A Theorem-Proving Approach to CLP

Gerhard Wetzel Robert A. Kowalski Francesca Toni
Department of Computing
Imperial College, London SW7 2BZ, UK
{gw1,rak,ft}@doc.ic.ac.uk
20 September 1995

1 Introduction

This paper proposes a *T*heorem-*P*roving approach to *C*onstraint *L*ogic *P*rogramming, the *TPCLP framework*. The framework combines the use of definitions, as in ordinary logic programming, with the use of integrity constraints, rather than built-in constraint solvers, for constraint handling. Thus the programmer can choose to represent knowledge either in definition or integrity constraint form. Definitions are executed in conventional logic programming goal reduction manner, whereas integrity constraints are executed in forward reasoning style.

Our use of integrity constraints is similar to Frühwirth's Constraint Handling Rules [Fr95]. However, in contrast to [Fr95], if-and-only-if form is used for definitions, and disjunctions and existential quantifiers are represented explicitly. Another distinction is that we seek to develop an approach to CLP which incorporates Abductive Logic Programming (ALP) (see [CoThTo91]), and Semantic Query Optimization (SQO) (see [ChGrMi87]) as additional instances. A related framework has been defined in [Fu95].

The n-queens problem (for $n = 4$) will be used as an example throughout the paper. Applications which have been studied include job-shop scheduling and warehouse location problems (see [KoToWe94] and [WeKoTo95]).

2 The TPCLP framework

In the TPCLP framework, knowledge is represented both by definitions and by integrity constraints:

- *if-and-only-if definitions* (iff definitions): $H \leftrightarrow D_1 \vee \ldots \vee D_n$
  where H is an atom and each D_i is a conjunction of atoms (for simplicity, negative literals are not considered in this paper). Quantifiers are implicit: variables in the D_i which do not appear in H are existentially quantified on the right side, all other variables are universally quantified over the entire definition.

- *integrity constraints*: $A_1 \wedge \ldots \wedge A_m \rightarrow B$
  where B and the A_i are atoms. All variables in B must appear in some A_i (i.e. the clauses must be range-restricted) and are (implicitly) universally quantified over the entire implication.

The integrity constraints are required to be *properties* of the iff definitions. A sentence is a property of a Horn clause program, whether in iff or in if form, if and only if it is true in the minimal Herbrand model of the program.[1] For example, transitivity:

[1] The definition of property can be extended to frameworks allowing negation by requiring the sentence to be true in *every* intended model or "extension", which depends on the chosen semantics.

ancestor(X,Y) $\wedge$ ancestor(Y,Z) $\rightarrow$ ancestor(X,Z)

is a property of the program

ancestor(X,Y) $\leftrightarrow$ parent(X,Y) $\vee$ $\exists$ Z [parent(X,Z) $\wedge$ ancestor(Z,Y)]

no matter how parent is defined. Irreflexivity and anti-symmetry:

parent(X,X) $\rightarrow$ false

parent(X,Y) $\wedge$ parent(Y,X) $\rightarrow$ false

are properties of any meaningful definition of the parent relation.

According to the definition of "property", the only-if-halves of the iff definitions are properties of the if-halves of the definitions. $\mathcal{T} \models_m \mathcal{S}$ denotes that the sentences in S are properties of the iff definitions $\mathcal{T}$ in the defined sense, where the subscript m is added to avoid confusion with ordinary logical implication.

There are three kinds of predicates in the TPCLP framework:

- *user-defined* predicates, defined by user-provided iff definitions
- *external* predicates, defined by inaccessible or unknown iff definitions
- *built-in* predicates, implicitly defined by built-in iff definitions

User-defined iff definitions are required to be in *homogenized* form, where the only terms occurring in the head, H, are variables, all of which are mutually distinct. External predicates include both extensional predicates in deductive databases (where, for SQO purposes, the extensional database is regarded as inaccessible) and abducibles in ALP. Built-in predicates include equality and arithmetic inequality. The built-in definition of equality can be understood as assigning the meaning, $t = t$, to all ground terms t. With this understanding, the axioms of Clark's Equality Theory are integrity constraints, which are properties of the definition. They are "built-in" in the sense that they are assumed to be part of the integrity constraints. Definitions and integrity constraints for inequality and other predicates might similarly be made available by means of system libraries.

