A Unifying Framework for ALP, CLP and SQO

Robert A. Kowalski, Gerhard Wetzel, Francesca Toni

Logic Programming Section Technical Report

Department of Computing
Imperial College, London SW7 2BZ, UK
Tel.: +44-171-589-5111
{rak,gw1,ft}@doc.ic.ac.uk
http://www-lp.doc.ic.ac.uk

April 1996

Abstract

This paper presents the *TPCALP framework*, a theorem-proving approach which aims to unify Abductive Logic Programming (ALP), Constraint Logic Programming (CLP) and Semantic Query Optimization (SQO). The framework combines the use of definitions, as in ordinary logic programming, with the use of integrity constraints, as in ALP and SQO. The programmer can choose to represent knowledge in either form subject to the condition that the integrity constraints be "properties" of the definitions. The paper defines a proof procedure for the framework and presents some formal results for the proof procedure with respect to the framework semantics. The proof procedure executes definitions in conventional logic programming goal reduction manner, and integrity constraints in forward reasoning style to check potential answers for consistency. The integrity constraints are used to process goals when the definitions cannot be used, either because they are not accessible (as in ALP and SQO) or because their use would be computationally explosive (as in CLP and more generally).

1 Introduction

This paper presents the *TPCALP framework*, a theorem-proving approach which aims to unify Abductive Logic Programming (ALP, [CoThTo91]), Constraint Logic Programming (CLP, [JaLa87]), and Semantic Query Optimization (SQO, [ChGrMi87]). The new framework is different from other attempts at unifying ALP and CLP [KaMi95].

ALP, CLP, and SQO can all be understood as extending Horn clause definitions

$Head \leftarrow Body$

of standard Logic Programming (LP), which are executed in backward reasoning, goal reduction manner, by integrity constraints

$Conditions \rightarrow Conclusions$

which are executed in forward reasoning style.

For example, consider the problem of configuring a computer system. The possible choices for the components (processor, monitor, memory, operating system, etc.) can be specified by definitions, e.g.

```
processor(pentium) ←
processor(sparc) ←
...
operating_system(os2) ←
operating_system(unix) ←
...
```

which we will generally write in if-and-only-if form and sometimes using a disjunction:

```
processor(X) ↔ ( X = pentium ) ∨ ( X = sparc ) ∨ ...
operating_system(X) ↔ ( X = os2 ) ∨ ( X = unix ) ∨ ...
```

Dependencies, whether system constraints or personal preferences, between the different choices can be represented by integrity constraints, in the form of both positive requirements and denials, e.g.

```
processor(sparc) → operating_system(unix)
processor(sparc) ∧ operating_system(os2) → false
```

In the framework instance corresponding to CLP, a query `processor(X)` ∧ `operating_system(os2)` results in the constraint `X=pentium` being derived as an answer (but not `X=sparc`).

In SQO, the definitions of `processor` and `operating_system` may be contained in the extensional database and be regarded as inaccessible during query optimization. A query `processor(sparc)` $\wedge$ `operating_system(os2)` would be rejected as inconsistent with the integrity constraints without consulting the extensional database.

In ALP, the definitions of `processor` and `operating_system` might be regarded as unknown and the predicates treated as abducibles. Given a query `processor(sparc)`, the answer `processor(sparc)` $\wedge$ `operating_system(unix)` is an abductive explanation which satisfies the integrity constraints (whereas an answer containing `operating_system(os2)` does not).

Integrity constraints, used as forward reasoning rules, are also useful to specify certain kinds of algorithms. For example, the core of the n-queens problem, which is to place n queens on an $n \times n$ chessboard so that they do not attack each other, can be represented by a single integrity constraint:

`queen(A,B)` $\wedge$ `queen(C,D)` $\wedge$ `legalmove((A,B),(C,D))` $\rightarrow$ `false`

This has the procedural interpretation: If one queen is placed at coordinates `(A,B)` and another queen at coordinates `(C,D)`, and the rules of chess allow a queen to move from `(A,B)` to `(C,D)`, then reject this solution. The definition of `legalmove` can be expressed in standard LP form.

The structure of the paper is as follows. Section 2 presents the general TPCALP framework, including a brief discussion of the instances the framework seeks to unify. Section 3 discusses the propertyhood notion of integrity constraints, which is used in the framework. Section 4 defines a declarative semantics and shows how semantics for ALP, CLP and SQO can be obtained as instances. Section 5 presents a proof procedure and section 6 contains some formal results. Possible extensions and efficiency improvements of the proof procedure are mentioned in the concluding section.

This paper builds upon [WeKoTo95] where the focus was on CLP. A related framework, with focus on ALP, has been defined in [Fu95, Fu96]. Applications which have been investigated include job-shop scheduling (see [KoToWe94], [To94]), warehouse location and configuration problems.

2 The TPCALP framework

2.1 General framework

In the TPCALP framework, knowledge is represented both by definitions and by integrity constraints.

