Concerning completeness, CIFF inherits the completeness results for IFF in Fung (1996) for the class of allowed IFF frameworks. In Fung (1996), the only requirement for ensuring completeness is the use of a *fair* selection function, i.e., a selection function that ensures that any node to which a proof rule can be applied is eventually selected in each branch of a derivation. This condition is also required in the case of CIFF. To illustrate *fairness*, suppose we have the following iff-definitions:

$$q \leftrightarrow p \vee a$$
$$p \leftrightarrow p,$$

where a is an abducible predicate. Consider the query q and an empty set of integrity constraints. After the unfolding of q, the IFF proof procedure would return the abductive answer a if the second disjunct is eventually selected, but it loops forever in the other case. A *fair* selection function ensures that the second disjunct is eventually selected during a derivation.

For the class of IFF-allowed frameworks, a CIFF derivation is exactly an IFF derivation, as there are no constraint atoms in the framework. Moreover, the **DA** rule can never apply in a derivation due to the following lemma, stating that for the class of CIFF statically allowed frameworks and queries (see Definition 3.3) there does not exist a CIFF derivation in which **DA** is applied.

Lemma 4.1 (*Static allowedness lemma*)
Let $\langle P, A, IC\rangle_{\mathfrak{R}}$ be an abductive logic program with constraints such that the corresponding CIFF framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and the query Q are both CIFF statically allowed. Then, given any CIFF derivation $F_1, F_2, \ldots$ with respect to $\langle P, A, IC\rangle_{\mathfrak{R}}$ and Q and given any selection function $\mathscr{S}$it is never the case that $\mathscr{S}(F_i) = \langle N_i, R18, \chi\rangle$ for any F_i, where $R18$ is the **DA** rule.

Indeed, the above lemma trivially applies also to IFF-allowed frameworks.

As a consequence, we can state the following result.

Theorem 4.3 (*CIFF completeness for IFF-allowed frameworks*)
Let $\langle P, A, IC\rangle_{\mathfrak{R}}$ be an abductive logic program without constraints such that the corresponding CIFF framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and the query Q do not contain constraint atoms and are IFF allowed.

If there exists an abductive answer with constraints $\langle \Delta, \sigma, \oslash\rangle$ for Q with respect to $\langle P, A, IC\rangle_{\mathfrak{R}}$, then there exists a CIFF derivation $\mathscr{D}$ for Q with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and to a fair CIFF selection function $\mathscr{S}$ such that

- $\langle \Delta', \langle \oslash, E, DE\rangle\rangle$ can be extracted from a successful CIFF node in $\mathscr{D}$; and
- there exists a ground substitution $\sigma'' \supseteq \sigma$ such that
  - (i) $P \cup \Delta'\sigma'' \models Q\sigma''$,
  - (ii) $P \cup \Delta'\sigma'' \models IC$, and
  - (iii) $\Delta'\sigma'' \subseteq \Delta\sigma''$.

Considering the whole class of CIFF frameworks, we cannot formulate a full completeness theorem for CIFF because tackling the allowedness problem dynamically, we could obtain undefined derivations, even with a *fair* selection function.

Example 4.2
Consider the following framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$ in which we assume an arithmetical constraint over integers in which $>$ has the expected meaning:

$$\begin{aligned} P:&\quad p(Y) \leftarrow a(Y)\\ Th:&\quad p(X) \leftrightarrow [X = Y \wedge a(Y)]\\ A:&\quad \{a\}\\ IC:&\quad V > 2 \rightarrow a(V). \end{aligned}$$

The following is a CIFF derivation $\mathscr{D}$ for the empty query:

$$\begin{aligned} F_1 &= \{\{[V > 2 \rightarrow a(V)]\}\} && \textbf{[Init]}\\ F_2 &= \{undefined : \{[V > 2 \rightarrow a(V)]\}\} && \textbf{[R18]} \end{aligned}$$

The only rule applicable to F_1 is the **DA** rule due to the presence of V in the constraint atom $V > 2$. Note that the existence of infinite values for V greater than 2 would give rise to an infinite set of abducibles arising from $a(V)$ in the head of the implication.

However, we can state a weak completeness theorem for the CIFF proof procedure if we assume CIFF derivations without undefined branches. The result is analogous to the completeness result shown for the $\mathscr{A}$-system (Kakas *et al*. 2001; Van Nuffelen 2004).

Theorem 4.4 (*Weak CIFF completeness*)
Let $\langle P, A, IC\rangle_{\mathfrak{R}}$ be an abductive logic program with constraints with the corresponding CIFF framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$, and let Q be a CIFF query. Let $\mathscr{D}$ be a finite CIFF derivation with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and Q such that each branch in $\mathscr{D}$ is either a failure or a successful branch. Then

(1) all the branches of $\mathscr{D}$ are failure branches if $P \cup IC \models_{3(\mathfrak{R})} \neg Q$; and
(2) there exists a successful branch in $\mathscr{D}$ if $P \cup IC \not\models_{3(\mathfrak{R})} \neg Q$ (i.e., $P \cup IC \cup Q$ is satisfiable).

The above result gives rise to the following completeness theorem for the CIFF proof procedure.

Theorem 4.5 (*Weak CIFF completeness for CIFF statically allowed frameworks*)
Let $\langle P, A, IC\rangle_{\mathfrak{R}}$ be an abductive logic program with constraints such that the corresponding CIFF framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and the query Q are both CIFF statically allowed. Let $\mathscr{D}$ be a finite CIFF derivation with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and Q. Then

(1) all the branches of $\mathscr{D}$ are failure branches if $P \cup IC \models_{3(\mathfrak{R})} \neg Q$; and
(2) there exists a successful branch in $\mathscr{D}$ if $P \cup IC \not\models_{3(\mathfrak{R})} \neg Q$ (i.e., $P \cup IC \cup Q$ is satisfiable).

All the correctness results so far focus on the three-valued completion semantics. However, it is worth noting that both IFF and CIFF are sound with respect to the well-founded semantics (van Gelder *et al*. 1991), since the well-founded model is a three-valued model of the completion of a logic program (van Gelder *et al*. 1991). However IFF (and thus CIFF for the class of IFF-allowed frameworks) is not complete with respect to that semantics. Indeed, considering the iff-definition

$$p \leftrightarrow p,$$

the negative literal $\neg p$ holds with respect to the well-founded semantics, while p is *undefined* with respect to the three-valued completion semantics. Accordingly, both IFF and CIFF fail to terminate for the query $\neg p$.

5 The CIFF system

The CIFF system is a SICStus Prolog (`http://www.sics.se/isl/sicstuswww/site/index.html`) implementation of CIFF. We rely upon the SICStus CLPFD solver integrated in the platform. This is a very fast and reliable constraint solver for finite domains (Fernández and Hill 2000). The version of the system described here is version 4.0 whose engine has been almost completely rewritten with respect to older versions (Endriss *et al*. 2004a, 2005), in order to improve efficiency.

Here we give a brief general description of the CIFF system. Further details can be found in Terreni (2008a) and in the CIFF user manual (Terreni 2008b).

The main predicate, to be run at Prolog top level is

```
run_ciff( +ALP, +Query, -Answer),
```

where `ALP` is a list of `.alp` files containing an abductive logic program with constraints[8]; `Query` is a CIFF query; and `Answer` will be instantiated to either

[8] All the files in the `ALP` list together represent a single abductive logic program with constraint. This is to facilitate writing CIFF applications. A typical example is a list with two elements in which one `.alp` file contains the clauses and the integrity constraints that specify the problem and the other file contains the specification of the particular problem instance. In this way the first file could be reused for other instances.

a CIFF extracted answer (see Definition 3.20) or the special atom `undefined` if an allowedness condition is not met. A CIFF extracted answer is represented by a triple, namely, a list of abducible atoms Δ, a list of CIFF disequalities DE, and finally a list of finite-domain constraints Γ. The set of equalities E is not returned, as the final substitution (in E) is directly applied by the system. Further answers are returned via Prolog backtracking. If no (further) answer is found, the system fails, returning the control to the Prolog top level.

Each abductive logic program with constraints consists of the following components, which could be placed in any position in any `.alp` file:

- Declarations of abducible predicates, using the predicate `abducible`. For example, an abducible predicate `abd` with arity 2, is declared via

  ```
  abducible(abd(_,_)).
  ```

- Clauses, represented as

  ```
  A :- L1,..., Ln.
  ```

- Integrity constraints, represented as

  ```
  [L1,..., Lm] implies [A1,..., An].
  ```

  in which the left-hand side list represents a conjunction of CIFF literals, while the right-hand side list represents a disjunction of CIFF atoms.

Equality/disequality atoms are defined via `=`, `\==`, and constraint atoms are defined via `#=`, `#\=`, `#<`, `#=<`, `#>`, `#>=`[9]. Finally, negative literals are of the form `not(Atom)` where `Atom` is an ordinary atom.

All the clauses defining the same predicate (here a predicate is identified by its name plus its arity) are preprocessed by the system in order to build the internal representation (an iff-definition). Each iff-definition is asserted in the Prolog global state in order to retrieve such information, when needed during a CIFF derivation, in a simple and efficient way.

The CIFF proof rules are implemented in CIFF 4.0 as Prolog clauses defining `sat(+State, -Answer)`, where `State` represents the current selected CIFF node; `State` is initialized to the internal representation of the `Query` plus all the integrity constraints in (all files in) the `ALP` argument.

Throughout the computation `State` is defined as

```
state(Diseqs,CLPStore,Imps,Atoms,Abds,Disjs)[10],
```

where the aggregation of the arguments represent a CIFF node; `Diseqs` represents the set of CIFF disequalities, `CLPStore` the current finite-domain constraint store, `Imps` the set implications, `Atoms` the set of defined atoms, and `Abds` the set of

[9] Note that whenever possible, disequalities in the system are managed through the operator `\==` rather than in the corresponding (and less efficient) implicative form.

[10] The representation of the current node, in the real code, needs some further elements dropped here for simplicity.

abduced atoms, and finally `Disjs` is the set of disjunctive CIFF conjuncts in the node.

The predicate `sat` calls itself recursively until no more rules can be applied to the current `State`, thus instantiating the `Answer`.

Finally a note on the implemented CIFF selection function. We use a classical Prolog-like selection function; i.e., we always select the left-most CIFF node in a CIFF formula. It is not a *fair* selection function in the sense that it does not ensure completeness (see Section 4 for further details), but it has been found as the only possible practical choice in terms of efficiency. Without entering into technical details, this is mostly because fixing the choice of the selected node in a CIFF formula as the leftmost CIFF node, we can directly take advantage of the Prolog backtracking mechanism in order to switch to another CIFF node in case of failure.

Concerning the order of selection of the proof rules in a CIFF node, this is determined by the order of the `sat` clauses. If a `sat` clause defining a CIFF proof rule, e.g., **Unfolding atoms (R1)**, is placed before the `sat` clause defining, e.g., **Propagation (R3)**, then the system tries first to find a rule input for **R1**, and only if no such rule input can be found, the system tries **R3**.

Below we sketch the most important techniques used to make the CIFF system an efficient abductive system. For further details on these topics, please refer to Terreni (2008a).

Managing variables and equalities. Variables play a fundamental role in nodes in CIFF: they can be either universally quantified or existentially quantified. Universally quantified variables can appear only in implications (which define their scope). Existentially quantified variables can appear in any element of the node, with scope the entire node. In the system the CIFF variables are Prolog variables, but to distinguish at run-time existentially quantified and universally quantified variables we use the Prolog facility of attribute variables (Holzbaur 1992), associating with each existentially quantified variable an `existential` attribute. Moreover, whenever possible, we use the unification of Prolog for managing equality rewriting and substitutions, but we also implement the Martelli–Montanari unification algorithm (Martelli and Montanari 1982) for managing, in particular, equality rewriting and substitutions involving universally quantified variables.

