2.3 Higher-Order Functional Constraint Networks

Enn Tyugu and Tarmo Uustalu

The Royal Institute of Technology, Dept. of Teleinformatics
Electrum 204, S-164 40 Kista, Sweden
tyugu | tarmo@it.kth.se

This paper discusses value propagation on functional constraint networks. We first focus on conventional "first-order" networks, and then show how the technique generalizes for what we call higher-order networks. Value propagation is attractive because of its simplicity and efficiency. Although it cannot solve all satisfiable networks (since it cannot break loops of constraints) and it may suggest solutions to unsatisfiable networks (since it does not discover inconsistency), it is of significant practical value. We describe several algorithms for planning value propagation, and point out that planning can be regarded as proof search in intuitionistic propositional logic.

2.3.1 Introduction

One of the oldest and still most common constraint solving techniques is value propagation along a constraint network. This technique is based on the direct usage of constraints as functional dependencies. The technique is very simple and efficient (Borning 1981). Efficiency is largely due to the technique's inherent "impatience" in deciding variable values, which sets limits to its applicability. As soon as, after having studied some chain of constraints in the given network, it is established that a variable X cannot take a value else than x, it is decided that X shall be x, without checking whether that is consistent with the other constraints. Thus, value propagation may in general produce wrong solutions. Nevertheless, if a network is known to be satisfiable in advance, the answers produced by the technique are surely correct, and such cases are not rare. Typical examples are constraint networks one uses for analyzing real-life situations, where constraints state the laws governing a situation and the values of known parameters. Such networks are obviously solvable, as the values are all "out there"; the problem is just how to get to know (derive) the values that cannot be measured directly. Also, clearly satisfiable are networks where any variable is the output of at most one constraint.

Value propagation is a way of gradually making constraint networks more explicit, and thus just a particular case of consistency propagation, such that propagation is interrupted as soon as non-trivial information is derived about each of the asked variables. Various consistency propagation algorithms for non-functional networks can be regarded as generalizations of value propagation, if one considers entities like sets or intervals as primitive values (e.g. tolerance propagation in interval arithmetic (Hyvönen 1992)). This is especially true in

the case when a constraint network schema can be transformed into a tree (Dechter, Pearl 1989; Dechter 1992; Montanari, Rossi 1991).

In the present paper, we first discuss value propagation on "first-order" functional constraint networks. Thereafter, we define a form of higher-order constraint networks and show how value propagation can be generalized to work on such networks. Algorithms are presented for planning value propagation on different classes of functional constraint networks.

Finding values for variables of higher-order functional constraints leads us to a concept of subproblem. Surprisingly enough, constraint satisfaction on certain classes of higher-order constraint networks has precise meaning in terms of propositional logics. These classes have been investigated in computer science in relation with automatic program construction by several authors (Mints, Tyugu 1983; Kanovich 1991).

2.3.2 Constraint Networks

2.3.2.1 Basic Definitions Throughout the paper, we will use underscored letters to denote either finite sets or finite lists (tuples). The set of all finite lists over a given set $\mathcal{M}$ will be denoted $\mathcal{M}^*$. The concatenation of two lists will be denoted by an infix comma, a list containg just one element will be identified with this element.

Definition 2.1 (Constraint network schema (CNS)) *A* constraint network schema (CNS) *is a triple* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C} \rangle$, *where* $\mathcal{X}$ *is a finite non-empty set of* variables, $\mathcal{R}$ *is a finite non-empty set of* constraints, $\mathcal{X} \cap \mathcal{R} = \emptyset$, *and* C *is a function from* $\mathcal{R}$ *into finite lists over* $\mathcal{X}$. *For any constraint* $R \in \mathcal{R}$, $C(R)$ *is called its* var-list.

Definition 2.2 (Constraint network (CN)) *A* constraint network (CN) *is a quintuple* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{I} \rangle$, *where* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C} \rangle$ *is a constraint network schema,* D *is a function from* $\mathcal{X}$ *into non-empty sets, and* I *is a function from* $\mathcal{R}$ *into relations on* $D(C(R))$. *For any variable* $X \in \mathcal{X}$, $D(X)$ *is called its* domain. *For any constraint* $R \in \mathcal{R}$, $I(R)$ *is called its* interpretation.

Relations can be defined either extensionally (by enumerating all the tuples in the relation), or intensionally, in general case in the form $\{\underline{x} \mid \phi[\underline{x}]\}$, where $\phi[\underline{x}]$ is a formula in an interpreted language whose all free variables appear in $\underline{x}$ (equivalently, in the form $\{\underline{x} \mid \phi(\underline{x})\}$, where ϕ is a closed predicate abstraction in an interpreted language). Relations with infinite extensions can only be defined intensionally.

Definition 2.3 (Valuation, solution of CN, equivalence of CNs) *Given a constraint network* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{I} \rangle$. *A* valuation *is a function defined on* $\mathcal{X}$, *such that, for any variable* $X \in \mathcal{X}$, $V(X) \in D(X)$. *The set of all valuations is denoted* $\mathcal{V}$. *A valuation* V satisfies *the network, and is called its* solution, *if the*

interpretations of its constraints hold at the values of their var-lists, i.e.

$$\bigwedge_{R \in \mathcal{R}} I(R)(V(C(R))).$$

Two constraint networks are called equivalent, *if they have the same solutions.*

Definition 2.4 (Constraint satisfaction problem (CSP), solution of CSP) *A* constraint satisfaction problem (CSP) *is given by a constraint network* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{I} \rangle$ *and a non-empty set* $\mathcal{X}_a \subseteq \mathcal{X}$ *of* asked variables. *The restriction of* $\mathcal{X}_a$ *of a solution of the constraint network onto* $\mathcal{X}_a$ *is called a* solution *of the CSP.*

It is convenient to represent constraint network schemas in the form of bipartite graphs, where nodes of one sort correspond to variables, nodes of the other sort correspond to constraints, and edges correspond to bindings between variables and constraints. (Given a schema, *bindings* are (unordered) variable-constraint pairs $\langle X, R \rangle$, such that $X \in C(R)$.) For an illustration of this representation, see Fig. 2.1. The information about the order of the variables in the var-list of a constraint is lost in the graph representation.

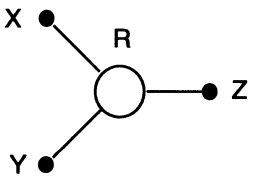

Fig. 2.1. Graph representation of CNSs.
The picture expresses that $C(R) = X, Y, Z$.

Alternatively, constraint network schemas can be represented as hypergraphs, where nodes correspond to variables and hyperedges correspond to constraints, and the incidence function of hyperedges corresponds to the var-list function of the network schema.

2.3.2.2 Equivalent Transformations Transforming a CN into an equivalent CN that meets certain requirements is a constituent of many CSP solving techniques. Typical equivalent transformations can be regarded as being built of augmentation and absorption steps, defined below.

Definition 2.5 (Augmentation step) *Given a constraint network* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{I} \rangle$. *Any set of its constraints* $\underline{R}$, *and any finite list of its variables* $\underline{X}$ *determine an* augmentation step *which amounts to augmenting* $\mathcal{R}$ *with a new constraint* R', *and defining* C *and* I *at* R' *as follows:*

- $C(R') = \underline{X}$,

- $I(R') = \{\underline{x} \mid \exists V \in \mathcal{V}.\, \mathcal{V}(\underline{\mathcal{X}}) = \underline{\S} \wedge \bigwedge_{\mathcal{R} \in \underline{\mathcal{R}}} \mathcal{I}(\mathcal{R})(\mathcal{V}(\mathcal{C}(\mathcal{R})))\}$.

