Artificial Intelligence 58 (1992) 3–20
Elsevier

ARTINT 948

# The logic of constraint satisfaction

Alan K. Mackworth*

*Department of Computer Science, University of British Columbia, Vancouver, BC, Canada V6T 1W5*

*Abstract*

Mackworth, A.K., The logic of constraint satisfaction, Artificial Intelligence 58 (1992) 3–20.

The constraint satisfaction problem (CSP) formalization has been a productive tool within Artificial Intelligence and related areas. The finite CSP (FCSP) framework is presented here as a restricted logical calculus within a space of logical representation and reasoning systems. FCSP is formulated in a variety of logical settings: theorem proving in first order predicate calculus, propositional theorem proving (and hence SAT), the Prolog and Datalog approaches, constraint network algorithms, a logical interpreter for networks of constraints, the constraint logic programming (CLP) paradigm and propositional model finding (and hence SAT, again). Several standard, and some not-so-standard, logical methods can therefore be used to solve these problems. By doing this we obtain a specification of the semantics of the common approaches. This synthetic treatment also allows algorithms and results from these disparate areas to be imported, and specialized, to FCSP; the special properties of FCSP are exploited to achieve, for example, completeness and to improve efficiency. It also allows export to the related areas. By casting CSP both as a generalization of FCSP and as a specialization of CLP it is observed that some, but not all, FCSP techniques lift to CSP and thereby to CLP. Various new connections are uncovered, in particular between the proof-finding approaches and the alternative model-finding approaches that have arisen in depiction and diagnosis applications.

## 1. Logical frameworks for constraint satisfaction

Informally, a constraint satisfaction problem (CSP) is posed as follows. Given a set of variables and a set of constraints, each specifying a relation on a particular subset of the variables, find the relation on the set of all the variables which satisfies all the given constraints. Typically, the given unary relation for each variable specifies its domain as a set of possible values; the required solution relation is a subset of the Cartesian product of the variable domains. If

*Correspondence to*: A.K. Mackworth, Department of Computer Science, University of British Columbia, Vancouver, BC, V6T 1W5

* Shell Canada Fellow, Canadian Institute for Advanced Research.

each domain is finite, the CSP is a finite constraint satisfaction problem (FCSP).

The formulation of the CSP paradigm has yielded substantial theoretical and practical results [6, 17]. It is important, though, not to conceive of the CSP paradigm in isolation but to see it in its proper context, namely, as a highly restricted logical calculus with associated properties and algorithms. The purpose of this paper is to place CSPs in that context, to redevise some old results in new, simpler ways, and to establish connections amongst the differing logical views of CSPs. Essentially the paper can be seen as an extended answer to the question, "Does the CSP framework make logical sense?". It can also be seen as a response to a cynical critic who asks, "Is CSP merely old wine in new bottles?". It is intended to lead to answers to the following questions:

- Can FCSP be posed and solved in logical frameworks?
- Can the special properties of FCSP be exploited to get better algorithms?
- Are tractable classes of FCSP revealed?
- What are the relationships among the several logical views of FCSP?
- Can the approaches for FCSP be lifted to CSP?
- Do new results and systems follow from this approach?

## 2. An FCSP: the Canadian flag problem

To fix the ideas of this paper, a trivial FCSP will be used as an example. Consider the well-known Canadian flag problem. A committee proposed a new design for the Canadian flag, shown in Fig. 1. The problem is to decide how to colour the flag. Only two colours, red and white, should be used; each region should be a different colour from its neighbours, and the maple leaf should, of course, be red. The problem is so trivial that its solution requires little thought, but it serves our purpose here.

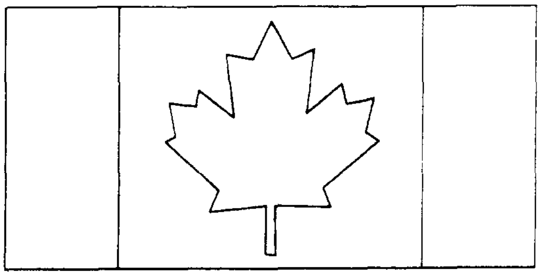

Fig. 1. A trivial FCSP: colour the Canadian flag.

## 3. FCS as theorem proving in FOPC

Solving an FCSP can be formulated as theorem proving in a restricted form of first-order predicate calculus, as follows [1, 16]. The FCSP decision problem is equivalent to determining if *Constraints*$\vdash$*Query* where *Query* has the form

*Query*:

$$\exists X_1 \exists X_2 \cdots \exists X_n \ \mathit{QMatrix}(X_1, X_2, \ldots, X_n)$$

or

*Query*:

$$\begin{aligned}\exists X_1 \exists X_2 \cdots \exists X_n \ & p_{X_1}(X_1) \wedge p_{X_2}(X_2) \wedge \cdots \wedge p_{X_n}(X_n) \wedge \\ & p_{X_1X_2}(X_1, X_2) \wedge p_{X_1X_3}(X_1, X_3) \wedge \cdots \wedge \\ & p_{X_1X_2X_3}(X_1, X_2, X_3) \wedge \cdots \wedge \\ & \vdots \\ & p_{X_1X_2X_3\cdots X_n}(X_1, X_2, X_3, \ldots, X_n)\end{aligned}$$

and *Constraints* is a set of ground atoms specifying the extensions of the predicates

*Constraints*:

$$\{p_{X_{i_1}X_{i_2}\cdots X_{i_m}}(c_{i_1}, c_{i_2}, \ldots, c_{i_m}) \mid 1 \leqslant i_k < i_{k+1} \leqslant n\}$$

where the $c_i$ are constants. Notice that in this formulation we are only specifying the tuples allowed by a relation, not the tuples forbidden, since *Constraints* consists of positive literals.