Many CIFF proof rules, for example, the **Propagation (R1)** and **Unfolding (R2, R3)** rules, typically need to be followed by a set of **Equality rewriting (R8, R9)** and **Substitution (R10, R11)** rules. In the CIFF system, these "equality" rules are not treated at the same level of the other main proof rules, but rather they have been integrated within them in order to improve efficiency. In particular rules **R8**, **R9**, **R10**, and **R11** are applied transparently to the user (i.e., they are not defined as `sat` clauses) at the very end of the other proof rules, e.g., **R1**, **R2**, and **R3**.

Loop management. Recall that in the definition of a CIFF derivation (Definition 3.15), we avoid repeated applications of certain proof rules. In the CIFF system this requirement is dealt with through a non-straightforward loop management that is designed to avoid repetitive application of CIFF proof rules, in particular

Propagation (R3) and **Factoring (R5)**, to the same rule input. Obviously, in order to manage even small-/medium-size problems, loop management needs to be efficient. We do not enter into details here but just give a hint of the technique. Loop management is done by enumerating univocally each potential rule input component for **R3** and **R5** (e.g., implications for **Propagation** and abducibles for **Factoring**) in a CIFF node, maintaining them sequentially ordered throughout the computation. Then, we can (non-straightforwardly) avoid loops, applying proof rules **R3** and **R5** to appropriate rule inputs, following the order given by the enumeration.

The loop management required in a CIFF derivation for **Equality** and **Substitution** rules is, instead, obtained (almost) for free due to the integration of those proof rules in the other main proof rules as discussed above.

Constraint solving. Interfacing efficiently the CIFF system with the underlying SICStus CLPFD solver is fundamental for performance purposes. Despite a clear interface made available by the Prolog platform, the main problem in the interaction with the solver is that the solver binds variables to numbers when checking the satisfiability of the current `CLPstore` (i.e., when the **Constraint solving (R7)** rule is applied), while we want to be able to return non-ground answers. The solution adopted in the CIFF system tackles this problem through an algorithm that allows, when needed, to check the satisfiability of the `CLPstore` as usual and then restores the non-ground values via a forced backtracking.

Groundable integrity constraints. The main source of inefficiency in a CIFF computation is probably represented by integrity constraints. The main problem is the presence of universally quantified variables that potentially lead, through the **Propagation** rule, to a new implication in a CIFF node for each propagated variable instance. It is worth noting that even in a small-/medium-size CIFF application, the number of such implications resulting from integrity constraints easily grows, thus representing the main computational bottleneck.

To deal with this, we have incorporated within CIFF a specialized algorithm that can be applied to a wide class of integrity constraints, called *groundable integrity constraints.* Intuitively, an integrity constraint I is *groundable* if the set of implications obtained through the exhaustive application of CIFF proof rules (in particular **Unfolding in implications** and **Propagation**) on I is "expected to become ground" at run-time. For example, consider an integrity constraint of the form

$$p(X), q(Y) \rightarrow r(X, Y),$$

where both p and q are defined through a set of N and M ground facts respectively. Intuitively, the exhaustive application of **Unfolding in implications** gives rise, at run-time, to a set of $N * M$ implications that become ground after the application of the substitutions on X and Y. This type of integrity constraint is included in the class of *groundable integrity constraints* which is formally defined in Terreni (2008a) together with the details of an algorithm for managing it. This algorithm handles most of the operations on groundable integrity constraints in the Prolog global state, via a non-straightforward combination of assertions/retractions of the (partial) instances of the groundable integrity constraints. The system checks automatically,

in the preprocessing phase, whether an integrity constraint is a groundable integrity constraint, and it prepares all the needed data structures. This feature significantly boosts the performance of the system because, firstly, the operations on implications performed in the Prolog global state are much faster than the operations performed in a CIFF node in the usual way and, secondly, the absence of a large set of implications in a node boosts also the application of the proof rules to the other elements.

Example 5.1
The following is an example of groundable integrity constraint:

```
[q(R,C)] implies [p(R,C)].
```

in which `q` is an abducible predicate. Indeed, for all the concrete ground instances of q which are abduced during a CIFF derivation, the above integrity constraint gives rise to a set of ground implication. Note that the class of groundable integrity constraints includes integrity constraints containing abducibles in their bodies because the algorithm also manages the cases in which such abducibles are propagated to an abducible atom containing existentially quantified variables.

Example 5.2
The following is an example of an integrity constraints which is not groundable:

```
[p(X)] implies [false].
```

in which a clause defining `p(X)` is

```
p(Y).
```

The problem in this case is given by the variable `X` in the body of the integrity constraint: unfolding `p(X)` we will obtain `X = Y`, and there is no way for `Y` to be grounded.

6 Related work, comparison, and experiments

There is a huge literature on ALP with and without constraints (see, for example, Eshghi and Kowalski 1989; Kakas and Mancarella 1990a, 1990b; Pereira *et al*. 1991; Denecker and De Schreye 1992, 1998; Kakas *et al*. 1992, 1998, 2000, 2001; Bressan *et al*. 1997; Fung and Kowalski 1997; Sadri and Toni 1999; Denecker and Kakas 2002; Lin and You 2002; Ciampolini *et al*. 2003; Alferes *et al*. 2004; Van Nuffelen 2004; Christiansen and Dahl 2005). The systems closest to CIFF are the $\mathscr{A}$-system (Van Nuffelen 2004) and SCIFF (Alberti *et al*. 2007). The latter has also been developed as an extension of the IFF proof procedure to handle numerical constraints as in CLP but with the focus on the specification and verification of interactions in open agent societies. The main features of SCIFF are the support of dynamical happening of events during computations, universally quantified variables in abducibles, the concept of fulfilment and violation of expectations, given a set of events, and integrity constraints of a specialized form which requires the inclusion in their body of at least one specific social construct (an event or an expectation).

Instead, CIFF is intended as a general-purpose abductive proof procedure, keeping the spirit of the original IFF proof procedure and conservatively adding numerical constraints.

The $\mathscr{A}$-System, as remarked in Van Nuffelen (2004), is a combination of three existing abductive proof procedures, namely, the IFF proof procedure (Fung and Kowalski 1997), the ACLP proof procedure (Kakas *et al*. 2000), and, most importantly, the SLDNFA proof procedure (Denecker and De Schreye 1998), of which the $\mathscr{A}$-system is a direct descendant. The $\mathscr{A}$-system is the state-of-the-art of ALPC, borrowing the most interesting features from the above-cited proof procedures. In Section 6.1 we give a detailed comparison between CIFF and the $\mathscr{A}$-system.

Many approaches to ALP (Kakas and Mancarella 1990a, 1990b; Kakas *et al*. 2000; Lin and You 2002) rely upon the stable models semantics (Gelfond and Lifschitz 1988) and its extensions. Answer Set Programming (ASP; Baral and Gelfond (1994)) is a LP-based paradigm for computing stable models and answer set semantics. The comparison of CIFF with the two dominant answer set solvers, DLV (Eiter *et al*. 1997) and SMODELS (Niemela and Simons 1997), is discussed in Section 6.2.

In Section 6.3, we present some experimental results on concrete examples and in comparison with the $\mathscr{A}$-system and the aforementioned answer set solvers. Note that the work of Christiansen and Dahl (2005) gives an extensive experimental comparison between Hyprolog, another relevant system for ALP, and CIFF, some ASP systems, and the $\mathscr{A}$-system. Whereas CIFF is a meta-interpreter, Hyprolog avoids meta-interpretation by directly extending Prolog to incorporate abduction and constraint handling *à la* CHR (Frühwirth 1998). However, Hyprolog has restrictions on the use of negation, as mentioned in Christiansen and Dahl (2005).

Finally, in Section 6.4 we give a comparison with analytic tableaux-based methods.

6.1 Comparison with the $\mathscr{A}$-system

The $\mathscr{A}$-system and CIFF share many common points. They both rely upon the three-valued completion semantics, and their computational schemas are both based on rewrite (proof) rules. Moreover, both systems are implemented under SICStus Prolog, and the syntax of the input programs is very similar. In both systems much effort has been done, though adopting different solutions, for obtaining considerable efficiency, by exploiting the data structures and the services available in a modern Prolog platform such as SICStus. However there are also some important differences.

Treatement of integrity constraints. The $\mathscr{A}$-system framework requires that integrity constraints are in denial form. Logically, implicative integrity constraints can be written in denial form, since

$$(B \rightarrow H) \equiv ((B \wedge \neg H) \rightarrow \mathit{false}).$$

However, the operational treatement of the two representations of integrity constraints is rather different in CIFF and in the $\mathscr{A}$-system. For example, given a CIFF

integrity constraint

$$a \rightarrow b$$

(where a and b are abducibles) and an empty query, CIFF computes the empty set of abducibles, whereas given the equivalent denial

$$a \wedge \neg b \rightarrow false$$

and the same query, the $\mathscr{A}$-system computes two alternative answers: the empty set of abducibles and $\{b\}$. Indeed, by assuming b the original implication becomes true. However, in some applications this treatment leads to unintuitive behaviors. For example, if a is *alarm_sounds* and b is *evacuate*, then, with the $\mathscr{A}$-system, *evacuate* is a possible answer independently of whether *alarm_sounds* has been observed or not. This and other examples are discussed in Sadri and Toni (1999).

Negation in implications/denials. The presence of a negative literal ($\neg A$) in the body of an implication is handled by CIFF through a **Negation rewriting** rule that moves A to the head of the implication. The $\mathscr{A}$-system, instead, manages such negations with a rule similar to a **Case analysis** rule. That is, it creates a two-terms disjunction with a disjunct containing A and the other disjunct containing $\neg A$ in conjunction with the rest of the original implication. This is exactly what CIFF does in the **Case analysis for equalities (R12)** and **Case analysis for constraints (R6)** rules. However, as noted also in (Fung 1996), applying a **Case analysis** rule to a defined/abducible atom A is not in the spirit of a *three-valued* semantics approach. This is the reason why in CIFF **Case analysis** is used only for equalities and constraints, whose semantics is two-valued.

6.2 Comparison with ASP

ASP (see, e.g., Baral and Gelfond 1994; Marek and Truszczynski 1999; Baral 2003) and ALPC are strongly interconnected mechanisms for representing knowledge and reasoning. This interconnection arises at first glance, just noting that ASP is based on the answer set semantics (Gelfond and Lifschitz 1991), an "evolution" of the stable models semantics (Gelfond and Lifschitz 1988), which in turn is used as the core semantics for many abductive proof procedures (e.g., Kakas and Mancarella 1990a; Kakas *et al*. 2000; Lin and You 2002), and that abduction can be modeled in ASP, as shown, e.g., in Bonatti (2002).

Nevertheless, ASP and ALPC show important differences that we briefly discuss here, assuming the reader has some familiarity with ASP.

The ASP framework is based upon some concrete assumptions. In particular ASP relies upon programs with finite Herbrand universe. This assumption has a high impact on the computational model and, hence, on the implemented answer set solvers.

The computational model of ASP, relying upon programs with a finite Herbrand universe, shares many common points with typical constraint solving algorithms, and it is very distinct from the classic computational model of LP (mostly used in

ALPC and also in CIFF). For an excellent comparison of the two computational models, see Marek and Truszczynski (1999).

Directly from the above observations, the implemented answer set solvers benefit from a number of features that have made them popular tools for knowledge representation and reasoning: *completeness*, *termination*, and *efficiency*.