Definition 2.1 (Framework language)

- A *definition* is a formula
  $H \leftrightarrow D_1 \vee \ldots \vee D_n,\ n \geq 1$
  where H is an atom, but not `false`, and each D_i is a conjunction of atoms.[1] Variables in H are implicitly universally quantified with scope the entire definition whereas variables in a disjunct D_i which do not occur in H are existentially quantified with scope the disjunct. H is called the head of the definition, $D_1 \vee \ldots \vee D_n$ is called the body. The heads of two different definitions must not unify.
- An *integrity constraint* is a formula
  $A_1 \wedge \ldots \wedge A_m \rightarrow B,\ m \geq 1$
  where B and the A_i are atoms. All variables are universally quantified with scope the entire B is called the conclusion and each A_i a condition of the entire integrity constraint.

Both definitions and integrity constraints must be range-restricted, i.e. all variables in the head must appear in at least one atom in the body, and this atom must not be an equality between two variables.[2]

Definition 2.2 (Predicate types)
There are three kinds of predicates in the TPCALP framework:

- *user-defined* predicates, defined by a theory $\mathcal{T}_u$ of user-provided definitions,
- *built-in* predicates, defined by a theory $\mathcal{T}_b$ of built-in definitions,
- *external* predicates, defined by an inaccessible or unknown theory $\mathcal{T}_e$ of definitions.

External predicates correspond to abducibles in ALP and extensional predicates in SQO.

Built-in predicates, which correspond to constraint predicates in CLP, always include equality and generally also arithmetic inequality ($<$,$\leq$,$\geq$,$>$) and operators (like `plus`). Conceptually, the main difference between built-in

[1] We allow `false` and `true` as atoms. The framework can be extended to definitions with negative literals in the body, as in [Fu96]. For simplicity, we restrict ourselves to Horn clause programs in this paper.

[2] A definition `p(X,Y)` $\leftrightarrow$ `X=Y` is not allowed because it may be rewritten to `p(X,X)` $\leftrightarrow$ `true`.

predicates and user-defined predicates is that the former have fixed definitions, which the user cannot change.

The symbol $\mathcal{T}$ is used to denote the union of the three sub-theories: $\mathcal{T} = \mathcal{T}_u \cup \mathcal{T}_b \cup \mathcal{T}_e$, and $\mathcal{T}'$ refers to the accessible parts of the theory: $\mathcal{T}' = \mathcal{T}_u \cup \mathcal{T}_b$

2.2 Suspension of goals

The computational task (defined formally in section 5) is to reduce a given initial goal to a disjunction of answers which is equivalent to the initial goal in the sense that the goal and the disjunction are satisfied, in $\mathcal{T}$, by the same assignments to the free variables. Such answers are obtained by using definitions to reduce (also called "unfold") atomic goals to equivalent goals in the form of disjunctions.

Definition 2.3 (Goals)
An *initial goal* is a conjunction of atoms with all variables being free. An *implication* is a formula $A_1 \wedge \ldots \wedge A_m \rightarrow B$, $m \geq 1$, where B and the A_i are atoms. A *subgoal* is an atom or an implication. A *simple goal* is a conjunction of subgoals. A *goal* is either a simple goal or a disjunction of goals.

An atomic goal A cannot be reduced ("unfolded") if its definition is inaccessible or unknown (i.e. if A is an atom of an external predicate). Moreover, to control the amount of search non-determinism, A should not be reduced if its predicate has accessible definitions, but variables in A have to be instantiated to unify with the head of one or more definitions. If reducing A were desired in this case, then the definitions can be written accordingly (cf. below after the example).

Definition 2.4 An atom A is *reducible* if there is an accessible definition $H \leftrightarrow D_1 \vee \ldots \vee D_n$ in $\mathcal{T}'$ such that A is an instance of H (i.e. $A = H\sigma$ for some substitution σ). Otherwise A is *suspended.*

Note that, since the heads of two definitions must not unify, there is one and only one definition for A if A is reducible.

Example 2.1 Let the user-defined predicate `p` be defined by
`p(X,a)` $\leftrightarrow$ `q(X)`

$\texttt{p(X,b)} \leftrightarrow \texttt{r(X)}$

Then p(X,a) and p(a,b) are reducible (and can be reduced to q(X) and r(a) respectively), whereas p(X,Y) and p(a,Y) are suspended. If the two given definitions are the only definitions for the predicate p, then p(X,c) is also suspended. If the user wants it to be reduced to false, an explicit definition $\texttt{p(X,c)} \leftrightarrow \texttt{false}$ or an integrity constraint $\texttt{p(X,c)} \rightarrow \texttt{false}$ should be added.

We assume that the definition of a built-in predicate p is of the form

$\texttt{p(t}_1\texttt{,}\ldots\texttt{,t}_m\texttt{,X}_1\texttt{,}\ldots\texttt{,X}_n\texttt{)} \leftrightarrow \texttt{X}_1\texttt{=c}_1 \wedge \ldots \wedge \texttt{X}_n\texttt{=c}_n$

where $m, n \geq 0$, $m + n > 0$ and $\texttt{t}_1,\ldots,\texttt{t}_m$ are ground terms. Thus the values of the *output parameters* $\texttt{X}_1,\ldots,\texttt{X}_n$ are uniquely determined if all *input parameters* $\texttt{t}_1,\ldots,\texttt{t}_m$ are ground.