## 4. FCS decision problems

An FCSP is specified by a (*Constraints*, *Query*) pair. A common candidate formulation of the FCS decision problem is to determine if it can be shown that a solution exists or does not exist: *Constraints*$\vdash$*Query*, or *Constraints*$\vdash\neg$*Query*. However, given the positive form specified for *Constraints*, it is never possible to establish that *Constraints*$\vdash\neg$*Query*, so this candidate formulation is unacceptable. Later, when we consider the completion of *Constraints*, we shall return to a variant of this formulation.

The FCS decision problem (FCSDP) is to determine if it can be shown that a solution exists or if it cannot be shown that a solution exists: *Constraints*$\vdash$*Query* or *Constraints*$\nvdash$*Query*.

If the decision problem is posed in the form of FCSDP and the constraints are supplied or discovered incrementally in the form of additional allowed tuples, extending the set *Constraints*, then the answers to FCSDP are monotonic: a "No" may change to "Yes" but not vice versa.

**Proposition.** *FCSDP is decidable.*

**Proof.** For an FCSP specified by the pair (*Constraints*, *Query*), a decision algorithm to determine if $\mathit{Constraints} \vdash \mathit{Query}$ or $\mathit{Constraints} \nvdash \mathit{Query}$ is required. The Herbrand universe $H$ of the theory $\mathit{Constraints} \cup \neg \mathit{Query}$ is

$$H = \{c \mid p_{\Upsilon}(\ldots, c, \ldots) \in \mathit{Constraints}\}\ .$$

$H$ is finite.

Consider the following algorithm:

```
Decision Algorithm DA:

$Success \leftarrow$ No
For each tuple $(c_1, c_2, \ldots, c_n) \in H^n$
  If $Constraints \vdash QMatrix(c_1, c_2, \ldots, c_n)$ then
    $Success \leftarrow$ Yes
Report $Success$
End DA
```

where $\mathit{Constraints} \vdash \mathit{QMatrix}(c_1, c_2, \ldots, c_n)$ iff for each *Atom* mentioned in $\mathit{QMatrix}(c_1, c_2, \ldots, c_n)$ it is the case that $\mathit{Atom} \in \mathit{Constraints}$.

DA always terminates. It reports "Yes" iff $\mathit{Constraints} \vdash \mathit{Query}$. It reports "No" iff $\mathit{Constraints} \nvdash \mathit{Query}$. $\square$

The number of predicate evaluations made by DA is

$$(\#\ \mathit{atoms\ in}\ \mathit{QMatrix}(X_1, X_2, \ldots, X_n)) \times |H|^n\ .$$

## 5. Completing the constraints

Consider the completion of *Constraints* with respect to *Query*. Each predicate mentioned in *Query* can be completed, using Reiter's Closed World Assumption [23], in the following sense:

$$\mathit{completion}(\mathit{Constraints}) =$$

$$\mathit{Constraints} \cup$$

$$\{\neg p_{\Upsilon}(c_1, c_2, \ldots, c_k) \mid c_i \in H,\ p_{\Upsilon}(c_1, c_2, \ldots, c_k) \notin \mathit{Constraints}\}\ .$$

In other words, the complete extension of each $k$-ary predicate over $H^k$ is specified, positively and negatively, in *completion*(*Constraints*).

Notice that $Constraints \vdash Query$ iff $completion(Constraints) \vdash Query$ and $Constraints \nvdash Query$ iff $completion(Constraints) \vdash \neg Query$. Hence, DA reports "Yes" iff $completion(Constraints) \vdash Query$ and "No" iff $completion(Constraints) \vdash \neg Query$.

We may choose to interpret the answer from DA in the original sense of FCSDP or under the assumption that *Constraints* has been completed. Both interpretations are correct. "Yes" means $Constraints \vdash Query$ and $completion(Constraints) \vdash Query$; on the other hand, "No" means $Constraints \nvdash Query$ and $completion(Constraints) \vdash \neg Query$.

## 6. The flag FCSP in FOPC

Using the FCSP formalism presented above we can formulate the flag problem as follows (see Fig. 2):

*Query*:

$$\exists X \exists Y \exists Z \exists U\ p(X) \wedge q(Y) \wedge s(Z) \wedge t(U) \wedge ne(X, Y) \wedge ne(Y, Z) \wedge ne(Y, U) .$$

*Constraints*:

$$\{p(r), p(w), q(r), q(w), s(r), s(w), t(r), ne(r, w), ne(w, r)\} .$$

$H = \{r, w\}$ , where $r$ stands for red and $w$ for white .

$$H^4 = \{(r, r, r, r), (r, r, r, w), \ldots , (w, w, w, w)\} .$$

On the FCSP (*Query*, *Constraints*), algorithm DA returns "Yes" succeeding on the tuple $(r, w, r, r)$.