Completeness and termination follow directly from the assumption that the Herbrand universe of a program is finite.

The idea of applying constraint solving techniques in the computational model, together with hardware improvements, makes it possible to have also efficient answer set solvers, and indeed state-of-the-art solvers are able to handle hundreds of thousands of ground Herbrand terms in acceptable times. This is sufficient for many medium- to large-size applications.

However, the ASP assumptions also introduce some important limitations on the expressiveness of the framework. Even if many application domains can be modeled through ASP, there are some applications that need the possibility of introducing non-ground terms. The Web-site-repairing example described in Section 6.3.3 is one such (simple) application that is being further investigated (Mancarella *et al*. 2007, 2009). Moreover, there are applications that can be effectively modeled in ASP but for which non-ground answers could be more suitable. Consider, for example, a planning application in which we search for a plan to solve a goal G by time $T = 5$. Assume that a certain action A solves the goal. In a plan obtained from an answer set solver the action A will be bound to a ground time, for example 4 or 3. However, it might be preferable to have a more general plan with A associated with a non-ground time TA together with the constraint $TA \leqslant 5$. Obviously, this is just a hint of a planning framework that is outside the scope of this paper. Work focused on these topics include, for example, (Mancarella *et al*. 2004), and part of the SOCS European Project (SOCS-Consortium 2002–2005).

To illustrate the main conceptual differences when programming applications in ASP and CIFF, let us consider the well-known N-queens domain, where N queens have to be placed on a N*N board in such a way that for no pair of queens Q_i and Q_j, Q_i and Q_j are in the same row, or in the same column, or in the same diagonal.

We represent the problem in CIFF as follows (N is a placeholder for a natural number):

$$
\begin{array}{ll}
P: & exists\_q(R) \leftarrow q\_domain(R) \wedge q\_domain(C) \wedge q\_pos(R, C) \\
 & q\_domain(R) \leftarrow R \geqslant 1 \wedge R \leqslant N \\
 & safe(R1, C1, R2, C2) \leftarrow C1 \neq C2 \wedge\ (R1 + C1 \neq R2 + C2) \wedge \\
 & \qquad\qquad (C1 - R1 \neq C2 - R2) \\
A: & \{q\_pos\} \\
IC: & q\_pos(R1, C1) \wedge q\_pos(R2, C2) \wedge R1 \neq R2 \rightarrow safe(R1, C1, R2, C2) \\
Q: & exists\_q(1) \wedge \ldots \wedge exists\_q(N).
\end{array}
$$

The CIFF specification of the problem is very compact. A CIFF computation for the query Q proceeds as follows (we abstract away from the concrete CIFF selection function). Each $exists\_q(R)$ atom in the query (where R is one of the N integer values

between 1 and N) is unfolded giving rise to three atoms: $q\_domain(R)$, $q\_domain(C)$, and the abducible $q\_pos(R, C)$. The first two atoms are in turn unfolded populating the CIFF node with the finite-domain constraints

$$R \geqslant 1, R \leqslant N, \qquad C \geqslant 1, C \leqslant N,$$

that will be evaluated by the constraint solver. Note that the constraints concerning R are obviously *ground*, while the constraints concerning C are not *ground*.

The third atom $q\_pos(R, C)$ is instead an abducible non-ground atom (due to the presence of the constraint variable C).

Assuming that all the unfolding, the equality rewriting, and the substitutions have been done, we will obtain a node with the following abducible atoms:

$$q\_pos(1, C_1), \ldots, q\_pos(N, C_N).$$

Each pair of these has to be propagated to the integrity constraint firing N^2 non-ground instances of the $safe$ atom. The condition $R1 \neq R2$ in the body of the integrity constraint in IC avoids the propagation of the same abducible twice; i.e., it avoids having an instance like $safe(R_1, C_1, R_1, C_1)$.

At this point the $safe$ atoms are unfolded, resulting in the whole set of non-ground finite-domain constraints needed to ensure correct positioning of the queens. Finally, this set, once the solver checks its satisfiability, is returned as part of the extracted answer. The extracted answer "contains" all the possible solutions: the corresponding ground answers identifying the concrete positions of the queens can be obtained performing a *labeling* on the constraint variables (the CIFF system automatically performs the final labeling if the user wishes it).

Consider now the following ASP representation[11]:

$$row(1)$$
$$\ldots$$
$$row(N)$$
$$row(R) \rightarrow q\_pos(R, 1) \vee \ldots \vee q\_pos(R, N)$$
$$q\_pos(R1, C) \wedge q\_pos(R2, C) \wedge R1 \neq R2 \rightarrow false$$
$$q\_pos(R1, C1) \wedge q\_pos(R2, C2) \wedge row(R) \wedge R2 = R1 + R \wedge C1 = C2 + R \rightarrow false$$
$$q\_pos(R1, C1) \wedge q\_pos(R2, C2) \wedge row(R) \wedge R2 = R1 + R \wedge C2 = C1 + R \rightarrow false.$$

Also in this case all the possible solutions are returned by the answer set solvers, even if enumerating them in a ground form.

Abstracting away from syntactical differences, there is an important difference between the two specifications. The CIFF specification takes advantage of the constraint solver because it delegates the constraints on the variables inside the clause concerning the $safe$ predicate as informally described above. Conversely, in

[11] We choose the DLV representation, borrowed from `http://www.dbai.tuwien.ac.at/proj/dlv/tutorial/` because it is the representation closest to ours, and we can easily highlight the differences. For the same reason we present the DLV specification as a set of ALPC integrity constraints: DLV syntax is slightly different.

an ASP computation, the conditions on the queen positions are checked locally, resulting in a huge set of *groundable* integrity constraints, each one containing a ground pair of queen positions.

As expected (and as shown in Section 6.3.1), delegating the checks to a finite-domain constraint solver results in performances an order of magnitude faster than any answer set solver. Note that the ASP community is aware of this problem, and recently some work has been initiated on integrating ASP with constraint solvers, in an effort to reduce the grounding size and speed computation (e.g., Baselice *et al*. 2005; Mellarkod and Gelfond 2008) but for limited forms of constraints and restricted combinations of logic programs and constraints.

6.3 Experimental results

In this section, we show some experimental results obtained by running two of the most typical benchmark examples, namely, the *N-queens* problem and the *graph coloring* problem. We also present a simple instance of a Web-site-repairing framework that could be handled by CIFF. Note that we focus our experimental evaluation on examples in which abduction benefits from constraint solving, in order to illustrate the main innovative feature of CIFF with respect to its predecessor IFF, as well as related systems (ALP solvers and $\mathscr{A}$-system).

In this performance comparison we restricted our attention to three systems: the $\mathscr{A}$-system (Van Nuffelen 2004) and two state-of-the-art answer set solvers, namely, the DLV system (Eiter *et al*. 1997) and SMODELS (Niemela and Simons 1997).

All the tests have been run on a Fedora Core 5 Linux machine equipped with a 2.4-GHz PENTIUM 4, 1-GB DDR RAM. The SICStus Prolog version used throughout the tests is version 3.12.2. All execution times are expressed in seconds ("—" means that the system was still running after 10 minutes). In all examples, unless otherwise specified, the CIFF system query is the empty list `[]` representing *true* and the algorithm for groundable integrity constraints is activated. In each experiment, the formalization of the problems is taken from `http://www.dbai.tuwien.ac.at/proj/dlv/tutorial/` for DLV, from `http://www.baral.us/code/smodels/` for SMODELS, and from Van Nuffelen (2004) for the $\mathscr{A}$-system.

6.3.1 The N-queens problem

We recall the N-queens problem, already seen in Section 6.2: N queens have to be placed on a N*N board in such a way that for no pair of queens Q_i and Q_j, Q_i and Q_j are in the same row, or in the same column, or in the same diagonal.

The CIFF system formalization (**CIFF (1)**) of this problem is very simple (the query is a conjunction of N `exists_q(R)` in which each R is a natural number, distinct from each other, in $[1, N]$):

```
%%% CIFF (1)
%%% ABDUCIBLES
abducible(q_pos(_,_)).
```

```
%%% CLAUSES
q_domain(R)  :- R #>= 1, R #=< N.
  %%% N must be an integer in real code!

exists_q(R) :- q_domain(R),q_pos(R,C),q_domain(C).

safe(R1,C1,R2,C2) :- C1#\=C2, R1+C1#\=R2+C2, C1-R1#\=C2-R2.

%%% INTEGRITY CONSTRAINTS
[q_pos(R1,C1),q_pos(R2,C2),R1#\=R2] implies [safe(R1,C1,R2,C2)].
```

We also show another CIFF formalization that is a direct translation of the DLV formalization. Here, the checks on the queen position conditions are made locally in each groundable integrity constraint instance, and they are not delegated to the constraint solver. In these programs, `abs` is the absolute value function.

The DLV translation (**CIFF (2)**) is very similar to the (**CIFF (1)**) formalization, and the query is the same. But in this case the conditions on the queen positions are imposed locally in the body of the integrity constraints[12]:

```
%%% CIFF (2)
%%% DLV translation
%%% ABDUCIBLES
abducible(q_pos(_,_)).

%%% CLAUSES
row(1).
...
row(N).

%%% INTEGRITY CONSTRAINTS
[row(R)] implies [q_pos(R,1),..., q_pos(R,N)].
  %%% N must be an integer in real code!

[q_pos(R1,C),q_pos(R2,C),R1\==R2] implies [false].

[q_pos(R1,C1),q_pos(R2,C2),R1\==R2,(abs(R1-R2)#=abs(C1-C2))]
  implies [false].
```

In Table 2, we show the results for the first solution found. In the tables, we denote the $\mathscr{A}$-system as **ASYS** and the SMODELS as **SM**.

[12] The concrete CIFF syntax differs a bit from that of the program shown in Section 6.2. The conditions that avoid the placement of two queens in the same diagonal are integrated in a single integrity constraint, taking advantage of the `-` and `abs` functions of the constraint solver: the DLV system does not allow the expression of such functions. The straight DLV translation with two integrity constraints runs a bit slower in CIFF, as expected.

Table 2. `N-queens` *results (first solution).*

Queens	CIFF (1)	CIFF (2)	ASYS	SM	DLV
n = 4	0.01	0.02	0.01	0.01	0.01
n = 6	0.01	0.21	0.01	0.01	0.01
n = 8	0.03	1.29	0.03	0.01	0.01
n = 12	0.05	5.98	0.05	0.01	0.01
n = 16	0.09	410.33	0.07	0.36	0.61
n = 24	0.20	–	0.17	4.88	5.44
n = 28	0.29	–	0.27	55.32	35.17
n = 32	0.37	–	0.32	–	–
n = 64	1.62	–	1.52	–	–
n = 100	4.55	–	4.24	–	–

All systems return all the correct solutions, but we do not show the times for all solutions because the number of possible solutions is huge when N grows.

Only the CIFF system and the $\mathscr{A}$-system, through the use of the finite-domain constraint solver, can solve the problem, in a reasonable time, for a high number of queens. Note also that the CIFF system performances in the other "answer set" variants of the specification, i.e., **CIFF (2)**, is, as expected, worse in comparison with the first one, i.e., **CIFF (1)**. However, we argue that on the whole, the results show that the system is able to handle a reasonable number of ground instances.

6.3.2 The graph coloring problem

The graph coloring problem can be defined as follows: given a connected graph we want to color its nodes in a way that each node does not have the color of any of its neighbors.