In the case of equality and inequality we assume that there are no output parameters, and so the definition is an enumeration of all ground facts (0=0, 0<1, etc.). Arithmetic operators have two input and one output parameter ($\texttt{plus(1,2,X)} \leftrightarrow \texttt{X=3}$, etc.). Therefore, the atoms X<Y, X>0 and plus(2,X,Y) are all suspended, but 1>0 and plus(2,2,X) are reducible (to true and X=4, respectively). Atoms like 1 < 0, which are false in the intended model of "<", are suspended, but can generate false if appropriate integrity constraints are used (e.g. $\texttt{X < Y} \wedge \texttt{Y < X} \rightarrow \texttt{false}$).

The user can control which atoms of user-defined predicates are suspended by writing definitions in more or less *homogenized form* [3]: if a predicate is defined by an enumeration of all ground facts, then any of its atoms with variables in it is suspended, whereas atoms of predicates which have only one completely homogenized definition are never suspended.

User-defined suspension of atomic subgoals is an important feature of concurrent logic programming languages where it is used to eliminate search non-determinism entirely. In our framework, the user controls the extent of search non-determinism by the form in which the definitions of user-defined predicates are written.

Suspended atoms cannot be reduced by means of definitions in $\mathcal{T}$ and are processed instead by means of integrity constraints. This process may add to the goal redundant (surrogate) subgoals which are reducible and thus enable the reduction using definitions to proceed. Therefore, integrity constraints

[3] A definition is in homogenized form iff all terms in the head are mutually distinct variables.

are used as approximations of the definitions in $\mathcal{T}$. To guarantee that they are sound approximations, integrity constraints will be required to be *properties* of the definitions, as defined in section 3.

2.3 Framework instances

In standard LP, ALP, CLP and SQO there is no suspension of user-defined predicates. This corresponds to the case where all user-defined definitions are written in homogenized form.

In standard **LP** the only built-in predicate is equality, and there are no integrity constraints. Since (non-ground) equality atoms are the only suspended atoms, the computational task is to find answers to the initial goal, in the form of a conjunction of equalities assigning values to the free variables in the initial goal.

In **ALP**, the external predicates are abducible, and equality is the only built-in predicate. Since abducibles and (non-ground) equality atoms are the only suspended atoms, the computational task is to find an explanation for the initial goal in the form of a conjunction of abducibles and equalities which implies the initial goal and satisfies the integrity constraints.

In **SQO**, the external predicates are extensional database (EDB) predicates. For the purposes of query optimization, their definitions are considered inaccessible. The user-defined predicates are intensional database predicates, and integrity constraints approximate the extensional part. The only built-in predicate is equality. The computational task is to find answers to the initial goal in the form of conjunctions of atoms of EDB predicates (plus equality). Since the EDB is inaccessible and the integrity constraints only approximate the EDB, such answers only approximate those obtained by submitting them to the EDB.

In **CLP** there are no external predicates. Built-in predicates are called constraints. Rather than employing a built-in constraint solver, as in most frameworks modelled after the $CLP(X)$ scheme of [JaLa87], the constraint solver is programmed by means of built-in constraint definitions augmented with explicit integrity constraints (to process suspended constraint atoms). This use of integrity constraints is similar to Frühwirth's constraint handling rules [Fr95]. Because integrity constraints can contain both user-defined and built-in predicates, user-defined predicates may be regarded as user-defined constraints. In principle it is also possible to integrate, for efficiency reasons, built-in constraint solvers into the TPCALP framework; conceptually they correspond to built-in integrity constraints. The treatment of equality in the proof procedure (see section 5) may be understood in this way.

3 Integrity constraint satisfaction

The integrity constraints $\mathcal{IC}$ can be thought of as approximations of the definitions in $\mathcal{T}$. The proof procedure checks possible answers to an initial query for satisfaction of the integrity constraints.

The two simplest views of integrity constraint satisfaction are the *theoremhood view*, requiring the integrity constraints to be properties of the definitions in the sense that they are true in all models of $\mathcal{T}$, i.e. $\mathcal{T} \models \mathcal{IC}$, and the *consistency view*, requiring the integrity constraints to be true in just one model of $\mathcal{T}$.

Neither of these two views takes into account the distinction between those models of $\mathcal{T}$ which are actually intended and those which are not. This paper proposes an alternative view of integrity constraint satisfaction which requires the integrity constraints to be true in the intended models of the theory:[4]

Definition 3.1 A first-order sentence Φ is a *property* of a set of definitions $\mathcal{T}$ iff Φ is true in all intended models of $\mathcal{T}$, written $\mathcal{T} \models_{int} \Phi$. If P is a set of sentences, $\mathcal{T} \models_{int} P$ denotes that every sentence in P is a property of $\mathcal{T}$.