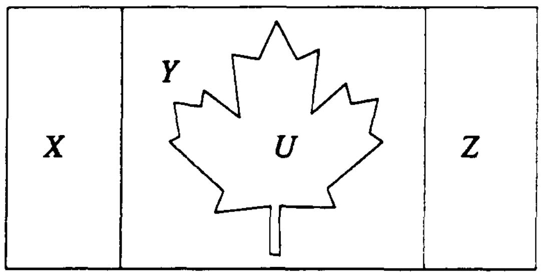

Fig. 2. The Canadian flag problem as an FCSP.

## 7. Logical representation and reasoning systems

Faced with a problem in representation and reasoning, a wide spectrum of logical representation systems is available to us. In choosing an appropriate system, we can rely on two sets of criteria: descriptive and procedural adequacy criteria [24]. These are often in conflict. The best advice is to use Occam's Razor: choose the simplest system with the level of descriptive adequacy required. Some representation and reasoning systems are shown, organized as a DAG, in Fig. 3. If there is a downward arc from system A to system B, then A's descriptive capabilities are a strict superset of B's. In the previous section, for example, FCS was shown to be equivalent to theorem-proving in a very restricted form of FOPC. Horn FOPC restricts FOPC in only allowing Horn clauses. (Recall that a clause, a disjunction of literals, is Horn if it has at most one positive literal. A Horn clause has one of four forms: a positive unit clause consisting of a single positive literal, a negative clause consisting only of negative literals, a mixed Horn clause consisting of one positive literal and at least one negative literal and, trivially, the empty clause, □. A definite clause has exactly one positive literal; it is either a unit positive clause or a mixed Horn clause.) Definite clause programs (DCP), without predicate completion, restrict Horn FOPC by allowing only one negative clause which serves as the query. Datalog essentially restricts DCP by eliminating function symbols. FCS restricts Datalog by disallowing rules, mixed Horn clauses. There are several further restrictions on FCS possible with correspond-

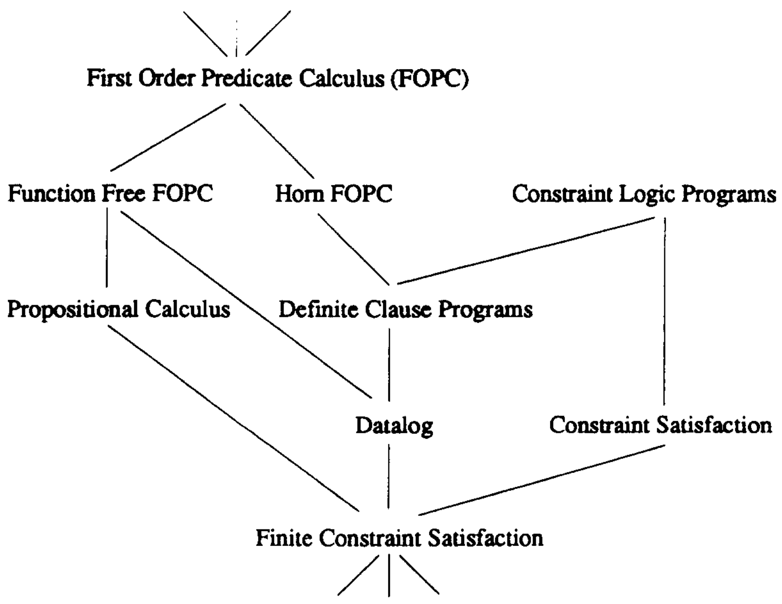

Fig. 3. Some logical representation and reasoning systems.

ing gains in tractability and some generalizations of FCS with gains in expressive power. We shall examine various logical formulations of FCS and investigate some of their interrelationships.

## 8. FCS as theorem proving in propositional calculus

The algorithm DA can be interpreted as implementing a view of FCS as theorem proving in the propositional calculus. *Query* is a theorem to be proved. If a solution exists, the theory *Constraints* $\cup \neg$ *Query* leads to a contradiction.

$$\neg\, Query:$$

$$\neg \exists X_1 \exists X_2 \cdots \exists X_n\, QMatrix(X_1, \ldots, X_n)$$

$$\forall X_1 \forall X_2 \cdots \forall X_n \neg QMatrix(X_1, \ldots, X_n)\,.$$

A solution exists iff *Constraints* $\cup \neg$ *Query* has no (Herbrand) models. There are no universal quantifiers in *Query* and so there are no existential quantifiers in the theory *Constraints* $\cup \neg$ *Query*. Hence no Skolem functions are introduced when the theory is converted to clausal form. This important restriction guarantees that the Herbrand universe $H$ is finite. This allows us to replace each of the universal quantifiers by the conjunction of the $\neg QMatrix(X_1, X_2, \ldots, X_n)$ clauses instantiated over $H^n$. This rewritten theory has the same set of Herbrand models as the original theory.