The CIFF system formalization is as follows (again, we omit the domain-dependent definitions of any specific graph):

```
%%% ABDUCIBLES
abducible(abd_color(_,_)).

%%% CLAUSES
coloring(X) :- color(C),abd_color(X,C).

%%% INTEGRITY CONSTRAINTS
[vertex(X)] implies [coloring(X)].
[edge(X,Y),abd_color(X,C),abd_color(Y,C)] implies [false].
```

The results are given in Table 3, where `Jean` and `Games` are two graph instances (up to a 120-node graph)[13].

[13] They are borrowed from `http://mat.gsia.cmu.edu/COLOR/instances.html`.

Table 3. `Graph coloring` *results (first solution).*

Nodes	CIFF	CIFF (G)	ASYS	SM	DLV
4	0.09	0.01	0.01	0.01	0.01
Jean	–	0.68	0.60	0.19	0.48
Games	–	2.39	3.61	0.28	1.14

As for the N-queens problem all the systems return all the solutions. Here answer set solvers have the best performances, as the constraint solver is not involved in the computation. However, it is worth noting that performances of both the $\mathscr{A}$-system and the CIFF system, when the algorithm for *groundable* integrity constraints is activated (second column), are encouraging, even if the domain is a typical ASP application.

6.3.3 *Web site repairing*

The last example is a practical problem in which abduction can be used effectively: checking and repairing links in a Web site, given the specification of the site via an abductive logic program with constraints. This example, which follows the approach in (Toni 2001), is currently being formalized, expanded, and investigated (Mancarella *et al*. 2007, 2009; Terreni 2008a).

Consider a Web site in which a *node* (representing a Web page) can be a *book*, a *review*, or a *library*. A *link* is a relation between two nodes. Nodes and links may need to be added to guarantee some properties:

- each node must not belong to more than one type, and
- each book must have at least a link to both a review and a library.

We represent the addition of links and nodes as abducibles and we impose that

- each abduced node must be distinct from all other nodes (either abduced or not), and
- each abduced link must be distinct from all other links (either abduced or not).

The CIFF System 4.0 formalization of this problem (together with a simple Web site instance) is the following:

```
%%% ABDUCIBLES
abducible(add_node(_,_)).
abducible(add_link(_,_)).

%%%CLAUSES
is_node(N,T) :- node(N,T), node_type(T).
is_node(N,T) :- add_node(N,T), node_type(T).
node_type(lib).
```

```
node_type(book).
node_type(review).

is_link(N1,N2) :- link(N1,N2), link_check(N1,N2).
is_link(N1,N2) :- add_link(N1,N2), link_check(N1,N2).
link_check(N1,N2) :- is_node(N1,_), is_node(N2,_), N1 \== N2.
book_links(B) :- is_node(B,book), is_node(R,review), is_link(B,R),
                 is_node(L,lib), is_link(B,L).

%%% INTEGRITY CONSTRAINTS
[add_node(N,T1), node(N,T2)] implies [false].
[add_link(N1,N2), link(N1,N2)] implies [false].
[is_node(N,T1), is_node(N,T2), T1 \== T2] implies [false].
[is_node(B,book)] implies [book_links(B)].

%%%WEB SITE INSTANCE
node(n1,book).
node(n3,review).
link(n1,n3).
```

The CIFF system returns two answers representing correctly the need of a new link between the *book* `n1` and a new *library node* `L`. The first answer is

```
[add_link(n1,L), add_node(L,lib)],   %%%ABDUCIBLES
[L\==n3, L\==n1],                    %%%DISEQUALITIES
[]                                   %%%FD CONSTRAINTS
```

Note that in the answer the fact that `L` must be a *new* node, i.e., a node distinct from both `n1` and `n3`, is included.

The second answer is more complex:

```
[add_link(n1,L), add_node(L,lib),
 add_link(n1,R), add_node(R,review)],      %%%ABDUCIBLES
[L\==n3, L\==n1, R\==n3, R\==n1, R\==L],   %%%DISEQUALITIES
[]                                         %%%FD CONSTRAINTS
```

In this case, the system also adds a *new review* node `R` and provides the right links among the new nodes. Note that, again, each node must be distinct from all others: this is expressed through CIFF disequalities.

Correctly, no further answers are found, and the system terminates accordingly.

For this example we do not make a performance comparison with other systems, as both answer set solvers and the $\mathscr{A}$-system seem unable to provide correct answers due to the presence of unbound variables.

6.4 Comparison with analytic tableaux

The overall framework of the CIFF procedure resembles the method of analytic tableaux, which has been used mostly for deductive inference in a range of different

logics (D'Agostino *et al*. 1999). A tableau proof proceeds by initializing a proof tree with a set of formulae to which we then apply expansion rules, similar to those of CIFF, until we reach an explicit contradiction on every branch. This can be used to prove that a set of formulae T is unsatisfiable or that a formula φ follows from a set T (by adding the complement of φ to T before expansion). There has been a very limited amount of work on applying the tableau method to the problem of abductive inference (Mayer and Pirri 1993; Aliseda-Llera 1997; Klarman 2008). The basic idea is that if an attempted proof of $T \models \varphi$ fails, then those branches that could not be closed can provide hints as to what additional formulae would allow us to close all branches. That is, we can compute an abductive answer for the query φ given the theory T in this manner. While, in principle, it is possible to use such an approach, the search space would be enormous. The rules of CIFF (which are more complicated and tailored to specific cases than the rules of most tableau-based procedures) have been specifically designed so as to avoid at least some of this complexity and search for abductive answers more directly. Most work on tableau-based abduction has concentrated on (classical and nonclassical) propositional logics (Aliseda-Llera 1997; Klarman 2008). The only work on tableau-based abduction for first-order logic that we are aware of does not focus on algorithmic issues (Mayer and Pirri 1993). We are also not aware of any major implementations of any of the tableau-based procedures for abduction proposed in the literature.

7 Conclusions

We have presented the CIFF proof procedure, a step forward at both theoretical and implementative levels in the field of ALP (with constraints). CIFF is able to handle variables in a non-straightforward way, and it is equipped with an interface useful to a constraint solver. We have proved that CIFF is sound with respect to the three-valued completion semantics, and it enjoys some completeness properties with respect to the same semantics.

In addition, we have described the CIFF system, a Prolog implementation of the CIFF proof procedure. The CIFF system reaches good levels of efficiency and flexibility and is comparable to other state-of-the-art tools for knowledge representation and reasoning. The system has been developed in SICStus Prolog but has recently been ported to SWI-Prolog (Wielemaker 2003), the state-of-the-art open-source Prolog platform, whose constraint solver is however less efficient than the one in SICStus.

We have developed an extension of CIFF, incorporating a more sophisticated form of integrity constraints, with negation as failure in their bodies. This extension is inspired by Sadri and Toni (1999) and is described in Terreni (2008a). Even though the current implementation supports this extended treatment of negation, further work is needed to give it a formal foundation.

At the implementative level, a main issue in CIFF is the lack of a graphical user interface (GUI) that would improve its usability: we hope to add it in the CIFF System 5 release.

Other interesting features that are planned to be added to the CIFF System 5 release are the following:

- Compatibility to the SICStus Prolog 4 release (which is claimed to be much faster: a porting of the system will benefit at once from this boost in performances).
- The possibility of invoking Prolog platform functions directly. We think that this would enhance performances and ease of programming in CIFF. However, some work has to be done in order to understand how to integrate them safely.
- Further improvements in the management of *groundable integrity constraints.*
- Further experimentations with other applications, for example planning.

Finally, we also plan to compare the CIFF system with tools in the Potsdam Answer Set Solving Collection (Potassco; `http://potassco.sourceforge.net/`), which incorporate efficient implementations of constraint solving within ASP.

Acknowledgements

We would like to thank Michael Gelfond and the anonymous reviewers for their comments and suggestions. The work described in this paper has been partially supported by European Commission FET Global Computing Initiative, within the SOCS project (IST-2001-32530).

Appendix Proofs of CIFF results

Proof of Proposition 4.1

To prove the first part of the proposition, we need the semantics of the constraint solver, while to prove the second part we need the *CET*. Both are embedded in our semantics ($\models_{3(\mathfrak{R})}$), and we will write explicitly $\models_{\mathfrak{R}}$ and $\models_{CET}$, respectively, instead of $\models_{3(\mathfrak{R})}$ where appropriate.

(1) Γ is the set of c-conjuncts in N, and this is a successful node. Then the **Constraint solving** rule **R7** cannot be applied to N. Thus, by the assumption of having a sound and complete constraint solver, we have that Γ is not an unsatisfiable set of constraints; i.e., we can always obtain a ground substitution θ such that

$$\models_{\mathfrak{R}} \Gamma\theta,$$

and so

$$\theta \models_{\mathfrak{R}} \Gamma.$$

(2) Let us consider $F = (E \cup DE)\theta$. Equalities in E are of the form

$$X_i = t_i \qquad (1 \leqslant i \leqslant n, n \geqslant 0),$$

where each X_i is an existentially quantified variable and t_i is a term (containing neither universally quantified variables nor X_i itself). The scope of each variable

in E is the whole CIFF node N, and each X_i does not appear elsewhere in the node due to the exhaustive application of the **Equality rewriting in atoms** rule **R8**.

The disequalities in DE are of the form

$$X_j = t_j \rightarrow \mathit{false} \qquad (n < j \leqslant m, m \geqslant 0),$$

where each X_j is an existentially quantified variable appearing also in E (due to the **Substitution in atoms** rule **R10**) and t_j is a term not in the form of a universally quantified variable.

The ground substitution θ contains an assignment to all the constraint variables occurring in $(E \cup DE)$. This is because (i) all the equalities in E are equalities over Herbrand terms by definition and (ii) there is no CIFF disequality in DE of the form $X_i = t_i \rightarrow \mathit{false}$, where $X_i = t_i$ is a c-atom, as the **Case analysis for constraints** rule **R6** replaced any such CIFF disequality with a c-conjunct of the form $X_i \neq t_i$.

Note also that CIFF disequalities of the form $X = Y \rightarrow \mathit{false}$ such that X is a constraint variable and Y is not (or vice versa) are not a problem. This is because X has been substituted by a ground term c by θ and there is no equality of the form $Y = c$ in $E\theta$, as in that case also Y would be a constraint variable and that equality would belong to Γ.

Finally, the proposition is proven by finding a ground substitution σ such that $\models_{CET} F\sigma$, and this can be done following the proof in (Fung 1996), as follows.

First we assign a value to each existentially quantified variable X_j in $DE\theta$. We do this by using a fresh function symbol g_j; i.e., the function symbol g_j does not appear in the CIFF branch whose leaf is N (we assume here that we have an infinite number of distinct function symbols in our language). Then we choose a constant c and assign $g_j(c)$ to X_j. We define $G = F\sigma_I$, where σ_I is the ground substitution composed of the above assignments.

The second step is to assign to each variable X_i in $(E\theta)\sigma_I$ its corresponding term $s_i = t_i\sigma_I$.

Finally, for each remaining existentially quantified variable, we use another fresh function and a constant c to make assignments as for what done for CIFF disequalities.

The whole set of assignments so far obtained is the ground substitution σ, which proves the proposition. This is because after $\theta\sigma$ has been applied, each equality originally in E is of the form $t = t$ and each CIFF disequality originally in DE is of the form $f(t) = g(t) \rightarrow \mathit{false}$, which are obviously entailed by CET.