We say that, $\underline{R}$ induces R' *on* $\underline{X}$.

Proposition 2.6 *Given a constraint network, applying any of its augmentation steps yields an equivalent constraint network.*

Proof. For any $V \in \mathcal{V}\mathcal{X}$,

$$\begin{aligned} &\bigwedge_{R \in \mathcal{R}} I(R)(V(C(R))) \equiv \\ &\quad \equiv \bigwedge_{R \in \mathcal{R}} I(R)(V(C(R))) \wedge \bigwedge_{R \in \underline{R}} I(R)(V(C(R)) \\ &\quad \equiv \bigwedge_{R \in \mathcal{R} \cup \{\mathcal{R}'\}} I(R)(V(C(R))), \end{aligned}$$

and hence the original and the modified CN have the same solutions.

Definition 2.7 (Absorption step) *Given a constraint network* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{I} \rangle$. *Any two of its constraints* R_0, R', *with var-lists* $\underline{X_0}, \underline{X'}$, *such that:*

- $\underline{X_0} \subseteq \underline{X'}$,
- $I(R') \subseteq \{\underline{x} \mid \exists V \in \mathcal{V}.\, \mathcal{V}(\underline{\mathcal{X}}) = \underline{\S} \wedge \mathcal{I}(\mathcal{R}_\prime)(\mathcal{V}(\underline{\mathcal{X}_\prime}))\}$,

determine an absorption step. *This step amounts to deleting* R_0 *from* $\mathcal{R}$. *We say that* R' subsumes R_0.

Proposition 2.8 *Given a constraint network, applying any of its absorption steps yields an equivalent constraint network.*

Proof. Omitted.

Example 2.9 Given a CN and two of its constraints R_1, R_2 with a common var-list $\underline{X}$. The interpretation of the constraint R' that R_1, R_2 induce on $\underline{X}$ is $I(R') = I(R_1) \cap I(R_2)$. R' subsumes both R_1 and R_2. Replacing R_1, R_2 with R' yields an equivalent CN.

Example 2.10 Given a CN and a finite list of its variables $\underline{X} \in \mathcal{X}^*$, the interpretation of the constraint R' induced by the empty set of constraints on $\underline{X}$ is $I(R') = D(\underline{X})$ (the total relation). Adding R yields an equivalent CN.

Often, it is assumed about a constraint network that, for any finite list of variables $\underline{X} \in \mathcal{X}^*$ (or, for any set of variables $\underline{X} \subseteq \mathcal{X}$), there is at most one (or even exactly one) constraint $R \in \mathcal{R}$ with list $\underline{X}$ (or a fixed listing of set $\underline{X}$) as its var-list. Modifications in the spirit of the previous examples allow to transform any constraint network into such a form.

Example 2.11 Given a CN and a non-empty set of its constraints $\underline{R} \cup \{R_0\} \subseteq \mathcal{R}$, with the var-list of R_0 being $\underline{X_0}$. Let R' be a name for the constraint these constraints induce on $\underline{X_0}$. Then

$$I(R') = I(R_0) \cap \{\underline{x} \mid \exists V \in \mathcal{V}.\, \mathcal{V}(\underline{\mathcal{X}_l}) = \underline{\S} \wedge \bigwedge_{\mathcal{R} \in \underline{\mathcal{R}}} \mathcal{I}(\mathcal{R})(\mathcal{V}(\mathcal{C}(\mathcal{R})))\}.$$

R' subsumes R_0. Replacing R_0 with R' yields an equivalent CN.

In the previous example, R_0 is replaced with a constraint R', whose interpretation takes into account the restrictions that constraints $\underline{R}$ impose on variables in $\underline{X_0}$, and is, therefore, tighter than the interpretation of R_0. Such explication transformation might be called a *consistency-enforcement step*, and it is the basic instrument of the numerous consistency techniques. Their underlying idea is to bring an original network into a shape from which the solutions of a problem can be read off with the least pain. This is accomplished by gradually explicating the network by a series of consistency-enforcement steps.

Example 2.12 Given a CN. Let R' be a name for the constraint that the set $\mathcal{R}$ of all constraints of the CN induces on an enumeration $\underline{X}$ of the set $\mathcal{X}$ of all variables of the CN. Then

$$I(R') = \{\underline{x} \mid \exists V \in \mathcal{V}.\, \mathcal{V}(\underline{\mathcal{X}}) = \underline{\S} \wedge \bigwedge_{\mathcal{R} \in \mathcal{R}} \mathcal{I}(\mathcal{R})(\mathcal{V}(\mathcal{C}(\mathcal{R})))\}.$$

R' subsumes all the constraints in $\mathcal{R}$. Replacing $\mathcal{R}$ with R' yields an equivalent CN which has exactly one constraint R'. R' *represents* the solutions of the CN.

2.3.3 Functional Constraint Networks

2.3.3.1 Basic Definitions Functional constraint networks are a specialization of constraint networks, where constraints can only be interpreted as functional relations.

Definition 2.13 (Functional relation) *A $(n+m)$-ary relation P on $D_1 \times D_2$ is* functional *if it is a graph of a (total) function from D_1 into D_2, i.e. if*

$$\forall \underline{x} \in D_1 . \exists ! \underline{y} \in D_2 . P(\underline{x}, \underline{y}).$$

Functional relations are typically defined intensionally, generally in the form $\{\underline{x}, \underline{y} \in | \; f[\underline{x}] = \underline{y}\}$, where $f[\underline{x}]$ is a term in an interpreted language whose all free variables appear in $\underline{x}$ (equivalently, in the form $\{\underline{x}, \underline{y} \in | \; f(\underline{x}) = \underline{y}\}$, where f is a closed function abstraction in an interpreted language).

For any function f, let us denote its graph by $\lceil f \rceil$.

The definitions of CNS and CN are specialized in the following way.

Definition 2.14 (Functional constraint network schema (FCNS)) *A* functional constraint network schema (FCNS) *is a triple* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C} \rangle$, *where* $\mathcal{X}$ *is a finite non-empty set of* variables, $\mathcal{R}$ *is a finite non-empty set of* constraints, $\mathcal{X} \cap \mathcal{R} = \emptyset$, *and* C *is a pair of functions* $\langle C_{\text{in}}, C_{\text{out}} \rangle$, *both from* $\mathcal{R}$ *into finite lists over* $\mathcal{X}$. *For any constraint* $R \in \mathcal{R}$, $C_{\text{in}}(R)$ *is called its* input-var-list, *and* $C_{\text{out}}(R)$ *is called its* output-var-list.

Below, a writing $C(R) = \underline{X} \to \underline{Y}$ will be often used as a shortcut notation for stating that $C_{\text{in}}(R) = \underline{X}$ and $C_{\text{out}}(R) = \underline{Y}$.

Definition 2.15 (Functional constraint network (FCN)) *A* functional constraint network (FCN) *is a quintuple* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{I} \rangle$, *where* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C} \rangle$ *is a functional constraint network schema,* D *is a function from* $\mathcal{X}$ *into non-empty sets, and* I *is a function from* $\mathcal{R}$ *into functional relations on* $D(C_{\text{in}}(R), C_{\text{out}}(R))$. *For any variable* $X \in \mathcal{X}$, $D(X)$ *is called its* domain. *For any constraint* $R \in \mathcal{R}$, $I(R)$ *is called its* interpretation.