The *initial goal* is a conjunction of atoms; all variables occurring in the initial goal are free (not quantified). Informally speaking, the computational task is to reduce the 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 the theory consisting of the iff definitions, by the same assignments to the free variables. Such answers are obtained by using definitions to reduce (also called *unfold*) atomic goals to equivalent disjunctions of goals. A disjunct is an answer if none of its atoms can be reduced but it satisfies the integrity constraints. An atom cannot be reduced if its predicate is external. In such a case, the integrity constraints can be regarded as an approximation to the definition, and satisfaction of the integrity constraints is a necessary, but not a sufficient condition that the answer corresponds to a solution to the initial goal.

An atom is irreducible also if its predicate is built-in and is insufficiently instantiated to be evaluated (i.e. unfolded using the built-in definitions, e.g. X = Y, X > 0, plus(2,X,Y)). In this case, integrity constraints may be able to detect that a conjunction of such goals is unsatisfiable.

Returning irreducible goals as answers not only prevents generating a large and possibly infinite set of explicit answers (e.g. X > 0 is not unfolded to X = 1 $\vee$ X = 2 $\vee$ ... in the integer case), but in many cases produces more informative "intensional" answers, in the sense that it captures a common property of the individual answers. Consider, for example, an employee database and a query asking for employees `X` who are entitled to certain benefits such as bonus payments or maternity leave. Returning `manager(X)` or `female(X)` as an answer to such a query may be more informative than a disjunction of all managers or female employees. Moreover, such intensional answers remain correct even if the database is changed. For these reasons, it is useful to allow programmers to associate mode declarations with user-defined predicates and to *suspend* the unfolding of atoms which are insufficiently instantiated according to the mode declarations, e.g.

```
suspend female(X) until ground(X)
```

If a list of all ground instances is required, the suspension declaration can simply be removed.

User-defined suspension of atomic subgoals is an important feature of concurrent logic programming languages. It reduces, and in some cases eliminates entirely, search non-determinism. In the remainder of the paper, suspension of all three kinds of atomic subgoals is treated uniformly by assuming that all atomic formulae are classified, in one way or another, as *suspended* or *unsuspended*.

4-queens example

In the n-queens problem, n queens have to be placed on an $n \times n$ chessboard in such a way that they do not attack each other. For $n = 4$ the initial goal is

```
place(1) ∧ place(2) ∧ place(3) ∧ place(4)
```

where `place` is user-defined by the iff definition

```
place(Col) ↔ queen(Col,1) ∨ queen(Col,2) ∨ queen(Col,3) ∨ queen(Col,4)
```

and `queen(Col,Row)` is an external predicate, expressing that there is a queen in column `Col` and row `Row`. The initial goal states that one queen has to be placed in each column.

A single integrity constraint is sufficient to solve the problem:

```
queen( Col, Row ) ∧ move( Col, Row, C, R ) ∧ queen( C, R ) → false
```

The rule states that two queens must not be able to move to each other's position, i.e. they must not be able to attack each other. The predicate `move` is user-defined by an iff definition:

`move(Col,Row,C,R)` ↔ "chess rules allow a queen to move from `(Col,Row)` to `(C,R)`"

and suspended as long as the first two arguments are not both ground. The user-defined predicate `place` is also suspended if its argument is not ground.

The single integrity constraint can be "divided" into two equivalent rules which make the concept of "attacking" explicit. They will be given the names `mark` and `dead` to stress their intended procedural interpretation.

```
mark:  queen( Col, Row ) ∧ move( Col, Row, C, R ) → attacked( C, R )
dead:  queen( Col, Row ) ∧ attacked( Col, Row ) → false
```

where the predicate `attacked` is external and therefore always suspended. The integrity constraint `mark` states that if a `queen` is placed at coordinates `(Col,Row)` and if it can move from there to coordinates `(C,R)`, then those coordinates are `attacked`. The integrity constraint `dead` states that a `queen` must not be placed at `attacked` coordinates. Declaratively, the conjunction of the two rules is equivalent to the single rule, as far as the two predicates `queen` and `move` are concerned. Procedurally, however, the generation of the `attacked` literals by the `mark` rule leads to a different operational behaviour, thus highlighting the flexibility of the framework for programming purposes. Whether the one- or the two-rule version is more efficient depends on the implementation, but the two-rule version makes it easier to illustrate certain features of the framework, so it will be used in the remainder of the paper.