We have

$$\sigma \models_{3(\mathfrak{R})} (E \cup DE)\theta,$$

and thus, owing to $\theta \models_{\mathfrak{R}} \Gamma$, we have

$$\theta\sigma \models_{3(\mathfrak{R})} \Gamma \cup E \cup DE. \quad \square$$

Proof of Proposition 4.2

Let us consider the set $\Delta\sigma'$. There can be existentially quantified variables in Δ not assigned by σ' because they do not appear in C. Then it is enough to choose arbitrary ground terms to assign to those variables to obtain a substitution σ such that $\sigma' \subseteq \sigma$, which proves the proposition. □

Proof of Proposition 4.3

We prove the proposition considering each of the CIFF proof rules in turn. Recall that apart from the **Splitting** rule, for each proof rule the set $\mathcal{N}$ of successor nodes of N is a singleton, i.e., $\mathcal{N} = \{N'\}$.

R1 - Unfolding atoms. This rule applies a resolution step on a defined atom $p(\vec{t})$ in N and its iff-definition in Th:

$$p(\vec{X}) \leftrightarrow (D_1 \vee \ldots \vee D_n).$$

Hence, the atom $p(\vec{t})$ is replaced in N' by

$$(D_1 \vee \ldots \vee D_n)[\vec{X}/\vec{t}].$$

The replacement is obviously equivalence preserving with respect to P and $\models_{3(\mathfrak{R})}$.

R2 - Unfolding within implication. This rule resolves a defined atom $p(\vec{t})$ with its iff-definition in Th,

$$p(\vec{X}) \leftrightarrow (D_1 \vee \ldots \vee D_n),$$

as in the previous rule. The result is a set of implications in N' replacing the original implication, each one containing one of the disjuncts $D_i\theta$, with $1 \leqslant i \leqslant n$, where $\theta = [\vec{X}/\vec{t}]$. Without loss of generality, suppose that the original implication is of the form

$$(p(\vec{t}[\vec{W}, \vec{Y}]) \wedge R[\vec{W}, \vec{Y}]) \rightarrow H[\vec{W}, \vec{Y}],$$

where R is a conjunction of literals and H is a disjunction of atoms. We use the notation $E[\vec{Y}]$ to say that $\vec{Y}$ may occur in E for a generic E. Suppose that all and only the variables in $\vec{W}$ occur also in another non-implicative CIFF conjunct (recall that in a CIFF node variables appearing only within an implication are implicitly universally quantified with scope the implication itself and variables appearing outside an implication are existentially quantified with scope the whole node). Making the quantification explicit, the implication becomes

$$\exists \vec{W} \forall \vec{Y} (p(\vec{t}[\vec{W}, \vec{Y}]) \wedge R[\vec{W}, \vec{Y}] \rightarrow H[\vec{W}, \vec{Y}]).$$

To simplify the presentation, in the following we assume that $\vec{W}$ and $\vec{Y}$ may occur everywhere in the implication without denoting it explicitly. Applying resolution we obtain

$$\exists \vec{W} \forall \vec{Y} ((\exists \vec{Z}_1 D'_1\theta \vee \ldots \vee \exists \vec{Z}_n D'_n\theta) \wedge R \rightarrow H),$$

where each D_i is of the form $\exists \vec{Z}_i D'_i$ and the vectors $\vec{Z}_i$ of existentially quantified variables arise from the iff-definition. Thus we have

$$\begin{aligned}
&\exists \vec{W} \forall \vec{Y} ((\exists \vec{Z}_1 (D'_1\theta) \vee \ldots \vee \exists \vec{Z}_n (D'_n\theta)) \wedge R \rightarrow H) && \equiv \\
&\exists \vec{W} \forall \vec{Y} (\neg(\exists \vec{Z}_1 (D'_1\theta) \vee \ldots \vee \exists \vec{Z}_n (D'_n\theta)) \vee \neg R \vee H) && \equiv \\
&\exists \vec{W} \forall \vec{Y} ((\neg(\exists \vec{Z}_1 (D'_1\theta)) \wedge \ldots \wedge \neg(\exists \vec{Z}_n (D'_n\theta))) \vee \neg R \vee H) && \equiv \\
&\exists \vec{W} \forall \vec{Y} ((\neg(\exists \vec{Z}_1 (D'_1\theta)) \vee \neg R \vee H) \wedge \ldots \wedge (\neg(\exists \vec{Z}_n (D'_n\theta)) \vee \neg R \vee H)) && \equiv \\
&\exists \vec{W} (\forall \vec{Y} (\neg(\exists \vec{Z}_1 (D'_1\theta)) \vee \neg R \vee H) \wedge \ldots \wedge \forall \vec{Y} (\neg(\exists \vec{Z}_n (D'_n\theta)) \vee \neg R \vee H)) && \equiv \\
&\exists \vec{W} (\forall \vec{Y}, \vec{Z}_1 (\neg D'_1\theta \vee \neg R \vee H) \wedge \ldots \wedge \forall \vec{Y}, \vec{Z}_n (\neg D'_n\theta \vee \neg R \vee H)) && \equiv \\
&\exists \vec{W} (\forall \vec{Y}, \vec{Z}_1 (D'_1\theta \wedge R \rightarrow H) \wedge \ldots \wedge \forall \vec{Y}, \vec{Z}_n (D'_n\theta \wedge R \rightarrow H)).
\end{aligned}$$

Note that the variables $\vec{Z}_i$ in the new implications are universally quantified with scope the implication in which they occur. So with our convention for implicit quantification, the last sentence is

$$(D_1\theta \wedge R \rightarrow H) \wedge \ldots \wedge (D_n\theta \wedge R \rightarrow H).$$

R3 - Propagation. This rule uses an atomic CIFF conjunct $p(\vec{s})$ and an atom $p(\vec{t})$ within an implication of the form $(p(\vec{t}) \wedge B) \rightarrow H$, and it adds in N' an implication of the form

$$\vec{t} = \vec{s} \wedge B \rightarrow H.$$

It is obvious that due to the fact that the second implication is a consequence of the CIFF conjunct and the implication and both remain in N', the **Propagation** rule is equivalence preserving.

R4 - Splitting. This rule uses a disjunctive CIFF conjunct of the form $D = D_1 \vee \ldots \vee D_k$ and builds a set of CIFF successor nodes $\mathcal{N} = \{N_1, \ldots, N_k\}$ such that in each N_i the conjunct D is replaced by D_i.

It is obvious that the **Splitting** rule is equivalence preserving because it is an operation of disjunctive distribution over a conjunction, i.e., is a case of the tautology

$$A \wedge (D_1 \vee \ldots \vee D_k) \equiv (A \vee D_1) \wedge \ldots \wedge (A \vee D_k).$$

R5 - Factoring. This rule uses two atomic CIFF conjuncts of the form $p(\vec{t})$ and $p(\vec{s})$ and replaces them in N' by a disjunction of the form

$$(p(\vec{s}) \wedge p(\vec{t}) \wedge (\vec{t} = \vec{s} \rightarrow \mathit{false})) \vee (p(\vec{t}) \wedge \vec{t} = \vec{s}).$$

To show that the rule is equivalence preserving, consider the tautology

$$(\vec{t} = \vec{s} \rightarrow \mathit{false}) \vee \vec{t} = \vec{s}.$$

We have that

$$\begin{aligned}
&p(\vec{t}) \wedge p(\vec{s}) && \equiv \\
&p(\vec{t}) \wedge p(\vec{s}) \wedge ((\vec{t} = \vec{s} \rightarrow \mathit{false}) \vee \vec{t} = \vec{s}) && \equiv \\
&(p(\vec{t}) \wedge p(\vec{s}) \wedge (\vec{t} = \vec{s} \rightarrow \mathit{false})) \vee (p(\vec{t}) \wedge p(\vec{s}) \wedge \vec{t} = \vec{s}) && \equiv \\
&(p(\vec{t}) \wedge p(\vec{s}) \wedge (\vec{t} = \vec{s} \rightarrow \mathit{false})) \vee (p(\vec{t}) \wedge p(\vec{t}) \wedge \vec{t} = \vec{s}) && \equiv \\
&(p(\vec{t}) \wedge p(\vec{s}) \wedge (\vec{t} = \vec{s} \rightarrow \mathit{false})) \vee (p(\vec{t}) \wedge \vec{t} = \vec{s}).
\end{aligned}$$

R6 - Case analysis for constraints. Recall that variables in Con are all existentially quantified and that the constraint domain is assumed to be closed under complement; i.e., the complement $\overline{Con}$ of a constraint atom Con is a constraint atom:

$$\begin{array}{ll} (Con \wedge A) \rightarrow B & \equiv \\ Con \rightarrow (A \rightarrow B) & \equiv \\ (Con \rightarrow Con) \wedge (Con \rightarrow (A \rightarrow B)) & \equiv \\ Con \rightarrow (Con \wedge (A \rightarrow B)) & \equiv \\ \neg Con \vee (Con \wedge (A \rightarrow B)) & \equiv \\ \overline{Con} \vee (Con \wedge (A \rightarrow B)). & \end{array}$$

Variable quantification need not be taken into account here because each variable occurring in Con must be existentially quantified in order for the rule to be applied to it. Hence the quantification of those variables remains unchanged in the two resulting disjuncts.

R7 - Constraint solving. This rules replaces a set $\{Con_1, \ldots, Con_k\}$ of c-conjuncts in N by *false* in N', provided the constraint solver evaluates them as unsatisfiable. By the assumption that the constraint solver is sound and complete, the rule is obviously equivalence preserving.

R8 - Equality rewriting in atoms and **R9 - Equality rewriting in implications**. These rules are directly borrowed from the Martelli–Montanari unification algorithm. The equivalence preserving is proven by the soundness of this algorithm (Martelli and Montanari 1982).

R10 - Substitution in atoms and **R11 - Substitution in implications**. These rules simply propagate an equality either to the whole node or to the implication in which they occur. Again they are obviously equivalence-preserving rules.

R12 - Case analysis for equality. The equivalence preservation of this rule requires some carefulness due to the quantification of the variables involved. First of all note that if no variable in the **Given** formula is universally quantified the proof is trivial. For simplicity we provide the full proof for the case in which the **Given** formula contains only one universally quantified variable and no other existentially quantified variables except X. The proof can be then easily adapted to the general case. With this simplification, we need to prove that the following two formulae are equivalent (where implicit quantifications are made explicit):

F1 $\exists X \, \forall Y ((X = t \wedge B) \rightarrow H)$

F2 $[\exists X, Y (X = t \wedge (B \rightarrow H))] \vee [\exists X \forall Y (X = t \rightarrow \mathit{false})]$.

We do a *proof by cases*, using the following two (complementary) hypotheses:

Hyp1 $\neg \exists X \exists Y (X = t)$

Hyp2 $\exists X \exists Y (X = t)$.

The equivalence under **Hyp1** is trivial.

Assume **Hyp2** holds. Let s be a ground value for X such that

$\exists Y (s = t)$,

and let ϑ be the ground substitution for X and Y such that $X\vartheta = s$ and $(X = t)\vartheta$. Note that by CET, given s, such a ground substitution is unique. Consider now the formulae obtained from **F1** and **F2** by substituting X by s:

F1(s) $\forall Y((s = t \wedge B) \rightarrow H)$

F2(s) $[\exists Y(s = t \wedge (B \rightarrow H))] \vee [\forall Y(s = t \rightarrow \mathit{false})]$.

It is not difficult to see that **F1(s)** is equivalent to

$(B \rightarrow H)\vartheta$,

since for any ground instantiation of Y other than $Y\vartheta$ the implication $((s = t \wedge B) \rightarrow H)$ is trivially true.

Consider now **F2(s)**. The second disjunct is false by **Hyp2**, whereas the first disjunct is clearly equivalent to $(B \rightarrow H)\vartheta$ due to the uniqueness of ϑ.