Functional constraint network schemas are best represented in the form of directed bipartite graphs, where arcs correspond to directed bindings between variables and constraints. (Given a schema, *directed bindings* are variable-constraint pairs $\langle X, R \rangle$, where $X \in C_{\text{in}}(R)$, and constraint-variable pairs $\langle R, X \rangle$, where $X \in C_{\text{out}}(R)$). For an illustration of this representation, see Fig. 2.2.

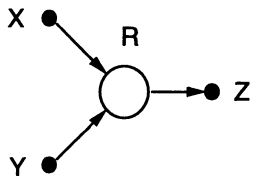

Fig. 2.2. Graph representation of FCNSs.
The picture expresses that $C(R) = X, Y \to Z$.

2.3.3.2 Equivalent Transformations Normally, functional relations are "implemented" as functions, which means that they are meant to be used "in one direction" — for producing an output value for given input values —, and not for producing input values for a given output value, for checking where a tuple values belongs to the graph of the function, or for anything else.

When solving a FCN whose constraints are efficiently implemented in one direction, we want to take advantage of it. Therefore, we are interested in equivalent transformations of FCNs that preserve efficiently implemented functionality of constraints. Not all augmentation and absorption steps do so. E.g. the transformation of Example 2.9 need not necessarily preserve functionality, since

the intersection of the graphs of two functions need not generally be the graph of a function. The constraint R' induced on X by two functional constraints R_1, R_2 with $C(R_1) = X \to Y, I(R_1) = \lceil f \rceil$, $C(R_2) =\to Y, I(R_2) = \lceil g \rceil$, may be a non-functional relation (if f is not one-to-one), and even if it is a functional relation (constant $\lceil f^{-1}(g) \rceil$), its implementation may not be directly derivable from those of R_1, R_2.

The following proposition concerns five important types of augmentation steps of FCNs, and demonstrates that they preserve efficiently implemented functionality. They are illustrated in Fig. 2.3.

Proposition 2.16 *Given a functional constraint network* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{I} \rangle$.

1. *For any two constraints* R_1, R_2 *with* $C(R_1) = \underline{Z} \to \underline{X}$, $C(R_2) = \underline{X} \to \underline{Y}$, *and* $I(R_1) = \lceil f \rceil$, $I(R_2) = \lceil g \rceil$, *the interpretation of the constraint* R' *induced by* $\{R_1, R_2\}$ *on* $\underline{Z} \to \underline{Y}$ *is*

$$I(R') = \lceil g \circ f \rceil.$$

2. *For any constraint* R *with* $C(R) = \underline{Z} \to \underline{X}, \underline{Y}$, *where* $\#\underline{X} = m_1, \#\underline{Y} = m_2$, *and* $I(R) = \lceil f \rceil$, *the interpretation of the constraint* R' *induced by* $\{R\}$ *on* $\underline{Z} \to \underline{X}$ *is*

$$I(R') = \lceil \mathrm{select}_{1 \ldots m_1}^{m_1 + m_2} \circ f \rceil.$$

3. *For any two constraints* R_1, R_2 *with* $C(R_1) = \underline{Z} \to \underline{X}$, $C(R_2) = \underline{Z} \to \underline{Y}$, *and* $I(R_1) = \lceil f \rceil$, $I(R_2) = \lceil g \rceil$, *the interpretation of the constraint* R' *induced by* $\{R_1, R_2\}$ *on* $\underline{Z} \to \underline{X}, \underline{Y}$ *is*

$$I(R') = \lceil (f, g) \rceil.$$

4. *For any constraint* R *with* $C(R) = \underline{Z} \to \underline{X}$, *where* $\#\underline{Z} = n_1$ *and* $I(R) = \lceil f \rceil$, *and any finite list of variables* $\underline{W}$, *where* $\#\underline{W} = n_2$, *the interpretation of the constraint* R' *induced by* $\{R\}$ *on* $\underline{Z}, \underline{W} \to \underline{X}$ *is*

$$I(R') = \lceil f \circ \mathrm{select}_{1 \ldots n_1}^{n_1 + n_2} \rceil.$$

5. *For any finite list of variables* $\underline{Z}$, *where* $\#\underline{W} = n$, *the interpretation of the constraint* R' *which the empty set of constraints induces on* $\underline{Z} \to \underline{Z}$ *is*

$$I(R') = \lceil \mathrm{id}^n \rceil.$$

Proof.

1.

$$\begin{aligned} I(R') = & \\ = & \{\underline{z}, \underline{y} \mid \exists x. I(R_1)(\underline{z}, \underline{x}) \wedge I(R_2)(\underline{x}, \underline{y})\} \\ = & \{\underline{z}, \underline{y} \mid \exists x. f(\underline{z}) = \underline{x} \wedge g(\underline{x}) = \underline{y}\} \\ = & \{\underline{z}, \underline{y} \mid g(f(\underline{z})) = \underline{y}\} \\ = & \lceil g \circ f \rceil. \end{aligned}$$

Proof of (2–4) omitted.

The following two examples show that, on a FCN, the augmentation substeps of the traditional arc- and path-consistency-enforcement steps computationally amount to application of a function on a constant and to composition of two functions, respectively. This is illustrated in Fig. 2.4.

Example 2.17 Given a FCN and two of its constraints R_1, R_2 with $C(R_1) = \rightarrow X$, $C(R_2) = X \rightarrow Y$, and $I(R_1) = \lceil f \rceil$, $I(R_2) = \lceil g \rceil$ (where f is a 0-ary function, i.e. constant, and g is a unary function). The interpretation of the constraint R' which R_1, R_2 induce on $\rightarrow Y$ is

$$I(R') = \lceil g(f) \rceil.$$

Example 2.18 Given a FCN and two of its constraints R_1, R_2 with $C(R_1) = Z \rightarrow X$, $C(R_2) = X \rightarrow Y$, and $I(R_1) = \lceil f \rceil$, $I(R_2) = \lceil g \rceil$ (where f and g are unary functions). The interpretation of the constraint R' which R_1, R_2 induce on $Z \rightarrow Y$ is

$$I(R') = \lceil g \circ f \rceil.$$

2.3.3.3 Relation to Logic Constraint networks have various logical interpretations (Mackworth 1992). Value propagation on functional constraint networks has a good explanation in terms of constructive logic. All augmentation steps of Proposition 2.16 can be regarded as valid proof rules of intuitionistic propositional logic annotated with realizations of formulae (I/O-var-lists correspond to formulae, interpretations correspond to realizations) :