3 Declarative semantics

Let $\mathcal{T}$ be the set of iff definitions for all user-defined, external and built-in predicates. Let $\mathcal{IC}$ be the set of all integrity constraints. Let all atomic formulae be classified as suspended or unsuspended in such a way that all atomic formulae having external predicates are suspended. Let G_0 be a conjunction of atomic formulae. Then A is said to be an *answer* to G_0 if and only if

(1) A is a conjunction of suspended literals

(2) $\mathcal{T} \models_m [G_0 \leftrightarrow A \vee Rest]$ for some formula $Rest$

(3) $\mathcal{T} \not\models_m [A \leftrightarrow false]$

where $\mathcal{T} \models_m \mathcal{S}$ means that the formula S is a property of $\mathcal{T}$ (as defined in section 2). Similarly to the definitions in $\mathcal{T}$, the equivalence $G_0 \leftrightarrow A \vee Rest$ is shorthand for
$\forall X_0, \ldots, X_n[G_0 \leftrightarrow A \vee Rest]$
where $X_0, \ldots, X_n$ are all the variables in G_0.[2] Then (2) means that G_0 and $A \vee Rest$ are true for the same assignments of the shared variables in the minimal Herbrand model of $\mathcal{T}$, namely G_0 and $A \vee Rest$ are *equivalent* in $\mathcal{T}$.

It is not always possible to verify condition (3), because the definitions in $\mathcal{T}$ might not be accessible, as in the case of external predicates, or because it would be impractical to access them, as in the case of built-in predicates. For example, consider the goal G
`X < 0` $\wedge$ `0 < X`
where "`<`" is built-in, defined over the integers and suspended if it contains non-ground arguments. Then G satisfies all the conditions for being an answer except condition (3). However, to detect the failure of (3), the proof procedure would have to unfold the two suspended literals in the goal, which is both impractical (because it does not terminate) and undesired (because the literals are suspended).

Thus one might want to replace condition (3) by
(3′) $\mathcal{IC} \not\models [A \leftrightarrow false]$
This condition is weaker than (3) because $\mathcal{IC}$ is only an approximation of $\mathcal{T}$. However, the more closely $\mathcal{IC}$ approximates $\mathcal{T}$, the better (3′) approximates (3). Now suppose that $\mathcal{IC}$ contains the integrity constraint
`X < Y` $\wedge$ `Y < X` $\rightarrow$ `false`
Then $\mathcal{IC} \models [G \leftrightarrow false]$, showing that (3′) is not satisfied, which implies that (3) is not satisfied, because $\mathcal{T} \models_m \mathcal{IC}$ and thus $\mathcal{IC} \models S$ implies $\mathcal{T} \models_m S$ (for any S).

Unfortunately, condition (3′) is too weak in many cases. For example, suppose that G is
`X < 0` $\wedge$ `1 < X`
and $\mathcal{IC}$ consists of the transitivity rule
`X < Y` $\wedge$ `Y < Z` $\rightarrow$ `X < Z`
Again condition (3) fails to hold, but this cannot be detected. However, the weaker condition (3′) is satisfied, which can be tested by resolving $\mathcal{IC}$ with G, generating the equivalent goal:
`X < 0` $\wedge$ `1 < X` $\wedge$ `1 < 0`
Since $\mathcal{IC} \not\models [1 < 0 \leftrightarrow false]$, G would be an answer if (3′) were used instead of (3). This is

[2] Note that $\forall X_0, \ldots, X_n[G_0 \leftrightarrow A \vee Rest]$ is a property of $\mathcal{T}$ if and only if $\mathcal{T} \models_m [G_0 \leftarrow A]$. With condition (2) formulated this way, the definition of answer for the special case of ALP closely resembles the notion of abductive explanation.