R13 - Negation rewriting. This rule uses common logical equivalences:

$$\begin{aligned}
&((A \rightarrow \mathit{false}) \wedge B) \rightarrow H && \equiv \\
&B \rightarrow \neg(A \rightarrow \mathit{false}) \vee H && \equiv \\
&B \rightarrow \neg(\neg A \vee \mathit{false}) \vee H && \equiv \\
&B \rightarrow (A \wedge \mathit{true}) \vee H && \equiv \\
&B \rightarrow A \vee H.
\end{aligned}$$

R14, R15, R16, R17 - Logical simplification #1–#4 rules. All the four simplification rules are again obviously equivalence-preserving rules, as they use common logical equivalences.

R18 - DA. This rule does not change the elements of a node N. Hence, given that $N' = N$, ignoring the marking, the equivalence preservation is proven. □

Proof of Corollary 4.1

The proof is an immediate consequence of Proposition 4.3 because for any CIFF formula F' obtained from F through the application of a CIFF proof rule ϕ on a node N, we have that

$$F' = F - \{N\} \cup \mathcal{N},$$

where $\mathcal{N}$ is the set of successor nodes of N with respect to ϕ. □

Proof of Theorem 4.1

Let us consider a CIFF successful node N. By definition of CIFF extracted answer, the node N from which $\langle \Delta, C \rangle$ is extracted, is a conjunction of the form

$$\Delta \wedge \Gamma \wedge E \wedge DE \wedge \mathit{Rest},$$

where $C = \langle \Gamma, E, DE \rangle$ and *Rest* is a conjunction of CIFF conjuncts.

Propositions 4.1 and 4.2 ensure the existence of a ground substitution $\overline{\sigma}$ such that

$$\Delta\overline{\sigma} \models_{3(\mathfrak{R})} \Delta \cup \Gamma \cup E \cup DE.$$

Let X be the set of variables occurring in Q and θ the restriction of $\overline{\sigma}$ over the variables in X.

Let γ be a ground substitution for all the variables occurring in $Q\theta$. Let $\sigma = \theta\gamma$. It is straightforward that

$\Delta\theta\gamma \models_{3(\Re)} \Delta \cup \Gamma \cup E \cup DE$,

as the substitution γ does not involve any variable in $\Delta \cup \Gamma \cup E \cup DE$.

To prove that $\langle \Delta, \sigma, \Gamma \rangle$ is an abductive answer with constraint, we need that

(1) there exists a ground substitution σ' for the variables occurring in $\Gamma\sigma$ such that $\sigma' \models_{\Re} \Gamma\sigma$, and
(2) for each ground substitution σ' for the variables occurring in $\Gamma\sigma$ such that $\sigma' \models_{\Re} \Gamma\sigma$, there exists a ground substitution σ'' for the variables occurring in $Q \cup \Delta \cup \Gamma$, with $\sigma\sigma' \subseteq \sigma''$, in order that
   - $P \cup \Delta\sigma'' \models_{LP(\Re)} Q\sigma''$ and
   - $P \cup \Delta\sigma'' \models_{LP(\Re)} IC$.

Again, Propositions 4.1 and 4.2 ensure that

- there exists a ground substitution σ' for the variables occurring in $\Gamma\sigma$ such that $\sigma' \models_{\Re} \Gamma\sigma$, and
- for each ground substitution σ' for the variables occurring in $\Gamma\sigma$ such that $\sigma' \models_{\Re} \Gamma\sigma$, there exists a ground substitution σ'' for the variables occurring in $Q \cup \Delta \cup \Gamma$, with $\sigma\sigma' \subseteq \sigma''$, in order that

$$\Delta\sigma'' \models_{3(\Re)} \Delta \cup \Gamma \cup E \cup DE \qquad (+).$$

If we prove that $\Delta\sigma'' \models_{3(\Re)} Rest$, we have that

$$P \cup \Delta\sigma'' \models_{3(\Re)} N. \qquad (*)$$

From this, by induction and by Proposition 4.3, we will obtain

- $P \cup \Delta\sigma'' \models_{3(\Re)} Q\sigma''$ and
- $P \cup \Delta\sigma'' \models_{3(\Re)} IC$,

thus proving that $\langle \Delta, C \rangle$ is an abductive answer with constraints to Q with respect to $\langle P, A, IC \rangle_{\Re}$.

We now prove (*). It is obvious that

$$P \cup \Delta\sigma'' \models_{3(\Re)} \Delta \cup \Gamma \cup E \cup DE$$

by (+) above. We need to show that

$$P \cup \Delta\sigma'' \models_{3(\Re)} Rest.$$

Let us consider the structure of *Rest*. Due to the exhaustive application of CIFF proof rules, a CIFF conjunct in *Rest* cannot be any of the following:

- a disjunction (due to the exhaustive application of **Splitting**);
- a defined atom (due to the exhaustive application of **Unfolding atoms**);
- either *true* or *false* (due to the exhaustive application of **Logical simplification (#1–#4)** and the fact that N is not a failure node, respectively);

- an implication whose body contains a defined atom (due to the exhaustive application of **Unfolding in implications**);
- an implication with a negative literal in the body (due to the exhaustive application of **Negation rewriting**);
- an implication with *true* or *false* in the body (due to the exhaustive application of **Logical simplification (#1–#4)**);
- an implication with only equalities or constraint atoms in the body (due to the exhaustive application of **Case analysis for equalities**, **Case analysis for constraints**, **Substitution in implications**, and **DA**).

Thus, each CIFF conjunct in *Rest* is an implication whose body contains at least an abducible atom. We denote as $A_a \subseteq \Delta$ the set of abducible atoms in Δ whose predicate is a. Consider an implication $I \in Rest$ of the form $a(\vec{t}) \wedge B \rightarrow H$, where a is an abducible predicate and $\vec{t}$ may contain universally quantified variables.

Either $A_a = \oslash$ or not. If $A_a = \oslash$, then it trivially holds that $P \cup \Delta\sigma'' \models_{3(\Re)} I$ because the body of I falsified.

The case $A_a \neq \oslash$ is more interesting. Assume $A_a = a(\vec{s}_1), \ldots, a(\vec{s}_k)$. Due to the fact that a has no definition in P, $a(\vec{s}_1)\sigma'', \ldots, a(\vec{s}_k)\sigma''$ represent all and only the instances of $a(\vec{t})$ that are entailed by $P \cup \Delta\sigma''$ with respect to the three-valued completion semantics.

Hence, if $\vec{t} = \vec{s}\sigma''$, where $\vec{s}$ is such that $a(\vec{s})\sigma'' \notin A_a$, it trivially holds that $P \cup \Delta\sigma'' \models_{3(\Re)} I$, because the body of I falsified.

Consider now the case $\vec{t} = \vec{s}\sigma''$, where $\vec{s}$ is such that $a(\vec{s})\sigma'' \in A_a$. Because N is a CIFF successful node, **Propagation** has been exhaustively applied in the CIFF branch $\mathscr{B}$ whose leaf node is N. This means that for each $a(\vec{s}_i)\sigma'' \in A_a$, an implication I' of the form

$$\vec{t} = \vec{s}_i\sigma'' \wedge B \rightarrow H$$

occurs in at least a node $N_i \in \mathscr{B}$ (otherwise **Propagation** is still applicable and N is not a successful node). Then, if B of the body does not contain other abducibles, the implication I' is not in *Rest* and has been reduced to a conjunction in N.

Otherwise, if B contains another abducible atom, the process is applied again on it. Because a successful branch is finite, the proof is obtained by induction on the number of abducible atoms in B.

Hence, it holds that

$$P \cup \Delta\sigma'' \models_{3(\Re)} Rest$$

and

$$P \cup \Delta\sigma'' \models_{3(\Re)} N.$$

Let us consider the CIFF branch $\mathscr{B}$ whose leaf node is N, i.e., the branch $\mathscr{B} = N_1 = Q \wedge IC, N_2, \ldots, N_l = N$ with $l \geqslant 1$. If we prove that for each pair of nodes N_i and N_{i+1} belonging to $\mathscr{B}$ it holds that if

$$P \cup \Delta\sigma'' \models_{3(\Re)} N_{i+1}$$

then

$$P \cup \Delta\sigma'' \models_{3(\Re)} N_i,$$

we have, by induction, that

$$P \cup \Delta\sigma'' \models_{3(\Re)} Q\sigma'' \wedge IC.$$

Suppose $P \cup \Delta\sigma'' \models_{3(\Re)} N_{i+1}$, for some i. Due to the definition of CIFF branch, each node $N_{i+1} \in \mathscr{B}$ is one of the successor nodes of N_i. If N_{i+1} is obtained by N_i by applying a CIFF proof rule distinct from the **Splitting** rule, it follows immediately that

$$P \cup \Delta\sigma'' \models_{3(\Re)} N_i,$$

given that N_{i+1} is the only successor node of N_i, and thus, from Proposition 4.3, we have that $N_i \equiv N_{i+1}$. If the **Splitting** rule has been applied, however, then the node N_i is of the form

$$RestNode \wedge (D_1 \vee \ldots \vee D_n)$$

and N_{i+1} is of the form

$$(RestNode \wedge D_i) \qquad \text{for some } i \in [1, n].$$

It is obvious that the latter formula entails the former.

Summarizing, we have that

$$P \cup \Delta\sigma'' \models_{3(\Re)} Q\sigma'' \wedge IC,$$

which implies that
$P \cup \Delta\sigma'' \models_{3(\Re)} Q\sigma''$ and
$P \cup \Delta\sigma'' \models_{3(\Re)} IC$. □

Proof of Theorem 4.2
From the definition of failure CIFF derivation, $\mathscr{D}$ is a derivation starting with $Q \cup IC$ and such that all its leaf nodes are CIFF failure nodes that are equivalent to *false*.

Hence, due to Corollary 4.1 and the transitivity of the equivalence, it follows immediately that

$$P \cup IC \models_{3(\Re)} (Q \wedge IC) \leftrightarrow false.$$

Because IC occurs in both the left- and right-hand sides of the statement, we have that

$$P \cup IC \models_{3(\Re)} Q \leftrightarrow false,$$

and thus

$$P \cup IC \models_{3(\Re)} \neg Q. \quad \square$$

The proof of Lemma 4.1 requires some auxiliary definition and result given in the sequel

Definition A.1
An atom is a *pure* constraint atom if it is either a constraint atom or an equality $t = s$, where either t or s are non-Herbrand terms.

For example, the equality $X = 3$ is a *pure* constraint atom, whereas the equality $X = a$ is not.

Definition A.2 (*Statically allowed implication*)
An implication of the form $B \rightarrow H$ is *statically allowed* if and only if the following hold:

- each universally quantified variable occurring in H occurs also in B;
- each universally quantified variable occurring in a negative literal or in a pure constraint atom in B occurs also in an atomic non-constraint atom in B;
- if a universally quantified variable in B occurs only in an equality $t = s$ of B, then either t or s does not contain universally quantified variables.

Lemma A.1 (*Static allowed implications lemma*)
Let $\langle P, A, IC\rangle_{\mathfrak{R}}$ be an abductive logic program with constraints such that the corresponding CIFF framework $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and the query Q are both CIFF statically allowed. Let $\mathscr{D}$ be a CIFF derivation with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and Q. Let F_i be a CIFF formula in $\mathscr{D}$ and N be a CIFF node in F_i such that each implication (as a CIFF conjunct) in N is statically allowed. Then, for each CIFF proof rule ϕ such that

$$F_i \xrightarrow{\frac{N,\chi}{\phi}} F_{i+1},$$

each node N' in the set of CIFF successor nodes $\mathscr{N}$ of N in $\mathscr{D}$ is such that each implication (as a CIFF conjunct) in N' is statically allowed.