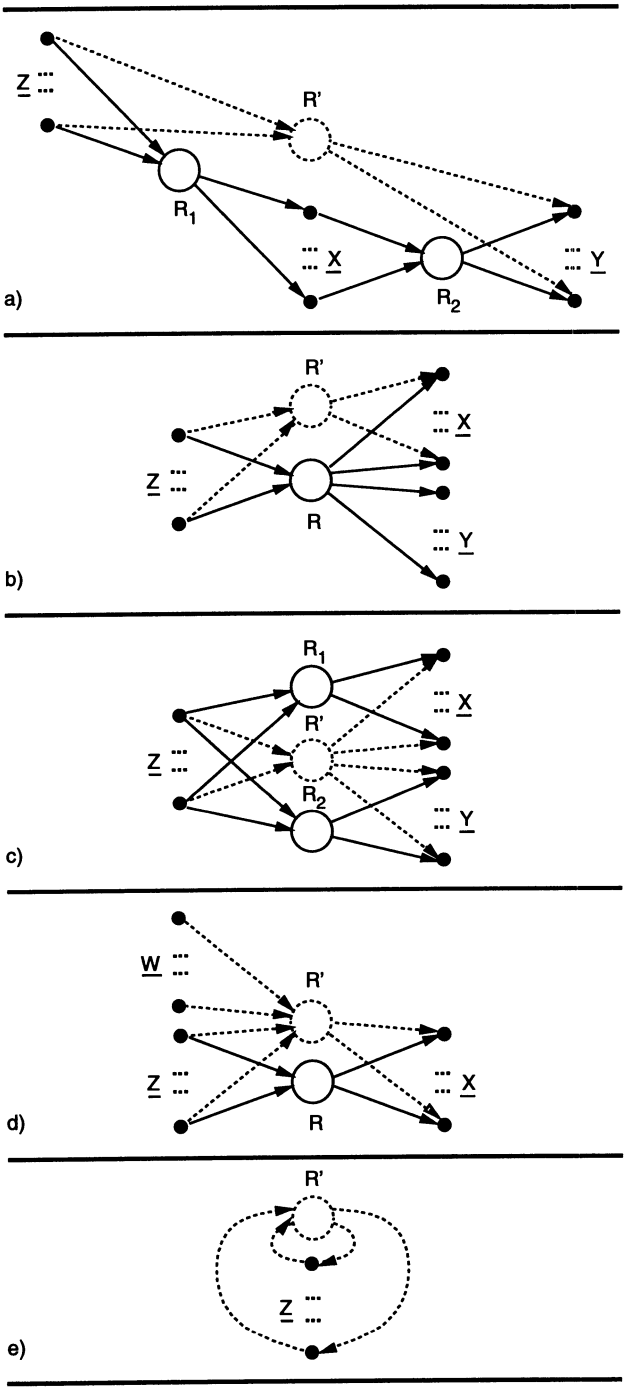

Fig. 2.3. Augmentation steps for FCNs.

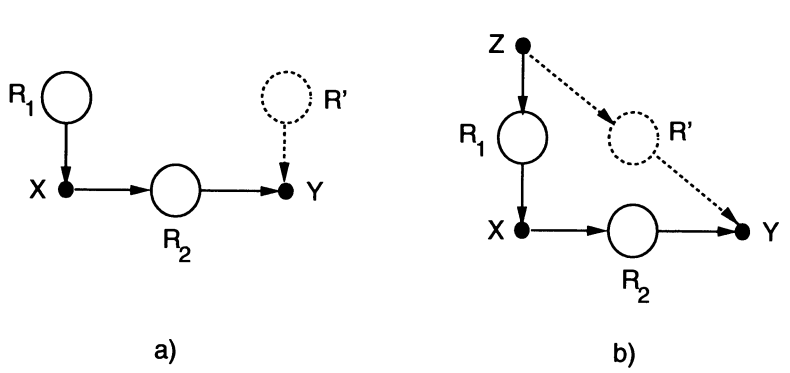

Fig. 2.4. Arc- and path-augmentation steps for FCNs.

$$\text{I1:}\quad \frac{f\ \mathbf{r}\ \underline{Z} \to \underline{X} \qquad g\ \mathbf{r}\ \underline{X} \to \underline{Y}}{g \circ f\ \mathbf{r}\ \underline{Z} \to \underline{Y}}$$

$$\text{I2:}\quad \frac{f\ \mathbf{r}\ \underline{Z} \to \underline{X}, \underline{Y}}{\mathrm{select}_{1\ldots m_1}^{m_1+m_2} \circ f\ \mathbf{r}\ \underline{Z} \to \underline{X}}$$

$$\text{I3:}\quad \frac{f\ \mathbf{r}\ \underline{Z} \to \underline{X} \qquad g\ \mathbf{r}\ \underline{Z} \to \underline{Y}}{(f,g)\ \mathbf{r}\ \underline{Z} \to \underline{X}, \underline{Y}}$$

$$\text{I4:}\quad \frac{f\ \mathbf{r}\ \underline{Z} \to \underline{X}}{f \circ \mathrm{select}_{1\ldots n_1}^{n_1+n_2}\ \mathbf{r}\ \underline{W}, \underline{Z} \to \underline{X}}$$

$$\text{I5:}\quad \frac{}{\mathrm{id}^n\ \mathbf{r}\ \underline{Z} \to \underline{Z}}$$

Rule (I1) is a version of implication elimination, (I2–3) are elimination and introduction rules of conjunction, (I4) is a weakening rule on the left, and (I6) is an identity axiom.

2.3.3.4 Value Propagation The underlying idea of value propagation is in finding an induced (0+1)-ary functional constraint on $\to X$ for as many asked variables X of a given CSP as possible. These constraints would tell what values the variables necessarily have to have in every solution of the CSP (if there are any solutions at all).

Proposition 2.19 *Given a FCN $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{I} \rangle$ with a constraint R', such that $C(R') =\!\to X$ and $I(R') = \lceil v \rceil$ (v is a constant), then in every solution V of the FCN, $V(X) = v$.*

Proof. If V is a solution, then $I(R')(V(X))$ must hold, and this is equivalent to $V(X) = v$.

Only augmentation steps of Proposition 2.16 are used in value propagation. Since their applicability depends only on the I/O-var-lists of the inducing constraints and not on their interpretation, value propagation can be pre-planned on a FCNS. We shall soon see that various efficient algorithms exist for value propagation planning. But, first, we will point out the important limitations of the technique.

If the value of a certain variable X is not uniquely determined by the constraints in a network, then no functional constraint can be induced on $\rightarrow X$, and value propagation yields no information about that particular variable. The same can happen, if the value is uniquely determined just due to a particular interpretation of constraints, since the purely schema-based planning of value propagation uses no information about the interpretation of constraints. (Example: equation systems.)

If two or more functional constraints with different interpretations are induced on $\rightarrow X$, then the FCN is clearly unsatisfiable. Most value propagation algorithms find just one induced constraint for each asked variable, and do not thus detect unsatisfiablity.

It is sufficient to use only the rules (I1–I3) with empty $\underline{Z}$ in "first-order" value propagation, augmenting the original network only with constraints whose input-var-lists are empty. Production of any such new constraint means that its output variables have become "known".

The following is the naive algorithm for value propagation planning of FCNs.

Algorithm 2.20 (Value propagation planning on FCNSs, ver 1)

```
proc naive-propag(𝒳, ℛ, C, 𝒳a, plan, succeeded) is
  𝒳k := ∅;
  initlist(plan);
  succeeded := false;
  changes := true;
  while changes and not succeeded do
    changes := false;
    for R ∈ ℛ do
      if Cin(R) ⊆ 𝒳k and Cout(R) ⊈ 𝒳k then
        changes := true;
        plan := addto(plan, R);
        𝒳k := 𝒳k ∪ Cout(R);
        if 𝒳a ⊆ 𝒳k then
          succeeded := true
        endif
      endif
    endfor
  endwhile
endproc
```

The time-complexity of Algorithm 2.20 is $O(N^2)$, where N is the number of constraints of a FCNS.

Search for appropriate inducing constraints can be guided by counters which show the number of still unknown input variables of a constraint, as done in the following algorithm.