undesirable, and the following condition, which is a better approximation to (3) than is (3′), can be used instead:

(3″) $\mathcal{IC}(\mathcal{T}) \not\models [A \leftrightarrow false]$

where $\mathcal{IC}(\mathcal{T})$ is obtained by considering all instantiations (including non-ground ones) of the integrity constraints in $\mathcal{IC}$ and unfolding all literals (whether in the head or the body of an instantiated rule) as far as possible, i.e. as long as they are not suspended, using the iff definitions in $\mathcal{T}$. Unsuspended literals that cannot be unfolded via $\mathcal{T}$ are replaced by `false`. In the example, this means that $\mathcal{IC}(\mathcal{T})$ contains the rule

`1 < Y` $\wedge$ `Y < 0` $\rightarrow$ `false`

obtained by unfolding `1 < 0` to `false` in the head of the instance of the transitivity rule in $\mathcal{IC}$ with `X=1` and `Z=0`, using the theory $\mathcal{T}$. Thus G does not satisfy (3″). $\mathcal{IC}(\mathcal{T})$ should be thought of as the "best approximation" to $\mathcal{T}$ which the integrity constraints $\mathcal{IC}$ can provide by taking $\mathcal{T}$ into account, but without actually adding the whole of $\mathcal{T}$ to them.

The definition of $\mathcal{IC}(\mathcal{T})$ can also be motivated by looking at the special instance of the TPCLP framework which corresponds to ALP. The condition that all atoms in formulae in $\mathcal{IC}(\mathcal{T})$ are suspended corresponds to the condition in ALP that all atoms in integrity constraints are abducibles. The construction of $\mathcal{IC}(\mathcal{T})$ from $\mathcal{IC}$ corresponds in ALP to the transformation of arbitrary integrity constraints, by unfolding, into integrity constraints that contain only abducible atoms. A similar transformation also applies in the case of Semantic Query Optimization, where integrity constraints are unfolded so that they contain extensional atoms.

4-queens example (continued)

One of the two possible answers for the initial goal (see section 2) is

A = `queen(1,2)` $\wedge$ `queen(2,4)` $\wedge$ `queen(3,1)` $\wedge$ `queen(4,3)`

Since `queen` is an external predicate, A consists of suspended literals only and thus satisfies condition (1) of the answer definition. Moreover, the initial goal is equivalent in $\mathcal{T}$ to

(`queen(1,1)` $\vee \ldots \vee$ `queen(1,4)`) $\wedge$ (`queen(2,1)` $\vee \ldots \vee$ `queen(2,4)`) $\wedge$
(`queen(3,1)` $\vee \ldots \vee$ `queen(3,4)`) $\wedge$ (`queen(4,1)` $\vee \ldots \vee$ `queen(4,4)`)

Therefore A is one of the $4^4 = 256$ disjuncts in the formula resulting from the application of distributivity. So A satisfies condition (2) as well. Finally, A complies with (3″) because no two queens in A attack each other and `false` cannot be derived from $\mathcal{IC}(\mathcal{T}) \cup \{A\}$.

Note that, to reject a potential answer which contains both `queen(1,1)` and `queen(1,2)`, the following implications in $\mathcal{IC}(\mathcal{T})$ are used

`queen(1,1)` $\wedge$ `true` $\rightarrow$ `attacked(1,2)`
`queen(1,2)` $\wedge$ `attacked(1,2)` $\rightarrow$ `false`

The atom `true` in the first implication comes from unfolding `move(1,1,1,2)`.