For the flag example the rewritten theory is:

$$\{p(r), p(w), q(r), q(w), s(r), s(w), t(r), ne(r, w), ne(w, r)\}$$

$$\cup$$

$$\begin{aligned}
\{&\neg p(r) \vee \neg q(r) \vee \neg s(r) \vee \neg t(r) \vee \neg ne(r, r)\,,\\
&\neg p(r) \vee \neg q(r) \vee \neg s(r) \vee \neg t(w) \vee \neg ne(r, r) \vee \neg ne(r, w)\,,\\
&\vdots\\
&\neg p(r) \vee \neg q(w) \vee \neg s(r) \vee \neg t(r) \vee \neg ne(r, w) \vee \neg ne(w, r)\,, \qquad (*)\\
&\vdots\\
&\neg p(w) \vee \neg q(w) \vee \neg s(w) \vee \neg t(w) \vee \neg ne(w, w)\}\,.
\end{aligned}$$

This theory, now a propositional formula in CNF, has a particular restricted form for any FCSP. It consists only of a set of unit positive clauses, arising from the constraints, and a set of negative clauses, from the query. There are no mixed clauses. Note that it is also always Horn. It is unsatisfiable iff the FCSP has a solution. Since it is Horn, SAT is linear time in the size of the formula [8], but, of course, there are $|H|^n$ negative clauses in the formula. Also note that unit resolution alone is complete for this class of formulas. For the flag example, repeated unit resolution on the clause marked $(*)$ reduces it to the empty clause $\Box$, corresponding to the solution $\{X = r, Y = w, Z = r, U = r\}$. For any FCSP, using subsumption (whereby if clauses of the generic form $C$ and $C \vee D$ are both present, $C \vee D$ is deleted) does not affect the completeness result. Iterating unit resolution followed by subsumption leaves invariant the special properties of the formula (only negative and unit positive clauses) and, moreover, only decreases its size. Hence, it terminates (correctly). As, of course, does the linear time HornSAT algorithm. The HornSAT algorithm exactly mimics the algorithm DA. The propositional variable in each unit positive literal is set to T and each negative clause is checked: if any clause has each (negative) literal required to be F then the formula is unsatisfiable otherwise it is satisfiable.

## 9. FCS as theorem proving in definite theories

The methods discussed so far are not serious candidates for actually solving an FCSP: they simply serve to clarify the semantics and methods of the serious candidates. One such candidate is a Prolog interpreter, which is a theorem prover for theories consisting of definite clauses. Since an FCSP can be seen as a pure Prolog program, SLD-resolution is a sound and complete solution method. A depth-first SLD-resolution strategy may fail to find a proof for a query that is in fact a theorem for an arbitrary definite clause program but will always do so for an FCSP. It is also, generally speaking, more efficient than other resolution methods such as the one embodied in algorithm DA. For the flag example, we can assert the constraints as ground facts in the Prolog database and then define and pose the conjunctive query to Prolog:

```
%prolog
| ?- [user].
| p(r). p(w). q(r). q(w). s(r). s(w). t(r).
| ne(r, w). ne(w, r).
yes
| ?- p(X), q(Y), s(Z), t(U), ne(X, Y), ne(Y, Z), ne(Y, U).
X = Z = U = r,
Y = w
```

In finding the one solution the interpreter essentially checks every possible set of bindings for the variables X, Y, Z and U. By permuting the query one may reduce the size of the search space: a partially completed set of bindings can be rejected by a single failure. For the query

```
| ?- p(x), q(Y), ne(X, Y), s(Z), ne(Y, Z), t(U), ne(Y, U)
```

the search tree is somewhat smaller. Heuristics, such as instantiating the most constrained variable next, can be used to reorder the query dynamically but, on realistic problems, this tactic is doomed. In general, no variable ordering can avoid *thrashing* by repeatedly rediscovering incompatible variable bindings [16].

Just as for the algorithm DA, we may interpret Prolog's failure to find a proof for this class of theories as meaning either $\mathit{Constraints} \nvdash \mathit{Query}$ or $\mathit{completion}(\mathit{Constraints}) \vdash \neg \mathit{Query}$.

## 10. FCS as Datalog

Since FCS is a restriction of Datalog, the techniques developed in the relational database community are available [19]. The solution relation is the natural join of the relations for the individual constraints. The consistency techniques discussed below can be interpreted similarly; for example, making an arc consistent in a constraint network can be interpreted as a semi-join. Results that exploit this interpretation can be found in [1, 6, 17, 21, 24, 28].

## 11. FCS in constraint networks

Consideration of the drawbacks of the SLD-resolution approach mentioned above leads to a view of FCS in *constraint networks*. A constraint network represents each variable in the query as a vertex. The unary constraint $p_X(X)$ establishes the domain of $X$, and each binary constraint $p_{XY}(X, Y)$ is represented as the edge $(X, Y)$, composed of arc $(X, Y)$ and arc $(Y, X)$. This easily generalizes to $k$-ary predicates using hypergraphs. The network for the flag problem is shown in Fig. 4.

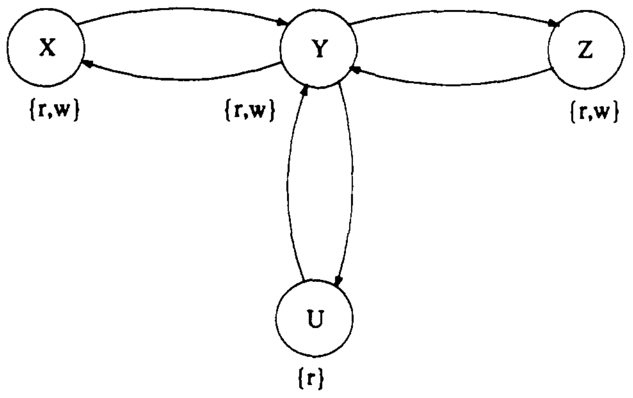

Fig. 4. Constraint network for the flag problem.