Algorithm 2.21 (Value propagation planning on FCNSs, ver 2)

```
proc propag(𝒳, ℛ, 𝒞, 𝒳a, plan, succeeded) is
  𝒳k := ∅;
  initqueue(beam);
  initlist(plan);
  succeeded := false;
  for R ∈ ℛ do
    c[R] := #C_in(R);
    if c[R] = 0 then
      enqueue(R, beam)
    endif
  endfor;
  while not emptyqueue(beam) and not succeeded do
    R := dequeue(beam);
    if C_out(R) ⊈ 𝒳k then
      for X ∈ C_out(R) − 𝒳k do
        for R' ∈ C_in^-1(X) do
          c[R'] := c[R'] − 1;
          if c[R'] = 0 then
            enqueue(R, beam)
          endif
        endfor;
        𝒳k := 𝒳k ∪ {X};
      endfor;
      plan := addto(plan, R);
      if 𝒳a ⊆ 𝒳k then
        succeeded := true
      endif
    endif
  endwhile
endproc
```

Proposition 2.22 *The time-complexity of Algorithm 2.21 is $O(L)$, where L is the number of bindings in a given FCNS.*

Proof. For any problem, all operations in the algorithm are performed at most once either per binding of type variable-constraint, or per binding of type constraint-variable, or per constraint, according to the following division:

Binding $\langle X, R\rangle$: During initializations: check if $X \notin \mathcal{X}_k$, increase $c[R]$ by one. During search: decrease $c[R]$ by one, check if $c[R] = 0$, set $X \in \mathcal{X}_k$.

Binding $\langle R, X\rangle$: During search: check if $X \notin \mathcal{X}_k$.

Constraint R : During initializations or search: enqueue R. During search: dequeue R, add R to plan, check if $\mathcal{X}_a \subseteq \mathcal{X}_k$.

The key observations to be made are that every constraint is enqueued at most once, and that every variable is marked as having become known at most once.

Value propagation planning algorithms with time-complexity linear in the number of bindings that make use of counters have been proposed independently by several people. The idea for Algorithm 2.21 originated from (Dikovski 1985).

2.3.4 Higher-Order Functional Constraint Networks

2.3.4.1 Basic Definitions We shall now make the following generalization: we will allow variables to take higher-order (relational) values. Higher-order variables will behave like ordinary variables in that they are uninterpreted in a network. From the other side, they will behave like constraints in that they will have var-lists. Their values are functional relations. Conceptually, higher-order variables will serve as names for constraints inducible by the network.

Definition 2.23 (Higher-order functional constraint network schema (HFCNS)) *A* higher-order functional constraint network schema (HFCNS) *is a triple* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}\rangle$, *where* $\mathcal{X}$ *is a finite non-empty set of* variables, *with a distinguished subset* $\mathcal{S}$ *of* higher-order variables, $\mathcal{R}$ *is a finite non-empty set of* constraints, $\mathcal{X} \cap \mathcal{R} = \emptyset$, *and* C *is a pair of functions* $\langle C_{in}, C_{out}\rangle$, *both from* $\mathcal{R} \cup \mathcal{S}$ *into finite lists over* $\mathcal{X}$. *For any constraint* $R \in \mathcal{R}$ *(any higher-order variable* $S \in \mathcal{S}$*),* $C_{in}(R)$ *(*$C_{in}(S)$*) is called its* input-var-list, *and* $C_{out}(R)$ *(*$C_{out}(S)$*)is called its* output-var-list.

Definition 2.24 (Functional constraint network (HFCN)) *A* higher-order functional constraint network (HFCN) *is a quintuple* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{D}, \mathcal{I}\rangle$, *where* $\langle \mathcal{X}, \mathcal{R}, \mathcal{C}\rangle$ *is a HFCNS,* D *is a function from* $\mathcal{X}$ *into non-empty sets, satisfying the condition that, for any* $S \in \mathcal{S}$, $D(S)$ *is the set of all functional relations on* $D(C_{in}(S), C_{out}(S))$, *and* I *is a function from* $\mathcal{R}$ *into functional relations on* $D(C_{in}(R), C_{out}(R))$. *For any variable* $X \in \mathcal{X}$, $D(X)$ *is called its* domain. *For any constraint* $R \in \mathcal{R}$, $I(R)$ *is called its* interpretation.

Definition 2.25 (Solution of HFCN) *A valuation* V satisfies *the network, and is called its* solution, *if*

$$\bigwedge_{R \in \mathcal{R}} I(R)(V(C(R))) \wedge \bigwedge_{S \in \mathcal{S}} V(S)(V(C(S))).$$

In fact, we look only for constructive solutions in which the values of higher-order variables are induced by the network. This means that whenever a functional constraint R' with $I(R') = \lceil f \rceil$ is inducible on $\underline{Z}, \underline{X} \to \underline{Y}$ and $C(S) = X \to Y$, then $V(S) = \lceil \lambda \underline{x}.f(V(\underline{Z}, \underline{x})) \rceil$. This is conceptually and logically motivated, and in concert with the principle of efficient implementability of functional relations.

Similarly to ordinary FCNSs, HFCNSs can be represented graphically, but the graphs are not bipartite in general. An example of a HFCN can be found in the calculus, where higher-order variables are connected by various constraints. Fig. 2.5 shows a HFCNS, where functions $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}', \boldsymbol{y}', \boldsymbol{f}', \boldsymbol{f}_{xy}$ are connected by superposition, integration, and differentiation constraints, and these functions themselves connect first-order variables t, x, y, f. As before, arrows show directions of bindings.

Below, we will restrict our study to HFCNs, where the var-lists of higher-order variables may only contain first-order variables (i.e. they are, in fact, second-order), and they themselves cannot appear in the output-var-lists of constraints. This does not restrict generality. Any HFCN can be transformed into this form by using abstraction and application operators as additional constraints. When we solve a CSP on a HFCN, and a second-order variable S has to be computed, we have to solve a problem of finding $C_{\text{out}}(S)$ from given $C_{\text{in}}(S)$. This is a CSP in its own right. Therefore, in our restricted HFCNs, second-order variables are also called *subproblems.*

There is an alternative notation for subproblem nodes in graphs shown in Fig. 2.6, where the direction of arcs is motivated by dataflow of computations. The dataflow direction for a subproblem S is from S to $C_{\text{in}}(S)$, and from $C_{\text{out}}(S)$ to S. The value of S in a solution to the network in Fig. 2.7 represents the constraint induced on $U \to V$ by constraints $\{R_1, R_2\}$. The interpretation of constraint R represents a functional, which uses the value of S for producing Y from X. In this process, the value of V is computed for as many values of U as needed, i.e. the sequence R_1, R_2 may be called several times.

Under the stipulation that we are only interested in constructive solutions, the following holds on HFCNs. Given a constraint R with $C(R) = \underline{Z}, \underline{X} \to \underline{Y}$ and $I(R) = \lceil f \rceil$, and a second-order variable S with $C(S) = \underline{X} \to \underline{Y}$, the interpretation of the constraint R' induced by $\{R\}$ on $\to S$ is $I(R') = \lceil \lambda \underline{x}.(\lambda \underline{z}.f(\underline{z}, \underline{x})) \rceil$. This augmentation step is illustrated in Fig. 2.8,*a*.