4 Proof procedure

Let G_0 be an initial goal. A *derivation* 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 $0 < i < n$, G_{i+1} is obtained from G_i by applying one of the following operations (defined formally later):

- *unfolding* an unsuspended atom, using an iff definition

- *propagation*, which is a form of resolution applied to conjuncts (regarded as clauses) in disjuncts in G_i
- *logical equivalence*, used as rewrite rules to replace a formula which matches the left-hand side by the formula on the right-hand side. These equivalences include:
  - $[(A \vee B) \wedge C] \leftrightarrow [(A \wedge C) \vee (B \wedge C)]$ (*splitting*)
  - $A \wedge true \leftrightarrow A$
  - $A \vee true \leftrightarrow true$
  - $A \wedge false \leftrightarrow false$
  - $A \vee false \leftrightarrow A$

Each of these operations, including the step from G_0 to G_1, derives an equivalent formula in $\mathcal{T}$, i.e. for $0 \leq i \leq n-1$
$\mathcal{T} \models_m [G_i \leftrightarrow G_{i+1}]$.
Unfolding together with splitting reduces the initial goal to the form $A \vee Rest$, which is needed to generate answers. Propagation tests that a potential answer A satisfies condition (3$''$) of the declarative semantics. In general, testing for satisfiability is not even semi-decidable. However, the fact that $\mathcal{IC}(\mathcal{T})$ may contain infinitely many sentences does not cause a problem in practice for two reasons. First, by compactness, if $\{A\} \cup \mathcal{IC}(\mathcal{T})$ is unsatisfiable then $\{A\} \cup \mathcal{IC}'$, for some finite subset $\mathcal{IC}'$ of $\mathcal{IC}(\mathcal{T})$, is unsatisfiable. Second, if $\mathcal{IC}(\mathcal{T})$ itself is satisfiable, then any inconsistency must involve A, and the test for satisfiability can use a set-of-support resolution strategy whereby one of the parents of every resolvent is in A or derived from A. This means that instead of first generating $\mathcal{IC}(\mathcal{T})$ and then testing for satisfiability, it is possible to resolve A with $\mathcal{IC}$ and to use unfolding to eliminate unsuspended atoms from the resolvent.

Unfolding is defined as follows. If $(H \leftrightarrow D_1 \vee \ldots \vee D_m) \in \mathcal{T}$ and there is a substitution σ such that $H\sigma$ is a subformula[3] of G_i and $H\sigma$ is not suspended, then replace $H\sigma$ in G_i by $(D_1 \vee \ldots \vee D_m)\sigma$ to obtain G_{i+1}. Furthermore, if an atomic subformula of G_i is not suspended, but cannot be unfolded using an iff definition in $\mathcal{T}$, then replace it by `false` to obtain G_{i+1}.

To define propagation precisely, the quantifier status of variables in non-initial goals $G_i, i > 0$, has to be clarified. In the following, quantifiers are represented explicitly, distinguishing between free variables, which are all those variables which appear in the initial goal, existentially quantified variables, which are introduced by unfolding, and universally quantified variables, which are associated with integrity constraints and with resolvents derived from integrity constraints.

Propagation is easiest to define for a goal G_i in the canonical form of a disjunction $D_1 \vee \ldots \vee D_n$, possibly containing free variables $X_1, \ldots, X_r$ where each disjunct D_j is an existentially quantified conjunction
$\exists Y_1, \ldots, Y_{r'}[C_{j1} \wedge \ldots \wedge C_{jm}]$
and each conjunct C_{jk} is either an atomic formula or a universally quantified implication of the form
$\forall Z_1, \ldots, Z_{r''}[A_1 \wedge \ldots \wedge A_l \rightarrow B]$
Any goal G_i in a derivation can be put into this form by appropriate application of logical

[3] Including atoms in both the conditions and the heads of implications, i.e. of integrity constraints and resolvents.

equivalences. Then *propagation* is defined as any complete resolution strategy applied to atoms and implications in one conjunction $D_j = C_{j1} \wedge \ldots \wedge C_{jm}$ of G_i, such that at least one of the clauses resolved upon is not an integrity constraint in $\mathcal{IC}$, i.e. has been derived from the initial goal G_0. If R is the resulting resolvent and R is different from each conjunct C_{jk}, then G_{i+1} is obtained by replacing D_j by

$D'_j = C_{j1} \wedge \ldots \wedge C_{jm} \wedge R$

Depending on the resolution strategy, subsumption can be used to simplify D'_j. For the purpose of unification, free variables and existentially quantified variables are treated as (skolem) constants.