Proof of Lemma A.1
We need to prove that each implication I of the form $B \rightarrow H$ in each successor node N' of N is statically allowed.

For all CIFF proof rules but (R1), (R2), (R3), (R9), (R11), (R12), and (R13) the proof is trivial.

Unfolding atoms (R1). This rule resolves an atom $p(\vec{t})$ with its iff-definition $[p(\vec{X}) \leftrightarrow D_1 \vee \ldots \vee D_n] \in Th$. New implications can arise from negative literals (rewritten in implicative form) in some disjunct D_i ($i \in [1, n]$). However, by assumption, Th is statically allowed, and thus each universally quantified variable V occurring in a negative literal occurs elsewhere in a non-equality, non-constraint atom in the same disjunct. Hence any such newly introduced implication is statically allowed.

Unfolding within implications (R2). This rule resolves an atom $p(\vec{t})$ in the body of an implication with its iff-definition $[p(\vec{X}) \leftrightarrow D_1 \vee \ldots \vee D_n] \in Th$, producing n new implications $I_1, \ldots, I_n$ in the successor node of N. As for the previous case, since Th is statically allowed, each universally quantified variable V occurring in a disjunct D_i ($i \in [1, n]$) occurs elsewhere in a non-equality, non-constraint atom in the same disjunct. Hence each I_i ($i \in [1, n]$) is a statically allowed implication.

Propagation (R3). This rule resolves an atom $p(\vec{t})$ in the body of an implication I with an atom $p(\vec{s})$ as a CIFF conjunct in N, adding a new implication I' in the successor node of N, where $p(\vec{t})$ is replaced by $\vec{t} = \vec{s}$. By definition, all the variables in $\vec{s}$ are existentially quantified; hence the newly introduced implication is statically allowed.

Equality rewriting in implications (R9). This rule handles an implication I of the form $(t_1 = t_2 \wedge B) \rightarrow H$, replacing it with an implication I' of the form $((\mathscr{E}(t_1 = t_2) \wedge B) \rightarrow H$ in the successor node N' of N. Assume that I' is not a statically allowed implication. There are two cases:

- A universally quantified variable V in H occurred in B only in the equality $t_1 = t_2$, and the application of $\mathscr{E}(t_1, t_2)$ has eliminated V. This can never happen since being I statically allowed, cases (4) and (5) in the definition of $\mathscr{E}$ do not apply.
- A universally quantified variable V occurring only in $t_1 = t_2$ still occurs only in an equality $t' = s'$ introduced by the application of $\mathscr{E}(t_1 = t_2)$, and both t' and s' contain universally quantified variables. This can not happen either, since t' is a subterm of t_1, s' is a subterm of t_2, and either t_1 or t_2 does not contain universally quantified variables by the hypothesis that I is statically allowed.

Substitution in implications (R11). This rule handles an implication I of the form $(X = t \wedge B) \rightarrow H$ (where X is universally quantified and X does not occur in t), replacing it with an implication I' of the form $(B \rightarrow H)[X/t]$ in the successor node N' of N. Since I is statically allowed and I' contains one less universally quantified variable with respect to I, I' is also statically allowed.

Case analysis for equalities (R12). This rule handles an implication I of the form $(X = t \wedge B) \rightarrow H$ (where X is existentially quantified), replacing it with a disjunctive node of the form $[X = t \wedge (B \rightarrow H)] \vee [X = t \rightarrow \mathit{false}]$ (where all the variables in t in the first disjunct become existentially quantified) in the successor node N' of N. Being X existentially quantified, the implication $X = t \rightarrow \mathit{false}$ in the second disjunct is statically allowed. Moreover, due to the fact that all the variables in t become existentially quantified in the first disjunct, $B \rightarrow H$ is also statically allowed because it contains less universally quantified variables than I that is, by assumption, statically allowed.

Negation rewriting (R13). This rule handles an implication I of the form $((A \rightarrow \mathit{false}) \wedge B) \rightarrow H$, replacing it with an implication I' of the form $B \rightarrow (A \vee H)$ in the successor node N' of N. Being I statically allowed, for each variable V occurring in A, V must also occur in a non-equality, non-constraint atom in B, and thus I' is also statically allowed because each variable in $(A \vee H)$ occurs also in a non-equality, non-constraint atom in B. □

Corollary A.1

Let $\langle P, A, IC \rangle_{\mathfrak{R}}$ be an abductive logic program with constraints such that the corresponding CIFF framework $\langle Th, A, IC \rangle_{\mathfrak{R}}$ and the query Q are both CIFF statically allowed. Let $\mathscr{D}$ be a CIFF derivation with respect to $\langle Th, A, IC \rangle_{\mathfrak{R}}$ and Q. Then each implication occurring in $\mathscr{D}$ is a statically allowed implication.

Proof
Any implication in the initial node of $\mathscr{D}$ is statically allowed, since $\langle Th, A, IC\rangle_{\Re}$ and the query Q are both CIFF statically allowed by hypothesis. The result then follows directly from Lemma A.1. □

Proof of Lemma 4.1
We prove Lemma 4.1 by contradiction. Assume that there exists a CIFF derivation such that **R18 - DA** is selected. By definition of the **DA** rule, an implication of form $B \rightarrow H$ is selected such that

(i) either B is *true*
(ii) or B contains constraint atoms only, and

(iii) no other rule applies to the implication.

Due to the definition of the CIFF proof rules, (i), (ii), and (iii) above imply that

(iv) either B is *true* and H contains universally quantified variables
(v) or B contains constraint atoms only; each constraint atom in B contains universally quantified variables; and each equality atom in B is a pure constraint atom.

Note, in particular, that equalities in B are pure constraint atoms, since otherwise **R9**, **R11**, or **R12** would be applicable. In both cases (iv) and (v) the implication is not a statically allowed implication, contradicting Corollary A.1. □

Proof of Theorem 4.3
By assumption, both $\langle Th, A, IC\rangle_{\Re}$ and Q do not contain constraint atoms. This means that both the CIFF framework $\langle Th, A, IC\rangle_{\Re}$ and the CIFF query Q are also an IFF framework and an IFF query respectively. Moreover, the CIFF proof rules are a superset of the IFF proof rules. Directly from the same assumption **Case analysis for constraints** and **Constraint solving** (which are all the CIFF rules managing c-atoms) can never be applied in any derivation $\overline{\mathscr{D}}$ for Q with respect to $\langle Th, A, IC\rangle_{\Re}$.

Moreover, the fact that both $\langle Th, A, IC\rangle_{\Re}$ and Q are IFF allowed ensures that they are also CIFF statically allowed. This is trivial because an IFF-allowed query is defined exactly as a CIFF statically allowed query, and the notion of CIFF static allowedness and the notion of IFF allowedness for, respectively, a CIFF and an IFF framework, differ only for the CIFF static allowedness conditions over constraint atoms. As $\langle Th, A, IC\rangle_{\Re}$ does not contain constraint atoms, the two notions for $\langle Th, A, IC\rangle_{\Re}$ coincide. Hence $\langle Th, A, IC\rangle_{\Re}$ is also a CIFF statically allowed framework, and thus, for Lemma 4.1, **DA** is never applied. This means that any derivation $\overline{\mathscr{D}}$ for Q with respect to $\langle Th, A, IC\rangle_{\Re}$ is an IFF derivation, and thus, we can apply directly the completeness result stated in Fung and Kowalski (1997). □

Proof of Theorem 4.4

(1) It is easy to see that

$$P \cup IC \models_{3(\mathfrak{R})} \neg Q$$

is equivalent to

$$P \cup IC \models_{3(\mathfrak{R})} Q \leftrightarrow false.$$

Because IC occurs on the left-hand side of the statement, the above statement is equivalent to

$$(*) \qquad P \cup IC \models_{3(\mathfrak{R})} (Q \wedge IC) \leftrightarrow false.$$

Assume that there exists a CIFF successful branch in $\mathscr{D}$, and let *Ans* be the corresponding CIFF extracted answer. Due to the equivalence preservation of CIFF rules (Proposition 4.1) and the transitivity of the equivalence, we have that

$$P \cup IC \models_{3(\mathfrak{R})} (Q \wedge IC) \leftrightarrow (false \vee Ans),$$

which clearly contradicts the above statement (*) being *Ans* distinct from *false* due to the soundness of CIFF.

(2) Assume that all the branches in $\mathscr{D}$ are failure branches. Due to the equivalence preservation of CIFF rules (Proposition 4.1) and the transitivity of the equivalence, we have that

$$P \cup IC \models_{3(\mathfrak{R})} (Q \wedge IC) \leftrightarrow false,$$

which is equivalent to

$$P \cup IC \models_{3(\mathfrak{R})} Q \leftrightarrow false$$

and to

$$P \cup IC \models_{3(\mathfrak{R})} \neg Q,$$

which clearly contradicts that

$$P \cup IC \not\models_{3(\mathfrak{R})} \neg Q. \qquad \square$$

Proof of Theorem 4.5

By Lemma 4.1 we have that given a CIFF derivation $\mathscr{D}$ with respect to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and Q, $\mathscr{D}$ does not contain undefined branches. This is because the **DA** rule is never applied in $\mathscr{D}$, and this is the only rule that gives rise to an undefined node. Due to the assumption that $\mathscr{D}$ is finite, we have that all the final nodes in $\mathscr{D}$ are either successful or failure CIFF nodes. Hence Theorem 4.4 can be applied to $\langle Th, A, IC\rangle_{\mathfrak{R}}$ and Q, thus proving the statement. $\square$

References

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P. and Torroni., P. 2008. Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Transactions on Computational Logic 9*(4), 1–43.

Alferes, J. J., Pereira, L. M. and Swift, T. 2004. Abduction in well-founded semantics and generalized stable models via tabled dual programs. *Theory and Practice of Logic Programming 4*, 4, 383–428.

Aliseda-Llera, A. 1997. *Abduction in Logic, Philosophy of Sand Artificial Intelligence*, PhD Thesis, ILLC, University of Amsterdam, Amsterdam, The Netherlands.

Baral, C. 2003. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, New York.

Baral, C. and Gelfond, M. 1994. Logic programming and knowledge representation. *Journal of Logic Programming 19/20*, 73–148.

Baselice, S., Bonatti, P. A. and Gelfond, M. 2005. Towards an integration of answer set and constraint solving. In *Proceedings of the 21st International Conference on Logic Programming*, M. Gabbrielli and G. Gupta, Eds. Sitges, Spain, Lecture Notes in Computer Science, Vol. 3668, Springer, 52–66.

Bonatti, P. A. 2002. Abduction, ASP and open logic programs. In *Proceedings of the Ninth International Workshop on Non-Monotonic Reasoning (NMR-2002)*, 184–190. CoRR, cs.AI/0207021, http://arxiv.org/abs/cs.AI/0207021.

Bressan, S., Goh, C. H., Lee, T., Madnick, S. E. and Siegel, M. 1997. A procedure for mediation of queries to sources in disparate contexts. In *Proceedings of the International Logic Programming Symposium*, J. Maluszynski, Ed., Port Jefferson, NY, USA, MIT Press, 213–227.

Christiansen, H. and Dahl, V. 2005. Hyprolog: A new logic programming language with assumptions and abduction. In *Proceedings of the 21st International Conference on Logic Programming, (ICLP05)*, M. Gabbrielli and G. Gupta, Eds., Sitges, Spain, Lecture Notes in Computer Science, Vol. 3668, Springer, 159–173.

Ciampolini, A., Lamma, E., Mello, P., Toni, F. and Torroni, P. 2003. Cooperation and competition in ALIAS: A logic framework for agents that negotiate. *Annals of Mathematics and Artificial Intelligence 37*, 1–2, 65–91.