Definition 3.2 If $\mathcal{T}$ is a set of definitions and $\mathcal{IC}$ a set of integrity constraints such that $\mathcal{T} \models_{int} \mathcal{IC}$, then $\mathcal{T}$ is said to *satisfy* $\mathcal{IC}$ according to the *propertyhood view* of integrity constraint satisfaction.

If $\mathcal{T}$ is a Horn clause theory, then the only intended model is the unique minimal Herbrand model. In the general case, when $\mathcal{T}$ is a normal logic program containing negative literals, different notions of intended model may be considered. Many of these notions coincide for special cases, such as the case where $\mathcal{T}$ is acyclic [ApBe90]. For simplicity, in this paper we restrict ourselves to the case where $\mathcal{T}$ consists of Horn clauses only.

[4]It could be argued that this view is similar to the metalevel [Ko90, EsKo89] and epistemic [Re90] views of integrity constraint satisfaction. It also bears some similarity to the view adopted in [KaMa90].

Example 3.1 $\mathcal{T}$: `p` $\leftrightarrow$ `p`
The integrity constraint

`p` $\rightarrow$ `false`

is satisfied according to the consistency and propertyhood views, but not according to the theoremhood view. However, the integrity constraint

`true` $\rightarrow$ `p`

is satisfied according to the consistency view, but not according to the propertyhood and theoremhood views.

Example 3.2 Suppose $\mathcal{T}$ consists of all clauses of the form

`i < j` $\leftrightarrow$ true

where `i` and `j` are integers and `i` is smaller than `j`. Then the following sentences are all properties of $\mathcal{T}$, and therefore are integrity constraints satisfied according to the propertyhood view.

`X < X` $\rightarrow$ `false` (irreflexivity)

`X < Y` $\wedge$ `Y < X` $\rightarrow$ `false` (anti-symmetry)

`X < Y` $\wedge$ `Y < Z` $\rightarrow$ `X < Z` (transitivity)

None of these integrity constraints are satisfied according to the theoremhood view (whether $\mathcal{T}$ is formulated in if- or iff-form).[5]

The same sentences are also properties of the definition

$$\begin{aligned}\mathtt{S_1 < S_2} \leftrightarrow\ & [\ \mathtt{S_1{=}s_0} \wedge \mathtt{S_2{=}result(A,S_3)}\ ] \vee \\ & [\ \mathtt{S_1{=}result(A_1,S_3)} \wedge \mathtt{S_2{=}result(A_2,S_4)} \wedge \mathtt{S_3{<}S_4}\ ]\end{aligned}$$

of the user-defined predicate `<` in the situation calculus (see e.g. [KoSa94]).

If a sentence is true in all models of a theory $\mathcal{T}$, then it is true in all its intended models. Thus sentences which are logically implied by $\mathcal{T}$ are also properties of $\mathcal{T}$ ("$\models$" $\Rightarrow$ "$\models_{int}$"), but not vice versa, although for a conjunction A of atoms in which all variables are existentially quantified the two notions are equivalent: $\mathcal{T} \models_{int} \tilde{\exists} A$ if and only if $\mathcal{T} \models \tilde{\exists} A$.

Finally, note that if a sentence is a property of a Horn theory, then the sentence is also consistent with the theory.

In the rest of the paper $\mathcal{T} \models_{int} \mathcal{IC}$ will always be assumed.

4 Declarative semantics

[5] However, they can all be proved using an appropriate form of induction.

Definition 4.1 A is a *weak answer* to the initial goal G_0 if

(I1) A is a conjunction of suspended atoms

(I2) $\mathcal{T} \models_{int} \tilde{\forall}[G_0 \leftrightarrow A \vee Rest]$ for some formula $Rest$

(I3) $\mathcal{IC} \cup CET \cup \tilde{\exists}A$ is consistent, where CET denotes the Clark Equality Theory [Cl78]

A is a *good answer* to G_0 if (I1) and (I2) hold and

(I3a) $\mathcal{T}' \cup \mathcal{IC} \cup CET \cup \tilde{\exists}A$ is consistent

A is an *ideal answer* to G_0 if (I1) and (I2) hold and

(I3b) $\mathcal{T} \models_{int} \tilde{\exists}A$[6]

Condition (I1) makes explicit the requirement that goals should be reduced as far as possible. Condition (I1) generalizes the requirements of A being a conjunction of equalities in the LP case, a conjunction of abducibles in the ALP case, a conjunction of EDB atoms in the SQO case, and a conjunction of constraints in the CLP case.

Condition (I2) means that, in the minimal Herbrand model of $\mathcal{T}$, G_0 and $A \vee Rest$ are true for the same values of the free variables in G_0, i.e. G_0 and $A \vee Rest$ are equivalent in the intended models of $\mathcal{T}$. It can be easily shown that $\tilde{\forall}[G_0 \leftrightarrow A \vee Rest]$ is a property of $\mathcal{T}$ if and only if $\tilde{\forall}[G_0 \leftarrow A]$ is one. The latter formulation of (I2) resembles both the usual characterization of explanation in ALP and the requirement imposed on the final state of the constraint store in CLP. The use of the universal quantifiers relaxes the restriction in ALP that abductive explanations contain no variables.