2.3.4.2 Relation to Logic The augmentation step for subproblems can be regarded as the following valid proof rule of intuitionistic propositional logic annotated with realizations of formulae.

$$\text{I6:} \quad \frac{f \mathbf{r} \underline{Z}\underline{X}, \to \underline{Y}}{\lambda \underline{z}.(\lambda \underline{x}.f(\underline{z}, \underline{x})) \mathbf{r} \underline{Z} \to S} \text{ if } S \text{ is definitionally equivalent to } \underline{X} \to \underline{Y}$$

The proof rules (I1–I6) form a complete set of rules for the intuitionistic propositional logic in the following sense. Any finite set of propositional formulae can be translated into an intuitionistically deductively equivalent set of formulae in our (implicative) language (Mints, Tyugu 1983).

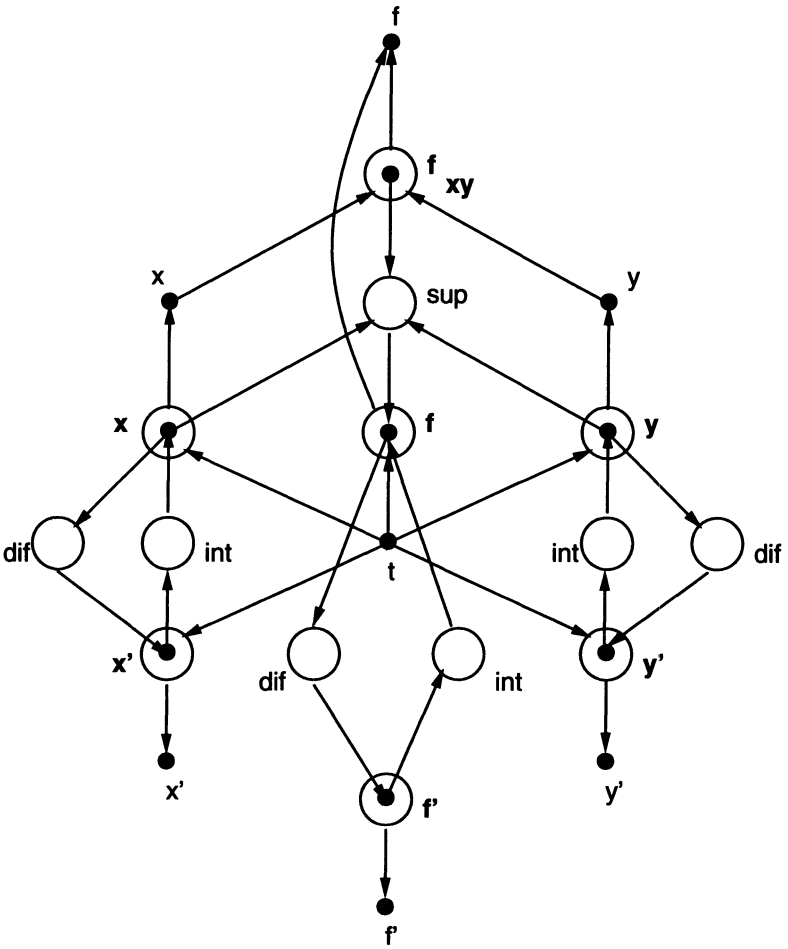

Fig. 2.5. The schema of a HFCN from the calculus.

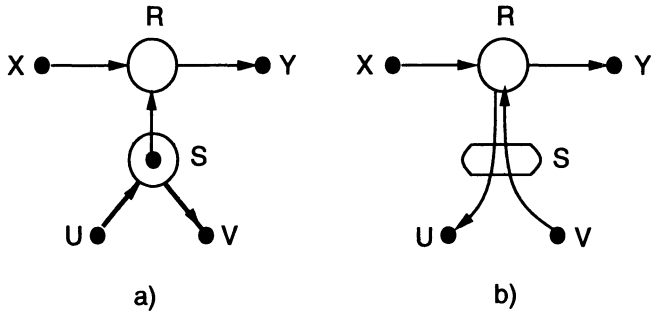

Fig. 2.6. A constraint with a subproblem ($C(R) = S, X \to Y, C(S) = U \to V$). Two graphical notations.

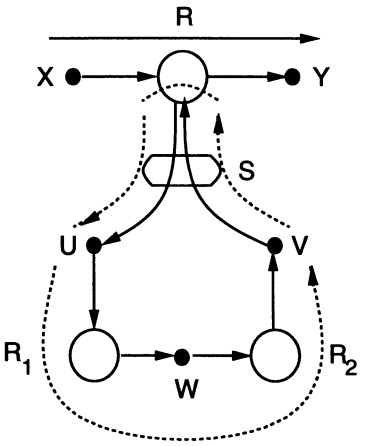

Fig. 2.7. Dataflow through a constraint with a subproblem.

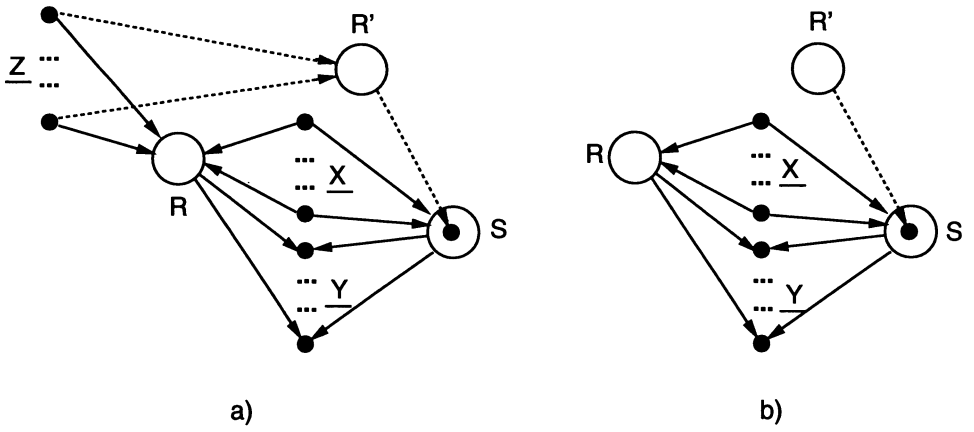

Fig. 2.8. The augmentation step for subproblems. *a)* general case, *b)* independent subproblems.

2.3.4.3 Value propagation Value propagation on HFCNSs is based on rules (I1-I6). The general problem of propagation planning on HFCNSs is PSPACE-complete, as it will be stated later.

Let us define the degree of subproblem dependence as the largest number of other subproblems whose inputs may be used when solving a subproblem. Fortunately, in the constraint networks which turn up in practice there are usually few subproblems and these do not interact very deeply. Thus, algorithms that work just for a fixed small degree of dependence of subproblems are practically meaningful and applicable.

If the degree of subproblem dependence is 0, then we talk about independent subproblems. In case of independent subproblems, rule (I6) can be applied with empty list $\underline{Z}$ only (see Fig. 2.8,*b*). Search for a propagation plan for a problem with M independent subproblems can be organized in the following way. We construct the main plan in parallel with construction of M subplans, each for one subproblem, with the following interactions between these parallel processes. If a subproblem is completed, the corresponding variable is marked as known in the process of the main problem. Whenever some variable becomes known for the main problem, it is also marked as known in the processes of all the subproblems. If the search fails, no plan for the problem can exist.

The following is the formal description of the algorithm.