If the initial goal G_0 has been transformed to a goal $G = (A \wedge Imp) \vee Rest$, then A is a *computed solution* to G_0 if and only if

- $A \neq false$
- A consists of atoms only; Imp consists of (non-trivial) implications only
- the operations unfolding, propagation and logical equivalence have been exhaustively applied to $A \wedge Imp$

Imp consists of the integrity constraints in $\mathcal{IC}$ and resolvents obtained by resolving with integrity constraints in $\mathcal{IC}$ and by unfolding any unsuspended atoms in such resolvents. If G_0 has been transformed to $false$, then the initial goal has no solution.

It is easy to see that every computed solution satisfies conditions (1) and (2) of the definition of answer in the declarative semantics. Moreover, from the definition of propagation it follows that condition (3″) and thus also condition (3) are violated if G is transformed to `false`, i.e. the proof procedure is "refutation sound". However, it is not possible to guarantee that every computed solution satisfies (3), cf. the discussion in the previous section. It appears (but has not yet been formally shown) that every computed solution satisfies condition (3″) of the declarative semantics ("affirmation soundness"). Completeness of the proof procedure is under investigation (see [WeKoTo95] for more details).

5 Improving efficiency

Propagation has to be restricted to make it more efficient and especially to avoid non-termination, at least in cases where termination is possible as in the following example. Suppose the initial goal G_0 is

$\mathtt{X < 2 \wedge 0 < X}$

and $\mathcal{IC}$ consists only of the transitivity rule for "<". G_0 is an answer but unrestricted resolution generates infinitely many resolvents

$\forall$ Z: 2 < Z $\rightarrow$ X < Z

$\forall$ Y,Z: 2 < Y $\wedge$ Y < Z $\rightarrow$ X < Z

$\forall$ Y′,Y,Z: 2 < Y′ $\wedge$ Y′ < Y $\wedge$ Y < Z $\rightarrow$ X < Z

...

Thus unrestricted resolution does not terminate even in this trivial example. A very useful restriction, which solves this problem, is, in addition to the set-of-support strategy defined above, never to resolve upon two literals which both contain universally quantified variables. It can be shown that this restriction does not affect the detection of inconsistencies.

In practice it is unnecessary and undesirable to transform goals into disjunctive normal form in order to apply propagation. Instead, it is desirable to delay splitting as long as possible and to deal with goals in the form of disjunctions

$D_1 \vee \ldots \vee D_n$

with $n \geq 1$[4] and

$D_i = C_1 \wedge \ldots \wedge C_r$

with $r \geq 1$, where the C_j are atoms or implications or disjunctions (e.g. of conjunctions of atoms and implications). Within D_i any conjuncts which are atoms or implications are called *global constraints*. If one of the conjuncts C_k is $(A \wedge B) \vee C$, then A and B are called *local constraints* with respect to the first disjunct of C_k.

Propagation needs to be redefined as a form of non-clausal resolution, so that it applies to a goal in this more general form. The reformulated non-clausal resolution strategy is referred to as *Constraint Propagation with Disjunctions (CPD)*. The result of applying CPD to a goal in general form is logically equivalent, via disjunctive normalization, to the result of applying several steps of simplified propagation to the disjunctive normal form of the goal. Let C_1 and C_2 be clausal subformulae of a goal and let R be their resolvent. Then three forms of CPD can be distinguished:

- *local-to-local* CPD, where C_1 and C_2 are both local to the same disjunct and R is added locally to the disjunct;
- *global-to-global* CPD, where C_1 and C_2 are both global to the same disjunct and R is added globally to the disjunct;
- *global-to-local* CPD, where C_1 is local and C_2 is global to the same disjunct, and R is added locally to the disjunct.

A fourth kind of CPD, *local-to-global* CPD, can be defined either by allowing disjunctions in the conditions of integrity constraints or by factoring common information out of the disjuncts of a disjunction. For a more precise definition of the four types of CPD see [WeKoTo95], but the example below should sufficiently clarify the intended meaning.

Depending on the application, it may be advisable to restrict the use of some of the CPD techniques to improve efficiency. This can be done using the framework's control component, which can also be used to delay splitting as long as possible, to reduce the size of the search tree, and to choose a disjunction for splitting according to user-defined heuristics. CPD is a powerful method to delay and avoid splitting, as the example below will illustrate.