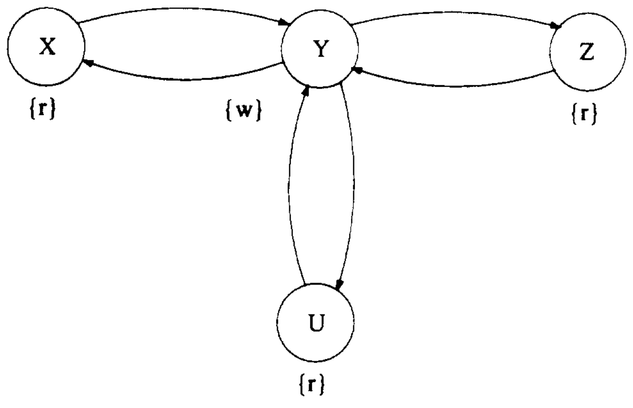

Fig. 5. Arc consistent network for the flag problem.

An arc $(X, Y)$ is *consistent* iff the following metalanguage sentence holds for *Constraints*:

$$\forall X\{p_X(X) \rightarrow \exists Y[p_Y(Y) \wedge p_{XY}(X, Y)]\} .$$

A network is arc consistent if all its arcs are consistent. An arc $(X, Y)$ may be made consistent without affecting the total set of solutions by deleting the values from the domain of $X$ that are not consistent with some value in $Y$. The original flag network is not arc consistent because the single arc $(Y, U)$ is inconsistent. Delete $r$ from the domain of $Y$ to make arc $(Y, U)$ consistent. This now makes arcs $(X, Y)$ and $(Z, Y)$ inconsistent. They can be made consistent by deleting $w$ from the domains of both $X$ and $Z$, making the network arc consistent as shown in Fig. 5.

Arc consistency can be enforced in time linear in the number of binary constraints; moreover, if the constraint graph is a tree, as it is for the flag, then arc consistency alone suffices as a decision procedure for the FCSP [18]. Various other graph-theoretic properties of the constraint network can be used to characterize and solve FCSPs [6, 11, 22].

## 12. Logical interpreters for FCSP

Using these ideas we can implement an interpreter for FCS [16, 27] which could be called LINC (logical interpreter for a network of constraints). Given

$$\textit{Query}: \quad \exists X \exists Y \exists Z \exists U \; p(X) \wedge q(Y) \wedge s(Z) \wedge t(U) \wedge \\ ne(X, Y) \wedge ne(Y, Z) \wedge ne(Y, U)$$

then, following [2], for LINC we choose to complete each predicate in

*Constraints* and represent it by its *definition*, its necessary and sufficient conditions, so

*Constraints*:

$$p(X) \leftrightarrow ((X = r) \vee (X = w)),$$
$$q(Y) \leftrightarrow ((Y = r) \vee (Y = w)),$$
$$s(Z) \leftrightarrow ((Z = r) \vee (Z = w)),$$
$$t(U) \leftrightarrow (U = r),$$
$$ne(X, Y) \leftrightarrow (X = r \wedge Y = w) \vee (X = w \wedge Y = r).$$

Restricting LINC to arc consistency, it non-deterministically rewrites *Constraints* using the *AC rewrite rule*:

$$p_X(X) \Leftarrow p_X(X) \wedge \exists Y[p_Y(Y) \wedge p_{XY}(X, Y)].$$

Here:

$$\begin{aligned} q(Y) &\Leftarrow q(Y) \wedge \exists U[t(U) \wedge ne(Y, U)] \\ &\Leftarrow (Y = r \vee Y = w) \wedge \exists U[U = r \wedge \\ &\qquad\qquad (Y = r \wedge U = w \vee Y = w \wedge U = r)] \\ &\Leftarrow (Y = w). \end{aligned}$$

The set of AC rewrite rules, corresponding to the set of arcs in the constraint network, applied to *Constraints* constitutes a production system with the Church–Rosser property. Repeated application of the rules, rewriting the definitions of the unary predicates, eventually reduces *Constraints* to a fixpoint:

$$p(X) \leftrightarrow (X = r),$$
$$q(Y) \leftrightarrow (Y = w),$$
$$s(Z) \leftrightarrow (Z = r),$$
$$t(U) \leftrightarrow (U = r),$$
$$ne(X, Y) \leftrightarrow (X = r \wedge Y = w) \vee (X = w \wedge Y = r).$$

In general, LINC must interleave the AC relaxation with some non-deterministic case analysis or higher-order network consistency.

CHIP [27] is, amongst other things, an implementation of this approach. Similarly, a connection graph theorem prover for full FOPC, as proposed in [15], essentially performs AC relaxation on the possible sets of substitutions for variables in the literals of each clause. Using such a prover with an SLD-resolution strategy on a FCSP query would produce an effect isomorphic to using LINC.