Algorithm 2.26 Value propagation planning on HFCNSs, independent subproblems

---

proc initialize($\mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{X}_k, \rfloor$, beam, plan, succeeded) is
  initqueue(beam);
  initlist(plan);
  succeeded := false;
  for $R \in \mathcal{R}$ do
    $c[R] := \#(C_{in}(R) - \mathcal{X}_k)$;
    if $c[R] = 0$ then
      enqueue(R, beam)
    endif
  endfor
endproc

---

proc recordnew($\mathcal{X}, \mathcal{R}, \mathcal{C}, \mathcal{X}_k, \rfloor$, beam, X) is
  for $R' \in C_{in}^{-1}(X)$ do
    $c[R'] := c[R'] - 1$;
    if $c[R'] = 0$ then
      enqueue(R', beam)
    endif
  endfor;
  $\mathcal{X}_k := \mathcal{X}_k \cup \{X\}$
endproc

```
proc search(𝒳, ℛ, 𝒞, 𝒳a, 𝒳k, ⌋, beam, plan, changes, succeeded) is
   while not empty(beam) and not succeeded do
      R := dequeue(beam);
      if Cout(R) ⊈ 𝒳k then
         for X ∈ Cout(R) − 𝒳k do
            recordnew(𝒳, ℛ, 𝒞, 𝒳k, ⌋, beam, X)
         endfor;
         plan := addto(plan, R);
         changes := true;
         if 𝒳a ⊆ 𝒳k then
            succeeded := true
         endif
      endif
   endwhile
endproc
```

```
proc propag-indep(𝒳, ℛ, 𝒞, 𝒳a, plan, succeeded) is
   𝒳k := ∅;
   initialize(𝒳, ℛ, 𝒞, 𝒳k, ⌋, beam, plan, succeeded);
   for S ∈ 𝒮 do
      𝒳aS := Cout(S);
      𝒳kS := Cin(S);
      initialize(𝒳, ℛ, 𝒞, 𝒳kS, ⌋S, beamS, planS, succeededS)
   endfor;
   changes := true;
   while changes and not succeeded do
      changes := false;
      search(𝒳, ℛ, 𝒞, 𝒳a, 𝒳k, ⌋, beam, plan, changes, succeeded);
      if not succeeded then
         for S ∈ 𝒮 do
            for X ∈ 𝒳k − 𝒳kS do
               recordnew(𝒳, ℛ, 𝒞, 𝒳kS, ⌋S, beamS, X)
            endfor;
            search(𝒳, ℛ, 𝒞, 𝒳aS, 𝒳kS, ⌋S, beamS, planS, changes, succeededS); if succeededS
            then
               recordnew(𝒳, ℛ, 𝒞, 𝒳k, ⌋, beam, S);
               plan := addto(plan, planS)
            endif
         endfor
      endif
   endwhile
endproc
```

Proposition 2.27 *The time-complexity of Algorithm 2.26 is $O(ML)$, where M is the number of subproblems and L is the number of bindings of a HFCNS.*

Proof. In this algorithm, $1+M$ plans are built pseudoparallelly: one main plan plus, for every $S \in \mathcal{S}$, a plan for finding $C_{\text{out}}(S)$ from $C_{\text{in}}(S)$. Each individual plan is formed on a separate copy of the network schema, essentially in the same way as in Algorithm 2.21.

Proposition 2.28 *Given d, a problem on a HFCNS with M subproblems and L bindings can be planned/detected to be unsolvable with the degree of subproblem dependence d in time $O(M^{d+1}L)$.*

Proof. For any given d, there is a suitable generalization of Algorithm 2.26: The following plans have to be built pseudoparallelly on separate copies of the network schema: one main plan plus, for every $S \in \mathcal{S}$ and every set $\{S_1, \ldots, S_k\} \subseteq \mathcal{S} - \{S\}$, where $k \leq d$, a plan for finding $C_{\text{out}}(S)$ from $C_{\text{in}}(S_1), \ldots, C_{\text{in}}(S_k), C_{\text{in}}(S)$. The total number of plans involved in the algorithm is $1 + M(\binom{M}{0} + \ldots + \binom{M}{d}) = O(M^{d+1})$.

Remark. Solvable problems with M subproblems are solvable with the degree of subproblem dependence $d = M - 1$.

Proposition 2.29 *The (general) problem of planning on HFCNSs is PSPACE-complete.*

Proof. The problem of derivability in intuitionistic propositional calculus is PSPACE-complete, and is polynomially reducible to the problem of planning on HFCNSs, as shown in (Mints, Tyugu 1983).

Kanovich (1991) has described and thoroughly analyzed a number of algorithms for what he calls program synthesis on relational knowledge bases, and what is essentially the same as propagation planning on HFCNSs. In particular, he has shown that linear space is sufficient for the general planning problem (with no limit on the dependence of subproblems).

2.3.4.4 Examples Let us study two examples of value propagation on HFCNs with dependent subproblems.

Example 2.30 ("Double integration") Consider the HFCNS with five first-order variables X, Y, A, B, D, two second-order variables S_1, S_2, and three constraints R_1, R_2, R_3, presented in Fig. 2.9. The var-lists of the constraints and second-order variables are as follows:

$$
\begin{aligned}
C(R_1) &= S_1 \to D, \\
C(S_1) &= Y \to B, \\
C(R_2) &= S_2 \to B, \\
C(S_2) &= X \to A, \\
C(R_3) &= X, Y \to A.
\end{aligned}
$$

This network can have several simple interpretations. For instance, it can embody a scenario of double integration in bounds $0 \ldots 1$ of a binary function on reals, if its constraints are interpreted as follows:

$$
\begin{aligned}
I(R_1) = I(R_2) &= \lceil \lambda\phi. \int_0^1 \phi \rceil, \\
I(R_3) &= \lceil f \rceil.
\end{aligned}
$$

A reasonable CSP to be solved on this HFCN is to ask the value of the double integral, i.e. D.

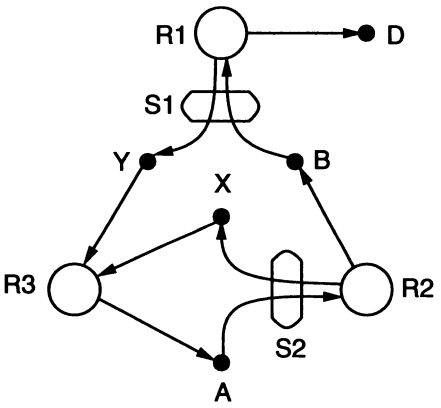

Fig. 2.9. The constraint network schema to Example "Double integration".

This HFCNS contains a dependent subproblem of finding A, given X. It can be solved only when solving of the other subproblem has started, because the value of Y is needed for producing the A. However, it is not hard to find the proper order of propagations on this constraint network for finding the value of D.

Value propagation plans for HFCSPs are labelled trees. One value propagation plan tree for this problem is presented in Fig. 2.10.

Every constructive solution V of this network has the following form:

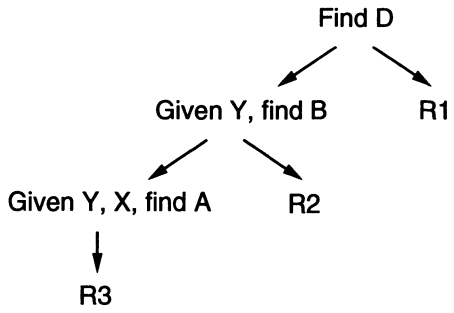

Fig. 2.10. A plan tree for Example "Double integration".

$$
\begin{aligned}
V(X) &= x_0, \\
V(Y) &= y_0, \\
V(A) &= f(x_0, y_0), \\
V(S_2) &= \lceil \lambda x. f(x, y_0) \rceil, \\
V(B) &= \int_0^1 f(x, y_0) \mathrm{d}x, \\
V(S_1) &= \lceil \lambda y. \int_0^1 f(x, y) \mathrm{d}x \rceil, \\
V(D) &= \int_0^1 \int_0^1 f(x, y) \mathrm{d}x \mathrm{d}y.
\end{aligned}
$$

Only S_1, D are uniquely determined. And they are the only variables for whom values can be found using value propagation.