The different variants of the third condition are introduced because the suspension of subgoals restricts the extent to which the satisfaction of integrity constraints can be determined. The case of SQO illustrates that it is not possible to insist that answers always be ideal: it cannot be expected in general that the final transformed query is satisfied in the EDB, so it is reasonable to insist only that good answers in the sense of condition (I3a) are computed, i.e. answers that are consistent with the accessible part of the theory and with the integrity constraints. This condition also corresponds to the condition imposed on abductive explanations in ALP semantics.

[6] If the framework is extended to allow negation, then (I3b) should be changed to $\mathcal{T} \not\models_{int} \neg\tilde{\exists}A$, which is equivalent to (I3b) in the Horn clause case.

Extending the requirement that the answer satisfies the integrity constraints to CLP and to the general case is difficult because user-defined and built-in predicates may also be suspended and may prevent the proof procedure from checking the consistency of suspended subgoals with $\mathcal{T}'$. The notion of weak answer is introduced to cover these cases: suspended subgoals are not checked for satisfiability with respect to $\mathcal{T}'$, but only with respect to the integrity constraints (note that $\mathcal{T}_b \models_{int} CET$ so that the axioms of CET can be regarded as integrity constraints for equality; see the remark after Definition 5.6).

Obviously, every good answer is a weak answer, and $\mathcal{T} \models_{int} \mathcal{IC} \cup CET$ ensures that every ideal answer is a good answer. The converse does not hold in either case.

Example 4.1 (Difference between answer notions)
If $\mathcal{IC}$ is empty, then

$A =$ `1 < X` $\wedge$ `X < 0`

is a good answer, as there are (non-intended) models of $\mathcal{T}$ in which A is true. But A is not an ideal answer, as A is false in the minimal Herbrand model (of the definition of "`<`").

If $\mathcal{IC}$ contains the transitivity and irreflexivity rules from Example 3.2, then A is no longer a good answer (because A, together with transitivity, implies `1 < 0` which, together with `0 < 1` from the theory $\mathcal{T}'$, implies `1 < 1` which, together with irreflexivity, yields a contradiction). But A is still a weak answer (because `0 < 1` is not implied by $\mathcal{IC}$).

5 Proof procedure

Definition 5.1 Let G_0 be an initial goal. A *derivation* of G_n from G_0 is a finite sequence $G_0, G_1, \ldots, G_n$ where G_1 is obtained from G_0 by conjoining all integrity constraints in $\mathcal{IC}$ to G_0, and, for $1 \leq i < n$, G_{i+1} is obtained from G_i by applying one of the following operations (defined later):

- *unfolding* a reducible atom, using a definition in $\mathcal{T}'$
- *logical equivalence transformation*
- *propagation*, which is a form of resolution between subgoals
- *equality rewriting*

Definition 5.2 If there is a derivation of $G_n = (A \wedge Imp) \vee Rest$ from the initial goal G_0, then A is a *computed answer* to G_0 if and only if

(C1) $A \neq false$

(C2) A consists of atoms only; Imp consists of implications only

(C3) the operations of the proof procedure have been exhaustively applied to $A \wedge Imp$

Imp may consist of integrity constraints in $\mathcal{IC}$ and implications obtained by resolving with integrity constraints in $\mathcal{IC}$ and by unfolding atoms in such resolvents or integrity constraints.

Unfolding and logical equivalence transformation reduce the initial goal to the form $A \vee Rest$ so that a computed answer satisfies conditions (I1) and (I2). Propagation and equality rewriting ensure that a computed answer A is consistent with $\mathcal{IC} \cup CET$ and therefore satisfies at least condition (I3). (Unfolding atoms in conditions of integrity constraints is needed to verify condition (I3a)).

In the simple case where integrity constraints contain only external predicates (as in SQO), the derivation of computed answers can be split neatly into two phases. In the first phase unfolding and logical equivalence transformation are used to satisfy conditions (I1) and (I2); in the second phase propagation and equality rewriting are used to satisfy condition (I3). Both to deal with the more general case and to improve efficiency by detecting violations of condition (I3) earlier, the operations should be interleaved.

Definition 5.3 (*Unfolding*)
Let A be a reducible atomic subgoal or a reducible atom in the condition of an implication in G_i. If there is a definition $H \leftrightarrow D_1 \vee \ldots \vee D_m$ in $\mathcal{T}'$ and a substitution σ such that $A = H\sigma$, then replace A by $(D_1 \vee \ldots \vee D_m)\sigma$.