## 13. CSP and CLP($\mathcal{D}$)

The FCS constraint form is a special case of the CLP($\mathcal{D}$) rule form [12]:

$$p(X, Y, \dots) \leftarrow a_1(X, Y, \dots) \wedge a_2(X, Y, \dots) \wedge \cdots \wedge$$
$$p_1(X, Y, \dots) \wedge p_2(X, Y, \dots) \wedge \cdots$$

where $a_i(\cdot)$ is a constraint on its arguments and $p_j(\cdot)$ is a predicate.

In definite clause programs $\mathcal{D} = H$ and the constraints are equalities on terms.

A general constraint satisfaction problem fits the CLP($\mathcal{D}$) scheme. Consider the CSP represented by this CLP($\mathcal{R}$) program:

*Constraints*:

$$p(X) \leftarrow (1 \leq X) \wedge (X \leq 3)\,,$$
$$q(Y) \leftarrow (0 \leq Y) \wedge (Y \leq 2)\,,$$
$$r(X, Y) \leftarrow X \leq Y$$

and the

*Query*:

$$\exists X \exists Y [p(X) \wedge q(Y) \wedge r(X, Y)]\,.$$

Using the same AC rewrite rule that LINC used for finite CSP *Constraints* is refined to the fixpoint

*Constraints*:

$$p(X) \leftarrow (1 \leq X) \wedge (X \leq 2)\,,$$
$$q(Y) \leftarrow (1 \leq Y) \wedge (Y \leq 2)\,,$$
$$r(X, Y) \leftarrow X \leq Y\,.$$

This demonstrates that these ideas lift from FCSP to CSP and CLP. It is an open research issue to determine the limits of their applicability and their usefulness [3, 26].

## 14. FCS as model finding in propositional logic

A qualitatively different logical framework for FCS is as model finding in propositional logic [4, 14, 20, 24]. For example, in [24] an account of depiction is presented. An interpretation of an image is defined to be a logical model of a theory describing the image, the scene and the mapping between them. Under certain assumptions this theory reduces to a propositional theory whose models are identified with possible states of the world. Finding those models corresponds directly to solving an FCSP. In [4, 20] the model-finding framework for FCSP is used to elucidate the connection to truth maintenance systems.

In this framework a propositional formula $F$ is constructed for the FCSP such that each model of $F$ corresponds to a solution of the FCSP. Each proposition in $F$ represents a possible binding of a variable to a value. For the flag example, the proposition $X : r$ means that variable $X$ takes the value $r$. $F$ may be in CNF with a set of clauses representing the fact that each variable must take a value, e.g. $X : r \vee X : w$, the fact that the values are pairwise exclusive, e.g. $\neg X : r \vee \neg X : w$, and the constraints on related variables. The constraints may be encoded as clauses in any suitable fashion. A "negative" encoding [4, 20] represents only the forbidden tuples of the constraints, e.g. $\neg W : r \vee \neg Y : r$. Using that encoding for the flag example, we have

$$\begin{aligned} F = \{&X : r \vee X : w, Y : r \vee Y : w, Z : r \vee Z : w, U : r, \\ &\neg X : r \vee \neg X : w, \neg Y : r \vee \neg Y : w, \neg Z : r \vee \neg Z : w, \\ &\neg X : r \vee \neg Y : r, \neg X : w \vee \neg Y : w, \neg Y : r \vee \neg Z : r, \\ &\neg Y : w \vee \neg Z : w, \neg Y : r \vee \neg U : r\}. \end{aligned}$$

In this framework we have a SAT problem again. It is crucial to observe that this approach is *not* the model-theoretic counterpart of the proof-theoretic approach presented earlier. In the propositional proof-finding framework, the FCSP has a solution iff the formula has no models. Under this model-finding framework each solution corresponds to a model of $F$. To find the models we could use the Davis–Putnam algorithm. But we note that the SAT problem has the same special form again: there are no mixed clauses in this encoding. This can be exploited to simplify the formula before deciding if there are any models. Two inference rules can be used—a specialized form of negative hyperresolution $H_2$ and a form of unit resolution $U$.

$$H_2: \quad \frac{\begin{array}{l} p \vee q \vee r \vee \cdots \vee u \\ \neg p \vee \neg v \\ \neg q \vee \neg v \\ \neg r \vee \neg v \\ \vdots \\ \neg u \vee \neg v \end{array}}{\neg v}$$

$$U: \quad \frac{\begin{array}{l} p \vee q \vee r \vee \cdots \vee u \\ \neg q \end{array}}{p \vee r \vee \cdots \vee u}$$

These rules of inference are supplemented with two subsumption rules: $S_p$ and $S_n$. Given two positive clauses $C_1$ and $C_2$ where all the literals in $C_1$ appear in $C_2$ then $S_p$ deletes $C_2$, the subsumed clause. $S_n$ deletes subsumed negative clauses.

Now, we define the *AC-resolution strategy*: $(H_2S_n^*US_p)^*$.

A trace of AC-resolution on the flag example as it simplifies $F$ is shown in Fig. 6.

The resultant simplified formula is

$$F_s = \{X : r, Y : w, Z : r, U : r, \neg X : w, \neg Y : r, \neg Z : w\}$$

which obviously has exactly one model, corresponding to the solution.