Example 2.31 ("Kripke") Consider the HFCNS with four first-order variables X, Y, A, B, two second-order variables S_1, S_2, and three constraints R_1, R_2, R_3, presented in Figure 2.11. The var-lists of the constraints are as follows:

$$
\begin{aligned}
C(R_1) &= S_1 \to B, \\
C(S_1) &= Y \to A, \\
C(R_2) &= S_2 \to X, \\
C(S_2) &= A \to B, \\
C(R_3) &= X, Y \to A.
\end{aligned}
$$

The intended domain for X, Y, A, B is the set of reals. The constraints are meant to be interpreted as follows:

$$
\begin{aligned}
I(R_1) = I(R_2) &= \lceil \lambda \phi. \sum_{i=1}^{3} \phi(i) \rceil, \\
I(R_3) &= \lceil \lambda x, y. x \cdot y \rceil.
\end{aligned}
$$

The problem is to find a value for B.

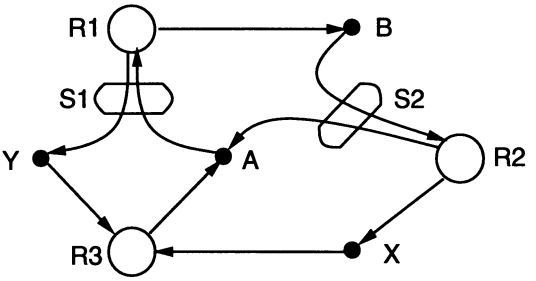

Fig. 2.11. The constraint network schema for Example "Kripke".

At the first glance, it may seem that this problem is unsolvable. However, this intuition is wrong. One possible plan tree for this problem is presented in Fig. 2.12. Executing the plan, we derive that B must be 108. The corresponding proof tree in our variant of intuitonistic propositional logic is in Fig. 2.13. The trick is that the constraint R_1 has to be used twice, first only for helping to solve a subproblem of the constraint R_2, and, second, for solving the main problem.

We have to note that there are other, alternative plans for solving this problem, and they produce other "only possible" values of B than 108. This shows that the HFCN is, in fact, inconsistent. Value propagation techniques do not detect that, and are not supposed to do so either.

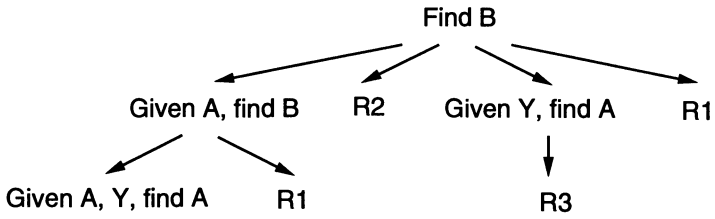

Fig. 2.12. A plan tree to Example "Kripke".

2.3.5 Conclusions

In this paper, we have shown that value propagation is a sufficient and efficiently implementable constraint satisfaction technique for a certain class of CSPs (namely, finding the values of the uniquely determined variables on satisfiable functional constraint networks). Remarkably, it not only works on constraint networks that are "first-order", but also on higher-order constraint networks. Unlike the "complete" consistency techniques, value propagation can

be pre-planned on the constraint network schemas, abstracting away from the concrete interpretations of constraints at the planning stage. The time-efficient planning algorithms are forward-oriented (the idea being: "produce all that you can until you get what you are striving for, and, in the end, find what was really needed"), and suitable for parallel implementation. Planning can be regarded as proof search in intuitionistic propositional logic.

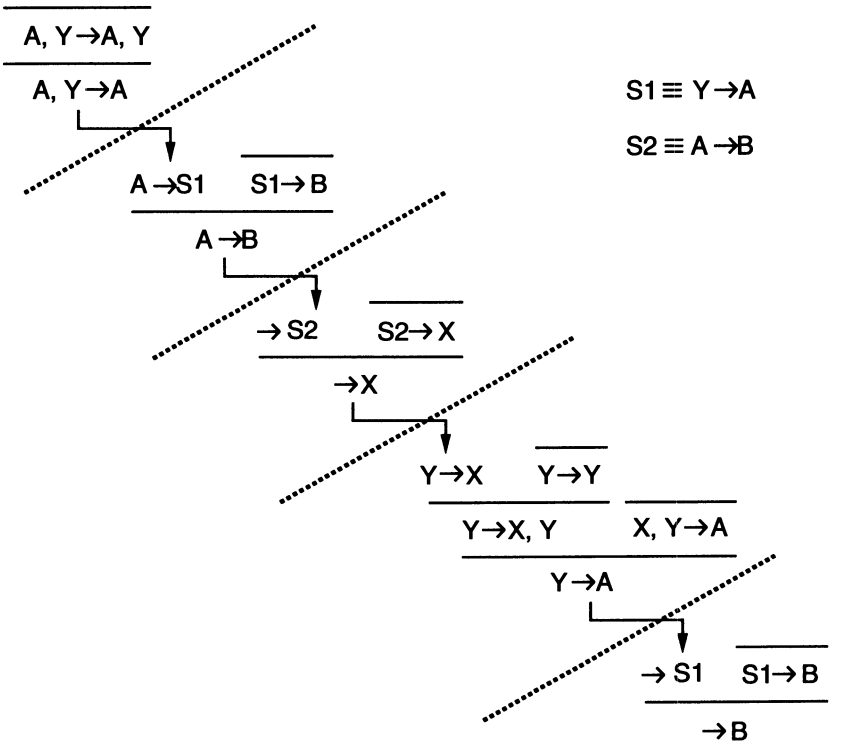

Fig. 2.13. A proof tree for Example "Kripke".

Acknowledgements

We are very thankful to Grigori Mints for his interest in the problems of propagation planning on computational models and his explanation of the propagation planning in terms of intuitionistic propositional logic and modal logic.

References

Borning, A. H. 1981. Programming language aspect of ThingLab: A constraint-oriented simulation laboratory. In ACM Transactions on Programming Languages and Systems **3**(4), 353–387

Dechter, R. Pearl, J. 1989. Tree clustering for constraint networks. Artificial Intelligence **38**, 353–366

Dechter, R. 1992. From local to global consistency. Artificial Intelligence **55**(1), 87–107

Dikovski, A. 1985. Solving algorithmic problems of synthesis of programs without loops in linear time. System Programming and Computer Software **3**, 38–49

Hyvönen, E. 1992. Constraint reasoning based on interval arithmetic: the tolerance propagation approach. Artificial Intelligence **581**–3, 71–112

Kanovich, M. 1991. Efficient program synthesis: semantics, logic, complexity. In Proc. Int'l Conf. on Theoretical Aspects of Computer Software, 615–632. LNCS 526 Berlin: Springer-Verlag.

Mackworth, A. 1992. The logic of constraint satisfaction. Artificial Intelligence **58**(1–3), 3–20

Mints, G., Tyugu, E. 1983. Justification of the structural synthesis of programs. Science of Computer Programming **2**(3), 215–240

Montanari, U., Rossi, F. 1991. Constraint relaxation may be perfect. Artificial Intelligence **48**, 143–170