Definition 5.4 A *logical equivalence transformation* is the application of one of the following equivalences as a rewrite rule to replace a formula which matches the left-hand side by the formula on the right-hand side:

- $(A \vee B) \wedge C \;\leftrightarrow\; (A \wedge C) \vee (B \wedge C)$ (*splitting*)
- $(A \vee B) \rightarrow C \;\leftrightarrow\; (A \rightarrow C) \wedge (B \rightarrow C)$ (*normalization of implications*)
- $A \wedge \texttt{true} \leftrightarrow A$, $A \wedge \texttt{false} \leftrightarrow \texttt{false}$, $A \vee \texttt{true} \leftrightarrow \texttt{true}$, $A \vee \texttt{false} \leftrightarrow A$

Propagation is easiest to define for simple goals (see Definition 2.1). Any goal can be transformed into a disjunction of simple goals by using logical equivalences.[7]

Definition 5.5 (*Propagation*)
Propagation adds to a simple goal, which contains the two subgoals
$\mathtt{p}(s_1, \ldots, s_n) \wedge Cond \rightarrow Conc$ and
$Susp \rightarrow \mathtt{p}(t_1, \ldots, t_n)$, the "resolvent"
$Cond \wedge Susp \wedge (s_1 = t_1 \wedge \ldots \wedge s_n = t_n) \rightarrow Conc$.

In the simplest case, $Susp$ is empty, i.e. $Susp \equiv \mathtt{true}$. This case corresponds to one way of implementing hyper-resolution (by P_1-deduction). In the more general case, $Susp$ is non-empty, because it consists of suspended atoms, which cannot be unfolded.

This definition of propagation is inefficient and prevents the proof procedure from terminating in many cases in which termination should be expected. Restrictions similar to those employed in [Fu96] as well as techniques from automated theorem proving can be used to restrict propagation and to overcome many of these inefficiencies. A discussion of such techniques is beyond the scope of this paper. Some of the more important issues are mentioned in the concluding section.

Equality atoms are introduced into goals both by propagation and by unfolding. Equality rewriting deals with such equalities, and it is defined via the following rewrite rules, taken from [Fu96] where they have been adapted from [MaMo82].

Definition 5.6 *Equality rewriting* is the application of one of the equivalences (EQ1)–(EQ5) as rewrite rules or of the term substitution rule (EQ6):

(EQ1) $f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n) \leftrightarrow t_1 = s_1 \wedge \ldots \wedge t_n = s_n$ ($n \geq 1$, f any function symbol)

(EQ2) $f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m) \leftrightarrow \mathtt{false}$ ($n, m \geq 0$; f, g any distinct function or constant symbols)

(EQ3) $x = x \leftrightarrow \mathtt{true}$

(EQ4) $y = x \leftrightarrow x = y$, if either x is a variable and y is not or both x and

[7] It is possible to define a more efficient form of propagation similar to "non-clausal resolution" [Mu82], which also applies to goals not in this form, see [WeKoTo95].

y are variables, but x is universally quantified and y is not

(EQ5) $x = t \leftrightarrow$ `false` where the term t contains x

(EQ6) If $x = t$ is an atomic subgoal of a simple goal G_s and none of (EQ1)–(EQ5) can be applied to $x = t$, then replace all other occurrences of the same variable x in G_s by t (not including $x = t$). If $x = t$ occurs as a condition of an implication in G_s, (EQ1)–(EQ5) cannot be applied to $x = t$, and x is a universally quantified variable, then replace all occurrences of the same variable x within the implication by t (including $x = t$).

The rewrite rules (EQ1)–(EQ5) and the substitution rule (EQ6) constitute a sound and complete implementation of the CET axioms (cf. [Fu96] and [MaMo82]). It can be shown that the CET axioms are all properties of the built-in definition of equality, i.e. $\mathcal{T}_b \models_{int} CET$. Therefore, equality rewriting as defined above may be regarded as an optimized form of propagation with (built-in) integrity constraints where the redundant half of the equivalence is deleted after propagation.

Each of the operations of the proof procedure, including the step from G_0 to G_1, derives an equivalent formula in $\mathcal{T}$:

Lemma 5.1 For $0 \le i \le n-1$, $\mathcal{T} \models_{int} \tilde{\forall}[G_i \leftrightarrow G_{i+1}]$.

Corollary 5.1 $\mathcal{T} \models_{int} \tilde{\forall}[G_0 \leftrightarrow G_n]$, if there is a derivation of G_n from G_0.

The following example illustrates the behaviour of the proof procedure in a very simple case. The reader may want to refer to [KoToWe94] and [WeKoTo95] for more elaborate examples and applications.

Example 5.1
$\mathcal{T}_u$: `p(X)` $\leftrightarrow$ `q(X)` $\vee$ `r(X)`
$\mathcal{IC}$: `q(a)` $\rightarrow$ `false`

G_0 : `p(a)`
G_1 : `p(a)` $\wedge$ [`q(a)` $\rightarrow$ `false`] (added $\mathcal{IC}$)
G_2 : (`q(a)` $\vee$ `r(a)`) $\wedge$ [`q(a)` $\rightarrow$ `false`] (unfolding) ($\downarrow$ splitting)
G_3 : (`q(a)` $\wedge$ [`q(a)` $\rightarrow$ `false`]) $\vee$ (`r(a)` $\wedge$ [`q(a)` $\rightarrow$ `false`])
G_4 : (`q(a)` $\wedge$ [`q(a)` $\rightarrow$ `false`] $\wedge$ `false`) $\vee$ (...) (propagation)
G_7 : `r(a)` $\wedge$ [`q(a)` $\rightarrow$ `false`] (3x logical equivalence transformation)
Computed answer: `r(a)`

6 Formal results

This section presents soundness and completeness results for the proof procedure with respect to the different answer notions of the declarative semantics. Proofs are omitted due to space limitations.