Notice that resolution and subsumption are used here in a nonstandard way, namely, not to prove a theorem but to find models. It is easy to verify that the AC-resolution strategy has the following properties:

(1) The set of models is invariant under AC-resolution. The models of $F$ are the models of $F_s$. This follows from the soundness theorem [9]. $F \models F_s$

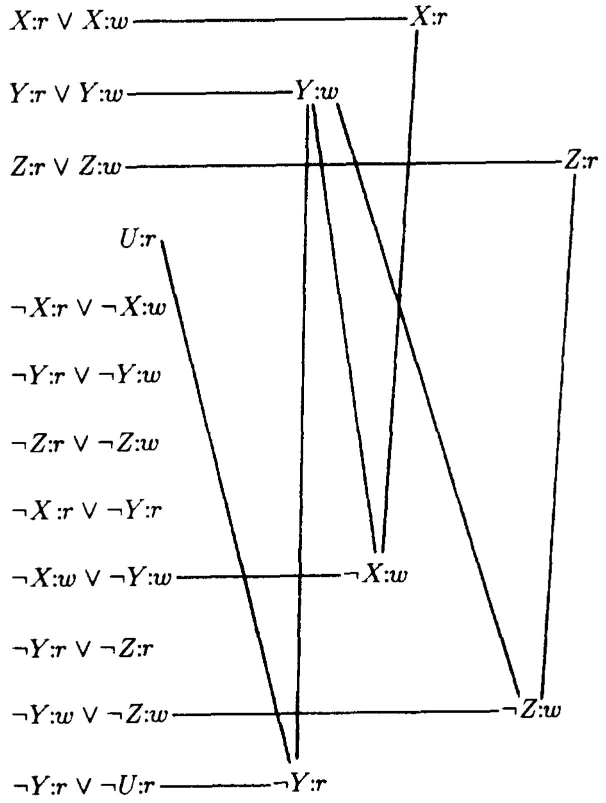

Fig. 6. A trace of AC-resolution on the flag problem.

(every model of $F$ is a model of $F_s$) because $F \vdash F_s$, and $F_s \models F$ because $F_s \vdash F$.

(2) No mixed clauses are generated so the separation into positive and negative clauses is invariant.

(3) The total number and length of clauses decreases monotonically.

(4) AC-resolution is $O(e)$ where $e$ is the number of constraints [18].

(5) In general, AC-resolution is incomplete in the sense that it does not always terminate with $F_s$ either as the empty clause $\square$ or consisting only of unit literals. It must be interleaved with search, such as assigning a truth value to a proposition, or enforcing higher-order network consistency, which corresponds to other forms of hyperresolution [4], to determine the models of $F$ explicitly.

(6) AC-resolution used for model-finding exactly mimics the behaviour of the LINC interpreter using the AC rewrite rule on FCSP (and the connection graph theorem prover) since each proposition, e.g. $X : r$, reifies a possible substitution for a variable in the FOPC theorem-proving framework.

The model-finding framework shows the relevance of a variety of serial and parallel complexity results on special cases of SAT, such as planar SAT [25], 2-SAT and HornSAT. If the variables in an FCSP have a maximum domain size of 2 and the constraints are only unary or binary (as is the case for the flag problem) then this encoding results in a 2-SAT problem which is $O(e)$ serial time [10] and polylogarithmic parallel time since it is in NC [13, 14].

A mixed encoding of the constraints may well be more appropriate than the negative encoding [24], especially for sparse constraints. Consider directed constraint networks [7] which are a specialization of FCSP. In a directed constraint network, for each constraint some subset of its variables can be considered as input variables to the constraint with the rest considered as output variables. The projection of the constraint relation on the input variables is the universal relation—that is, they are unconstrained. A constraint relation that is functional on the input variables has this form. A directed constraint that is not functional can be made functional by introducing additional input variables to discriminate between the different output values possible for the same values of the original input variables. The topology of the constraint network must respect this distinction between input and output variables.

A suitable mixed encoding of a directed constraint network in the model-finding framework can be arranged as a propositional theory, as follows. First, make each constraint functional as described above. A suitable theory can then be constructed with a positive clause for each variable, specifying that it must have a value in its domain, and a set of negative clauses for each variable, specifying that the values are pairwise exclusive, as before, but there are definite clauses for all the multivariable constraints. Each definite clause has negative literals for all the input variable/value bindings and a positive literal for the resulting output variable/value. If the values for the input variables to the network are known, the theory essentially collapses to become Horn as a modified HornSAT algorithm can determine the entire state of the network in O($e$) time [8]. On the other hand, determining the input values required to produce a given output can be much harder. Diagnosis of the internal state of a causal system can also be put in this framework: the states of the components correspond to introduced input variables [5].

## 15. Conclusions

In summary, the questions posed in Section 1 of this paper have all been answered affirmatively except, of course, the one posed by the cynical critic. The basic approach has been to see finite constraint satisfaction as a restricted logical calculus in a space of logical representation and reasoning systems. The FCSP framework has been formulated in a variety of logical settings: theorem proving in FOPC (which reduces to propositional theorem proving and hence SAT), the Prolog and Datalog approaches, constraint network algorithms, a logical interpreter for networks of constraints, the CLP paradigm and forms of propositional model-finding (and hence SAT, again, but in radically different context). Several standard, and some not-so-standard, logical methods can therefore be used to solve these problems. By doing this we obtain a specification of the semantics of the common approaches.