Clark, K. L. 1978. Negation as failure. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds., Plenum Press, New York, 77–106.

Console, L., Dupre, D. T. and Torasso, P. 1991. On the relationship between abduction and deduction. *Journal of Logic and Computation 1*, 5, 661–690.

D'Agostino, M., Gabbay, D. M., Hähnle, R., and Posegga, J., Eds. 1999. *Handbook of Tableau Methods*. Springer-Verlag.

Denecker, M. and De Schreye, D. 1992. SLDNFA: An abductive procedure for normal abductive programs. In *Proceedings of the Ninth Joint International Conference and Symposium on Logic Programming*, K. Apt, Ed., Washington, DC, USA, MIT Press, 686–700.

Denecker, M. and De Schreye, D. 1998. SLDNFA: An abductive procedure for abductive logic programs. *Journal of Logic Programming 34*, 2, 111–167.

Denecker, M. and Kakas, A. C. 2002. Abduction in logic programming. In *Computational Logic: Logic Programming and Beyond*, Essays in Honour of Robert A. Kowalski, Part I, A. C. Kakas and F. Sadri, Eds., Lecture Notes in Computer Science 2407, Springer, 402–436.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G. and Scarcello, F. 1997. A deductive system for non-monotonic reasoning. In *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning*, J. Dix, U. Furbach and A. Nerode, Eds., Lecture Notes in Computer Science 1265, Springer, London, 364–375.

Endriss, U., Hatzitaskos, M., Mancarella, P., Sadri, F., Terreni, G. and Toni, F. 2005. Refinements of the CIFF procedure. In *Proceedings of the 12th Workshop on Automated Reasoning*, A. Bundy and J. Fleuriot, Eds., University of Edinburgh.

Endriss, U., Mancarella, P., Sadri, F., Terreni, G. and Toni, F. 2004a. Abductive logic programming with CIFF: System description. In *Logics in Artificial Intelligence, 9th European Conference*, J. J. Alferes and J. A. Leite, Eds., Lisbon, Portugal, Lecture Notes in Computer Science 3229, Springer, 680–684.

Endriss, U., Mancarella, P., Sadri, F., Terreni, G. and Toni, F. 2004b. The CIFF proof procedure for abductive logic programming with constraints. In *Logics in Artificial Intelligence, 9th European Conference*, J. J. Alferes and J. A. Leite, Eds., Lisbon, Portugal, Lecture Notes in Computer Science 3229, Springer, 31–43.

Eshghi, K. and Kowalski, R. A. 1989. Abduction compared with negation by failure. In *Proceedings of the Sixth International Conference on Logic Programming*, G. Levi and M. Martelli, Eds., Lisbon, Portugal, MIT Press, 234–254.

Fernández, A. J. and Hill, P. M. 2000. A comparative study of eight constraint programming languages over the Boolean and finite domains. *Constraints 5*, 3, 275–301.

Frühwirth, T. W. 1998. Theory and practice of constraint handling rules. *J. Log. Program. 37*, 1–3, 95–138.

Fung, T. H. 1996. *Abduction by Deduction*, PhD Thesis. Imperial College, University of London, London.

Fung, T. H. and Kowalski, R. A. 1997. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming 33*, 2, 151–165.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming (ICLP/SLP)*, R. A. Kowalski and K. Bowen, Eds., Seattle, Washington, USA, MIT Press, 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

Holzbaur, C. 1992. Metastructures versus attributed variables in the context of extensible unification. In *Proceedings of Fourth Symposium on Programming Language Implementation and Logic Programming*, M. Bruynooghe and M. Wirsing, Eds., Leuven, Belgium, Lecture Notes in Computer Science 631, Springer, 260–268.

Jaffar, J. and Maher, M. 1994. Constraint logic programming: A survey. *Journal of Logic Programming 19/20*, 503–581.

Jaffar, J., Maher, M. J., Marriott, K. and Stuckey, P. J. 1998. The semantics of constraint logic programs. *Journal of Logic Programming 37*, 1–3, 1–46.

Kakas, A., Kowalski, R. and Toni, F. 1998. The role of abduction in logic programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming 5*. Oxford University Press, 235–324.

Kakas, A. C., Kowalski, R. A. and Toni, F. 1992. Abductive logic programming. *Journal of Logic and Computation 2*, 6, 719–770.

Kakas, A. C. and Mancarella, P. 1990a. Abductive logic programming. In *Proceedings of the Workshop Logic Programming and Non-Monotonic Logic*, V. W. Marek, A. Nerode, D. Pedreschi and V. S. Subrahmanian, Eds., Austin, TX, USA, 49–61.

Kakas, A. C. and Mancarella, P. 1990b. Generalized stable models: A semantics for abduction. In *Proceedings of the Ninth European Conference on Artificial Intelligence*, Stockholm, Sweden, 385–391.

Kakas, A. C., Mancarella, P., Sadri, F., Stathis, K. and Toni, F. 2004. The KGP model of agency. In *Proceedings of the 16th European Conference on Artificial Intelligence*, R. López de Mántaras, L. Saitta, Eds., Valencia, Spain, IOS Press, 33–37.

Kakas, A. C., Mancarella, P., Sadri, F., Stathis, K. and Toni, F. 2008. Computational logic foundations of KGP agents. *Journal of Artificial Intelligence Research 33*, 285–348.

Kakas, A. C., Michael, A. and Mourlas, C. 2000. ACLP: Abductive constraint logic programming. *Journal of Logic Programming 44*, 129–177.

Kakas, A. C., Van Nuffelen, B. and Denecker, M. 2001. A-system: Problem solving through abduction. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, B. Nebel, Ed., Seattle, Washington, USA, Morgan Kaufmann, 591–596.

Klarman, S. 2008. *ABox Abduction in Description Logic*, MS Thesis. ILLC, University of Amsterdam, Amsterdam, The Netherlands.

Kowalski, R. and Sergot, M. 1986. A logic-based calculus of events. *New Generation Computing 4*, 1, 67–95.

Kowalski, R. A., Toni, F. and Wetzel, G. 1998. Executing suspended logic programs. *Fundamenta Informaticae 34*, 3, 203–224.

Kunen, K. 1987. Negation in logic programming. *Journal of Logic Programming 4*, 4, 289–308.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7*, 3, 499–562.

Lin, F. and You, J.-H. 2002. Abduction in logic programming: A new definition and an abductive procedure based on rewriting. *Artificial Intelligence 140*, 1/2, 175–205.

Lloyd, J. W. 1987. *Foundations of Logic Programming*, 2nd extended ed. Springer-Verlag, New York.

Mancarella, P., Sadri, F., Terreni, G. and Toni, F. 2004. Planning partially for situated agents. In *Proceedings of the Fifth International Workshop on Computational Logic in Multi-Agent Systems*, J. A. Leite and P. Torroni, Eds., Lisbon, Portugal, Lecture Notes in Computer Science 3487, Springer, 230–248.

Mancarella, P., Sadri, F., Terreni, G. and Toni, F. 2007. Programming applications in CIFF. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning*, C. Baral, G. Brewka and J. S. Schlipf, Eds., Tempe, AZ, USA, Lecture Notes in Computer Science 4483, Springer, 284–289.

Mancarella, P., Terreni, G. and Toni, F. 2007. Web sites verification: An abductive logic programming tool. In *Proceedings of the 23rd International Conference on Logic Programming*, V. Dahl and I. Niemelä, Eds., Porto, Portugal, Lecture Notes in Computer Science 4670, Springer, 434–435.

Mancarella, P., Terreni, G. and Toni, F. 2009. Web sites repairing through abduction. *Electronic Notes on Theoretical Computer Science 235*, 137–152.

Marek, W. and Truszczynski, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*. Springer-Verlag, 375–398.

Martelli, A. and Montanari, U. 1982. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems 4*, 2, 258–282.

Mayer, M. C. and Pirri, F. 1993. First order abduction via tableau and sequent calculi. *Bulletin of the IGPL 1*, 1, 99–117.

Mellarkod, V. S. and Gelfond, M. 2008. Integrating answer set reasoning with constraint solving techniques. In *Proceedings of the 9th International Symposium on Functional and Logic Programming*, J. Garrigue and M. V. Hermenegildo, Eds., Ise, Japan, Lecture Notes in Computer Science 4989, Springer, 15–31.

Miller, R. and Shanahan, M. 2002. Some alternative formulations of the event calculus. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, A. C. Kakas and F. Sadri, Eds., Lecture Notes in Computer Science 2408, Springer, London, 452–490.

Niemela, I. and Simons, P. 1997. SMODELS – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning*, J. Dix, U. Furbach and A. Nerode, Eds., London, UK, Lecture Notes in Computer Science 1265, Springer, 421–430.

Pereira, L. M., Aparício, J. N. and Alferes, J. J. 1991. Nonmonotonic reasoning with well founded semantics. In *Proceedings of the Eighth International Conference on Logic Programming*, K. Furukawa, Ed., Paris, France, MIT Press, 475–489.

Sadri, F. and Toni, F. 1999. Abduction with negation as failure for active and reactive rules. In *AI*IA 99: Advances in Artificial Intelligence, Sixth Congress of the Italian Association for Artificial Intelligence*, E. Lamma and P. Mello, Eds., Bologna, Italy, Lecture Notes in Computer Science 1792, Springer, 49–60.

Sadri, F., Toni, F. and Torroni, P. 2002. An abductive logic programming architecture for negotiating agents. In *Logics in Artificial Intelligence, European Conference*, S. Flesca, S. Greco, N. Leone and G. Ianni, Eds., Cosenza, Italy, Lecture Notes in Computer Science 2424, Springer, 419–431.

Shanahan, M. 1989. Prediction is deduction but explanation is abduction. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, N. S. Sridharan, Ed., Detroit, MI, USA, Morgan Kaufmann, 1055–1060.

Simons, P. 2000. *Extending and Implementing the Stable Model Semantics*, Technical Report. Helsinki University of Technology, Espoo, Helsinki, Finland.

SOCS-Consortium. 2002–2005. Societies of computees (SOCS): A computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. IST-2001-32530 [online]. Accessed 22 July 2009. URL: `http://lia.deis.unibo.it/Research/SOCS/`

Terreni, G. 2008a. *The CIFF Proof Procedure for Abductive Logic Programming with Constraints: Definition, Implementation and a Web Application*, PhD Thesis. Università di Pisa, Pisa, Tuscany, Italy.

Terreni, G. 2008b. The CIFF system [online]. Accessed 22 July 2009. URL: `http://www.di.unipi.it/ ~terreni/research.php`

Toni, F. 2001. Automated information management via abductive logic agents. *Telematics and Informatics 18*, 1, 89–104.

Van Gelder, A., Ross, K., and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38*, 3, 620–650.

Van Nuffelen, B. 2004. *Abductive Constraint Logic Programming: Implementation and Applications*, PhD Thesis. K. U. Leuven, Leuven, Belgium.

Wetzel, G., Kowalski, R. A., and Toni, F. 1996. PROCALOG – programming with constraints and abducibles in logic (poster abstract). In *Proceedings of the 13th Joint International Conference and Symposium on Logic Programming*, M. J. Maher, Ed., Bonn, Germany, MIT Press, 535.

Wielemaker, J. 2003. An overview of the SWI-Prolog programming environment. In *Proceedings of the 13th International Workshop on Logic Programming Environments*, F. Mesnard and A. Serebrenik, Eds., Mumbai, India, Report CW371 Katholieke Universiteit Leuven, Department of Computer Science, 1–16. Accessed 22 July 2009. URL: `http://www.swi-prolog.org/`.