Theorem 6.1 (Weak affirmation soundness)
If A is a computed answer to G_0, then A is a weak answer to G_0.

Theorem 6.2 (Refutation soundness)
If there is a derivation of `false` from G_0, then G_0 has no good answers, and therefore no ideal answers.

Stronger affirmation soundness results are obtainable if additional assumptions are made. The following two theorems hold for the LP, ALP and SQO cases, i.e. under the assumption that equality is the only built-in predicate and that atoms of user-defined predicates are always reducible.

Theorem 6.3 (Affirmation soundness for LP, ALP and SQO)
In the LP, ALP and SQO cases, every computed answer to an initial goal G_0 is a good answer to G_0.

The next theorem shows that computing answers corresponds to "discovering" (or abducing) possible definitions $\mathcal{T}_e^*$ for the external predicates, as in ALP. However, the case of SQO shows that it cannot be expected that $\mathcal{T}_e^*$ corresponds to the actual EDB.

Theorem 6.4 (Approximation of $\mathcal{T}_e$)
In the LP, ALP and SQO cases, if A is a computed answer, then there exists a theory $\mathcal{T}_e^*$ such that $\mathcal{T}' \cup \mathcal{T}_e^* \models_{int} \tilde{\exists} A$ and $\mathcal{T}' \cup \mathcal{T}_e^* \models_{int} \mathcal{IC}$.

If the integrity constraints provide a complete axiomatization of the theory, then the proof procedure returns only ideal answers in any case.

Definition 6.1 $\mathcal{IC}$ is *satisfaction complete* if, for every conjunction A of suspended atoms, either $\mathcal{IC} \cup CET \models \tilde{\exists} A$ or $\mathcal{IC} \cup CET \models \neg\tilde{\exists} A$.

Since $\mathcal{T}$ is a Horn clause theory, for every conjunction A of suspended atoms either $\mathcal{T} \models_{int} \tilde{\exists} A$ or $\mathcal{T} \models_{int} \neg\tilde{\exists} A$. Moreover, as observed in the previous section, $\mathcal{T}_b \models_{int} CET$.

Theorem 6.5 If $\mathcal{IC}$ is satisfaction complete, conditions (I3), (I3a) and (I3b) of Definition 4.1 are equivalent.

Corollary 6.1 If $\mathcal{IC}$ is satisfaction complete, then every computed answer is an ideal answer.

Only a weak completeness result can be obtained without imposing restrictions on propagation which, in its present form, may prevent the proof procedure from terminating in many cases.

Definition 6.2 A is a *computed possible answer* to G_0 if there exists a derivation of G_n from G_0 such that $G_n = (A \wedge Imp) \vee Rest$, where $A \neq \texttt{false}$ consists of atoms only and Imp consists of implications only.

Definition 6.3 A is a *minimal good answer* to G_0 if A is a good answer to G_0 and no proper subconjunction of A is a good answer to G_0.

Theorem 6.6 (Weak completeness)
Let A be a minimal good answer to G_0. Then there exists a computed possible answer A' to G_0 such that $\tilde{\forall}(A' \rightarrow A)$.

7 Conclusion and future work

The proof procedure as presented in section 5 can be made more efficient in several respects. In particular, the proof procedure can be refined to ensure termination, at least in cases where it can reasonably be expected, in order to prove stronger completeness results. This can be achieved by improving propagation, by imposing priorities between the operations and by adding further operations which are compatible with the semantics.

Propagation can be improved by employing one or more of the several refinements of resolution documented in the theorem-proving and logic programming literature. For example, a useful restriction, which builds upon the set-of-support strategy, is to allow only one-way unification in propagation and maybe even prohibit (or at least delay) the resolution of two literals which both contain universally quantified variables (this may be justified by the range-restrictedness of the integrity constraints). Other useful strategies include hyper-resolution, i.e. propagate only if all conditions of an integrity constraint can be resolved away at once, locking and selection of literal restrictions.

Propagation can also be enhanced by allowing it to perform non-clausal resolution, thus propagating across disjunct boundaries and reducing the number of splitting operations. An example of such propagation is given in [WeKoTo95].

The proof procedure can also be enhanced by adding a *deletion* operation which removes logically redundant subgoals which are no longer needed (e.g., by applying transitivity, $X > 0$ can be deleted in the presence of $X > 1$). Any integrity constraint used for propagation can, in principle, also be used for deletion. Therefore, conditions under which an integrity constraint should be used for propagation and when it should be used for deletion need to be established.