This synthetic treatment also allows algorithms and results from many of these disparate areas to be imported, and specialized, to FCSP; the special properties of FCSP are exploited to achieve, for example, completeness and to improve efficiency. It also allows export to the related areas. By casting CSP both as a generalization of FCSP and a specialization of CLP it was observed that some, but not all, FCSP techniques lift to CSP and, perhaps, thereby to CLP. Various new connections have been uncovered, in particular between the proof-finding approaches and the alternative model-finding approaches that have arisen in depiction and diagnosis applications.

## Acknowledgement

The contents of this paper benefitted from discussions with Johan de Kleer, Alex Kean, Nick Pippenger, David Poole, Ray Reiter, Ron Rensink and Zhang Ying, and from the comments of the referees. This research was supported by the Canadian Institute for Advanced Research, the Natural

Sciences and Engineering Research Council of Canada and the Institute for Robotics and Intelligent Systems Network of Centres of Excellence.

## References

[1] W. Bibel, Constraint satisfaction from a deductive viewpoint, *Artif. Intell.* **35** (1988) 401–413.

[2] K.L. Clark, Negation as failure, in: H. Gallaire and J. Minker, eds., *Logic and Databases* (Plenum Press, New York, 1978) 293-322.

[3] E. Davis, Constraint propagation with interval labels, *Artif. Intell.* **32** (1987) 281–331.

[4] J. de Kleer, A comparison of ATMS and CSP techniques, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 290-296.

[5] J. de Kleer, A.K. Mackworth and R. Reiter, Characterizing diagnoses, in: *Proceedings AAAI-90*, Boston, MA (1990) 324-330.

[6] R. Dechter, Constraint networks, in: S.C. Shapiro, ed., *The Encyclopedia of AI* (Wiley, New York, 1992) 276-285.

[7] R. Dechter and J. Pearl, Directed constraint networks, in: *Proceedings IJCAI-91*, Sydney, Australia (1991) 1164-1170.

[8] W.F. Dowling and J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, *J. Logic Program.* **3** (1984) 267–284.

[9] H.B. Enderton, *A Mathematical Introduction to Logic* (Academic Press, Orlando, FL, 1972).

[10] S. Even, A. Itai and A. Shamir, On the complexity of timetable and multicommodity flow problems, *SIAM J. Comput.* **5** (4) (1976) 691–703.

[11] E.C. Freuder, Complexity of *k*-tree structured constraint satisfaction problems, in: *Proceedings AAAI-90*, Boston, MA (1990) 4-9.

[12] J. Jaffar and J.-L. Lassez, Constraint logic programming, in: *Proceedings 14th ACM Principles of Programming Languages Conference*, Munich, Germany (1987) 111-119.

[13] N.D. Jones, Y.E. Lien and W.T. Laaser, New problems complete for nondeterministic log space, *Math. Syst. Theor.* **10** (1976) 1–17.

[14] S. Kasif, Parallel solutions to constraint satisfaction problems, in: *Proceedings First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ont. (1989) 180–188.

[15] R. Kowalski, A proof procedure using connection graphs, *J. ACM* **22** (4) (1975) 572–595.

[16] A.K. Mackworth, Consistency in networks of relations, *Artif. Intell.* **8** (1977) 99–118.

[17] A.K. Mackworth, Constraint satisfaction, in: S.C. Shapiro, ed., *The Encyclopedia of AI* (Wiley, New York, 1992) 285-293.

[18] A.K. Mackworth and E.C. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artif. Intell.* **25** (1985) 65–74.

[19] D. Maier, *The Theory of Relational Databases* (Computer Science Press, Rockville, MD, 1983).

[20] D.A. McAllester, Truth maintenance, in: *Proceedings AAAI-90*, Boston, MA (1990) 1109–1116.

[21] U. Montanari, Networks of constraints: fundamental properties and applications to picture processing, *Inf. Sci.* **7** (1974) 95–132.

[22] U. Montanari and F. Rossi, Constraint relaxation may be perfect, *Artif. Intell.* **48** (2) (1991) 143–170.

[23] R. Reiter, Nonmonotonic reasoning, in: H.E. Shrobe, ed., *Exploring Artificial Intelligence* (Morgan Kaufmann, San Mateo, CA, 1988) 439-482.

[24] R. Reiter and A.K. Mackworth, A logical framework for depiction and image interpretation, *Artif. Intell.* **41** (1989) 125–155.

[25] R. Seidel, A new method for solving constraint satisfaction problems, in: *Proceedings IJCAI-81*, Vancouver, BC (1981) 338-342.

[26] G. Sidebottom and W.S. Havens, Hierarchical arc consistency applied to numeric processing in constraint logic programming, Tech. Rept. CSS-IS TR 91-06, Simon Fraser University, Burnaby, BC (1991).

[27] P. Van Hentenryck, *Constraint Satisfaction in Logic Programming* (MIT Press, Cambridge, MA, 1989).

[28] Y. Zhang and A.K. Mackworth, Parallel and distributed algorithms for constraint satisfaction problems, in: *Proceedings 3rd IEEE Symposium on Parallel and Distributed Processing*, Dallas, TX (1991) 394–397.