In addition to the operations of unfolding, propagation and logical equivalence, the further operation of *deletion* can be added to the proof procedure, to simplify goals and to improve efficiency. Conceptually, the deletion operation can be defined in the same way as propagation, just that the resolvent is not added but removed from the current goal. Obviously, propagation and deletion have to be operationally restricted so that they do not undo each other, potentially causing infinite loops. This can be done either by the user designating some integrity constraints exclusively for the purpose of deletion (and restricting deletion to resolution with these integrity constraints) or by making the proof procedure clever enough to decide for itself when an integrity constraint should be used for deletion and when for propagation. These issues are under investigation.

[4] I.e. the goal may consist of a single disjunct. This is the case for the initial goal.

Deletion is not needed in the n-queens problem, but is important in other applications, such as scheduling, when bounds are updated. For example, if `X` $\geq$ `1` is propagated and added to a goal which already contains `X` $\geq$ `0`, then the latter atom can safely be deleted, justified by the transitivity rule (if present in $\mathcal{IC}$). It seems that deletion can often, if not always, be thought of as subsumption — `X` $\geq$ `1` clearly subsumes `X = 0` $\vee$ `X` $\geq$ `1`, which is equivalent to `X` $\geq$ `0`, assuming the usual definition of "$\geq$" over the integers.

4-queens example (continued)

The reader may find it helpful to draw a chessboard when following the example. Integrity constraints and their resolvents will be omitted for simplicity.

First, unfolding is applied to the initial goal (see section 2) to obtain

$$(\texttt{queen(1,1)} \vee \ldots \vee \texttt{queen(1,4)}) \wedge \qquad \texttt{(D1)}$$
$$(\texttt{queen(2,1)} \vee \ldots \vee \texttt{queen(2,4)}) \wedge \qquad \texttt{(D2)}$$
$$(\texttt{queen(3,1)} \vee \ldots \vee \texttt{queen(3,4)}) \wedge \qquad \texttt{(D3)}$$
$$(\texttt{queen(4,1)} \vee \ldots \vee \texttt{queen(4,4)}) \qquad \texttt{(D4)}$$

Propagation using the `mark` integrity constraint can be applied locally within each of the 16 disjuncts appearing in the goal (local-to-local CPD) before any of the four disjunctions `D1`–`D4` is actually split. This leads to the generation of 16 hypothetical chessboards, represented by the given `queen` and the newly generated `attacked` literals. Four such chessboards are represented by the following conjunct `D1`$'$ of the goal resulting from exhaustively applying local-to-local CPD:

$$(\texttt{queen(1,1)} \wedge \texttt{attacked(1,2)} \wedge \ldots \wedge \texttt{attacked(4,4)}) \vee \ldots \vee$$
$$(\texttt{queen(1,4)} \wedge \texttt{attacked(1,3)} \wedge \ldots \wedge \texttt{attacked(4,4)})$$

Then the first disjunct of `D1`$'$ can be split off and the conjunction

$$\texttt{queen(1,1)} \wedge \texttt{attacked(1,2)} \wedge \ldots \wedge \texttt{attacked(4,4)} \wedge \texttt{D2}' \wedge \texttt{D3}' \wedge \texttt{D4}' \qquad \texttt{(*)}$$

considered next, where `D2`$'$–`D4`$'$ are obtained similarly to `D1`$'$ by exhaustively applying local-to-local CPD to `D2`–`D4`. The use of the `dead` integrity constraint together with the `attacked` literals, which are global in `(*)`, and the `queen` literals local to the disjuncts of `D2`$'$–`D4`$'$ will propagate `false` into the disjuncts of `D2`$'$–`D4`$'$ whose queens are "attacked" (global-to-local CPD). Then the disjuncts with `false` are eliminated by logical equivalence and thus `D2`$'$, `D3`$'$ and `D4`$'$ are respectively reduced to