The framework can be extended to allow negative literals in definitions and integrity constraints. Initial steps of this kind have been taken by [Fu96], focusing on the ALP case.

More work is required to obtain soundness and completeness results for the general case allowing suspension of atoms of user-defined predicates. Ultimately, we hope to be able to also integrate concurrent (C)LP [SaRi90] into the framework.

Finally, it needs to be emphasized that it is not the purpose of our framework to render the special cases of ALP, CLP and SQO unnecessary. Rather, our main aim is to generalize these cases with the aim of identifying new cases not captured by existing approaches.

Acknowledgements

We are grateful to Eric Fung for many helpful discussions. This work was partly supported by Fujitsu Research Laboratories.

References

[ApBe90] Apt, K. R.; Bezem, M.: *Acyclic programs*, in: Warren, D. H. D.; Szeredi, P. (eds.): *Proc. of the 7^{th} Int. Conf. on Logic Programming*, pp. 579–597, MIT Press 1990

[ChGrMi87] Chakravarthy, U. S.; Grant, J.; Minker, J.: *Foundations of Semantic Query Optimization for Deductive Databases*, in: Minker, J. (ed.): *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann 1987

[Cl78] Clark, K. L.: *Negation as failure*, in: Gallaire, H.; Minker, J. (eds.): *Logic and Data Bases*, pp. 292–322, Plenum Press 1978

[CoThTo91] Console, L.; Theseider Dupré, D.; Torasso, P.: *On the Relationship between Abduction and Deduction*, Journal of Logic and Computation 1 (5), pp. 661-690, 1991

[EsKo89] Eshghi, K.; Kowalski, R. A.: *Abduction Compared with Negation by Failure*, in: Levi, G.; Martelli, M. (eds.): *Proc. of the 6^{th} Int. Conf. on Logic Programming*, pp. 234–255, MIT Press 1989

[Fr95] Frühwirth, T.: *Constraint Handling Rules*, in: Podelski, A. (ed.): *Constraint Programming: Basic and Trends*, LNCS 910, Springer Verlag 1995

[Fu95] Fung, T. H.: *Abduction with Clark Completion*, in: Sterling, L. (ed.): *Proc. of the 12^{th} Int. Conf. on Logic Programming*, p. 817 (poster), MIT Press 1995

[Fu96] Fung, T. H.: *Abduction by Deduction*, PhD Thesis, Imperial College 1996

[JaMa94] Jaffar, J.; Maher, M.: *Constraint Logic Programming: A Survey*, Journal of Logic Programming 19/20, 1994

[KaMa90] Kakas, A. C.; Mancarella, P.: *Abductive Logic Programming*, in: Marek, W.; Nerode, A.; Pedreschi, D.; Subrahmanian, V. S. (eds.): *Proc. of the NACLP Workshop on Non-Monotonic Reasoning and Logic Programming*, pp. 49–61, 1990

[KaMi95] Kakas, A. C.; Michael, A.: *Integrating Abductive and Constraint Logic Programming*, in: Sterling, L. (ed.): *Proc. of the 12^{th} Int. Conf. on Logic Programming*, pp. 399–413, MIT Press 1995

[Ko90] Kowalski, R. A.: *Problems and Promises of Computational Logic*, in: Lloyd, J. W.: *Proc. of the Symp. on Computational Logic*, Springer Verlag 1990

[KoSa94] Kowalski, R. A.; Sadri, F.: *The Situation Calculus and Event Calculus Compared*, in: Bruynooghe, M. (ed.): *Proc. of the Int. Logic Programming Symposium*, pp. 539–553, MIT Press 1994

[KoToWe94] Kowalski, R. A.; Toni, F.; Wetzel, G.: *Towards a declarative and efficient glass-box CLP language*, in: Fuchs, N. E.; Gottlob, G. (eds.): *Proc. of the 10th Workshop Logische Programmierung*, 1994

[MaMo82] Martelli, A., Montanari, U.: *An efficient unification algorithm*, ACM Trans. on Prog. Lang. and Systems 4 (2), pp. 258–282, 1982

[Mu82] Murray, N. V.: *Completely non-clausal theorem proving*, Artificial Intelligence 18 (1), pp. 67–85, 1982

[Re90] Reiter, R.: *On asking what a database knows*, in: Lloyd, J. W.: *Proc. of the Symp. on Computational Logic*, Springer Verlag 1990

[SaRi90] Saraswat, V. A.; Rinard, M.: *Concurrent Constraint Programming*, Proc. of the 17$^{\text{th}}$ ACM Symp. on the Princ. of Progr. Lang. pp. 232–245, 1990

[To94] Toni, F.: *A theorem-proving approach to job-shop scheduling*, Technical Report, Imperial College, 1994

[WeKoTo95] Wetzel, G.; Kowalski, R. A.; Toni, F.: *A Theorem-Proving Approach to CLP*, in: Krall, A.; Geske, U. (eds.): *Proc. of the 11th Workshop Logische Programmierung*, pp. 63–72, 1995