$$(\texttt{queen(2,3)} \wedge \ldots) \vee (\texttt{queen(2,4)} \wedge \ldots) \qquad \texttt{(D2}''\texttt{)}$$
$$(\texttt{queen(3,2)} \wedge \ldots) \vee (\texttt{queen(3,4)} \wedge \ldots) \qquad \texttt{(D3}''\texttt{)}$$
$$(\texttt{queen(4,2)} \wedge \ldots) \vee (\texttt{queen(4,3)} \wedge \ldots) \qquad \texttt{(D4}''\texttt{)}$$

where the dots denote the `attacked` literals generated in the initial local-to-local propagation phase. In `D3`$''$, one of these literals, `attacked(2,3)`, has been generated in both of the two disjuncts. This common local constraint can thus be factored out of the disjunction `D3`$''$ and made global (local-to-global CPD). It can then be used together with the `dead` integrity constraint to propagate `false` into the disjunct with `queen(2,3)` in `D2`$''$ (global-to-local CPD). Therefore, after applying logical equivalence, only `queen(2,4)` is left in `D2`$''$. At this point `(*)` has been transformed to:

$$\texttt{queen(1,1)} \wedge \texttt{queen(2,4)} \wedge \texttt{D3}'' \wedge \texttt{D4}'' \wedge \ldots \qquad \texttt{(**)}$$

where the dots again refer to `attacked` literals. `queen(2,4)` has now become global and can be used with the `dead` integrity constraint to propagate `false` into the disjuncts of `D3`$''$ and `D4`$''$ containing `queen(3,4)` and `queen(4,2)` (global-to-local CPD). Thus `(**)` has been reduced to

$$\texttt{queen(1,1)} \wedge \texttt{queen(2,4)} \wedge \texttt{queen(3,2)} \wedge \texttt{queen(4,3)} \wedge \ldots \qquad (***)$$

But `queen(3,2)` and `queen(4,3)` attack each other which is detected by the `dead` integrity constraint. Thus `false` is added to `(***)` (by global-to-global CPD), and it can be concluded that there is no solution with `queen(1,1)`.

The other three disjuncts of $\texttt{D1}'$ will also be tried, and solutions found for `queen(1,2)` and `queen(1,3)`. No further splitting is necessary to identify the two solutions if all four types of CPD are fully applied.

It should be obvious that if the initially proposed single integrity constraint

$$\texttt{queen(Col, Row)} \wedge \texttt{move(Col, Row, C, R)} \wedge \texttt{queen(C, R)} \rightarrow \texttt{false}$$

had been used instead of the rules `mark` and `dead`, the behaviour of the proof procedure would have been very different. In particular, local-to-global CPD would not have been applicable, increasing the number of necessary splitting operations for the case $n = 6$ from 120 to 132 (compared to 1,917 if CPD was not applied at all and only basic propagation took place). Of course, this does not imply that the two-rule version is computationally more efficient than the single-rule version. It does, however, demonstrate the framework's flexibility in that two obviously declaratively equivalent approaches yield different operational behaviours.

6 Conclusion and future work

Due to the explicit representation of disjunctions, the TPCLP framework lends itself to problems involving search, and the four types of CPD constitute powerful methods to prune the search tree significantly. Several other frameworks have been proposed in the literature for handling disjunctions (disjunctive constraints). The relationship between these frameworks and the TPCLP framework is analysed in [WeKoTo95].

A prototype implementation of the framework exists, but more details of the control component have yet to be worked out before an efficient implementation can be obtained. Other applications, e.g. planning and configuration problems, are under investigation, as are the correctness and completeness of the proof procedure.

References

[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

[CoThTo91] Console, L.; Theseider Dupré, D.; Torasso, P.: On the Relationship between Abduction and Deduction, Journal of Logic and Computation 2(5), Oxford University Press 1991

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

[Fu95] Fung, T. H.: An Abductive Proof Procedure Based on the Clark Completion, Imperial College 1995 (draft)

[KoToWe94] Kowalski, R. A.; Toni, F.; Wetzel, G.: Towards a declarative and efficient glass-box CLP language, 10th WLP, Universität Zürich 1994

[WeKoTo95] Wetzel, G.; Kowalski, R.; Toni, F.: The TPCLP framework and Constraint Propagation with Disjunctions, Imperial College 1995 (draft)