Associons continued.

by

Edsger W.Dijkstra, W.H.J.Feijen and M.Rem

Up till now our original goal, that is high concurrency, has hardly been incorporated. We intend to introduce the possibility of high concurrency by introducing statements operating on sets. But rather than introducing variables, whose values are sets, we propose to characterize sets as "all the solutions of equations". The unknowns of these equations will be names: in our equations we shall indicate them with the reserved identifiers x0, x1, x2 etc.

For instance, let the store contain a large number of associons of the form (fatherof,"n1","n2"), where "n1" and "n2" stand for names of persons. To derive from this genealogy the brother-relation, we would write

(fatherof,x0,x1) and (fatherof,x0,x2) ⇒ (brotherof,x1,x2) .

Here the expression at the left-hand side of the implication sign is regarded as an equation in the names x0, x1 and x2, and it defines the set of all ordered triples "x0, x1, x2" such that, upon substitution, the two associons mentioned occur in store. For each solution for whivh (brotherof,x1,x2) is not in store it will be created; the amount of concurrency is up to the implementation. The statement terminates when for all solutions of the left-hand side the right-hand side holds as well.

The above program records any person with a father as a brother of himself. If that is not the intention, the situation could be adjusted by adding a next statement

(brotherof,x0,x0) ⇒ non (brotherof,x0,x0) .

Note: For the time being we impose upon ourselves the rule that the set of solutions must be fully determined by the equation at the left-hand side. As a result no x at the right-hand side may occur that does not occur at the left-hand side as well, and we, therefore, do not admit as a replacement for our last statement

() ⇒ non (brotherof,x0,x0);

according to our rule that only associons of the form mentioned at the right-hand

may be created or destroyed

(brotherof,x0,x0) ⇒ non ()

is equally inacceptable. (End of note.)

For all unknowns that occur only once in a statement --i.e. once at the left-hand side and not at the right-hand side-- it seems a bit superfluous to give that unknown a specific name. For instance, if we want to create associons indicating who is a father, we could of course do it with

(fatherof,x0,x1) ⇒ (isfather,x0)

but in the solution set, as defined by the left-hand side, a father with five sons occurs five times, while we know that we only need him once. We propose the notation

(fatherof,x0,?) ⇒ (isfather,x0) .

The question-mark has the function of the existential quantifier: (fatherof,n, holds if and only if there exists at least one person of whom n has been recorded to be the father. Viewed in terms of "associon matching" refers to a don't care position.

We may also have it at the right-hand side, provided that the term is negated:

() ⇒ non (fatherof,?,?)

would destroy the whole genealogy.

We do allow the question-mark in a negated term at the left-hand side as well, e.g. (fatherof,?,x0) and non (fatherof,x0,?) ⇒ (nooffspring,x0) .

The term (fatherof,?,x0) requires the presence of an a ssocion telling that x0 is a son, the term non (fatherof,x0,?) requires the total absence of associons of three elements, the first two of which are fatherof and "x0" respectively.

To create the above set of associons "nooffspring" without a question-mark is very painful. It could be done by

(fatherof,x0,x1) ⇒ (nooffspring,x1);
(fatherof,x0,x1) ⇒ non (nooffspring,x0)

The first one makes all sons in the class nooffspring, the second removes from that class all fathers.

What would have failed to work is

(fatheroff,x1,x0) and non (fatherof,x0,x2) ⇒ (nooffspring,x0) .

The term non (fatherof,x0,x2) is no restriction at all, because there are nearly infinitely many values of x2, such that the associon (fatherof,x0,x2) is absent. We, therefore, insist upon the rule that

> a negative trem at the left-hand side may contain no x that does not occur in a positive term of that left-hand side as well.

Note. The failing statement above cannot be corrected by adding positive terms containing x2 at the left-hand side. (End of note.)

* * *

Let for any node name n, n1, n2,

(r0,n) ⟺ "n is the name of a node of a given directed graph"
(r1,n1,n2) ⟺ "the directed edge from n1 to n2 belongs to the given directed graph."

Let it be requested to generate all associons:

(r2,n1,n2) ⟺ "from n1 the node n2 can be reached."

Program 1.

(r0,x0) ⇒ (r2,x0,x0);
(r2,x0,x1) and (r1,x1,x2) ⇒ (r2,x0,x2)

The first line expresses thst each node can be reached from itself; the second line expresses the rest. Note that, as solutions of the left-hand side are found and new associons are created, new solutions to the left-hand side are possible. The statem stops when the transitive closure has been formed.

Program 2. (r0,x0) ⇒ (r2,x0,x0);
(r1,x0,x1) ⇒ (r2,x0,x1);
(r2,x0,x1) and (r2,x1,x2) ⇒ (r2,x0,x2)

In program 2 the possible concurrency in the last statement may grow more rapidly than in program 1. (With N nodes in a directed ring, program 1 is linear in N, program 2 in logN.)

* * *

When no three points are on the same line, let

(r0,n) <==> "n is the name of a point in the plane".

(r1,n1,n2,n3) <==> "n1,n2,n3 is a clock-wise triangle."

Requested (r2,n1,n2) <==> "n1,n2 is an edge of the clock-wise convex hull."

Program

(r1,x0,x1,?) and non (r1,x1,x0,?) ⇒ (r2,x0,x1) .

* * *

Let

(r0,n) <==> "n is the name of a node".

We would like to select an arbitrary node, i.e. we should like to create exactly one associon

(r1,n) <==> "n is the chosen node".

We cannot write

(r0,x0) ⇒ (r1,x0)

because that would select all nodes. We could try the following one

(r0,x0) and non (r1,?) ⇒ (r1,x0) . (1)

This is OK under the assumption that the implementation selects only one solution of the left-hand side at a time: it would select e.g. x0 = n1, then create (r1,n1) and then it would stop. However, we would not like to impose this non-concurrency as a general requirement, i.e. we would like this statement also to allow that an unknown number of nodes is selected, $n_1$ through $n_k$, and that for all of them (r1,$n_i$) is created. In short: our statement selects an arbitrary non-empty subset of the set of nodes. (If the given set is empty, the selected set is empty as well.) Well, (1) is a nice, non-deterministic tool, but it is still not a way for selecting exactly one!

Alternatively: if we cannot select one, can we remove all but one? With the aid of

(r1,n1,n2) <==> "n1 is n2"

we could try

(r0,x0) ⇒ (r1,x0);

(r0,x0) ⇒ (r2,x0,x0);

(r1,x0) and (r1,x1) and non (r2,x0,x1) ⇒ non (r1,x1)

but this works equally bad: if x0,x1 is a solution, so is x1,x0 and the last two associons of the type (r1,n) could disappear simultaneously.

A new tool seems indicated: we must be able to prescribe that we only need one solution. We shall do so by using instead of the x, the equally reserved identifier y and the selection of the unique node will be accomplished by

(r0,y0) and non (r1,?) ⇒ (r1,y0) .

The term non (r1,?) is obligatory, because

(r0,y0) ⇒ (r1,y0)

would still copy the whole set, be it one at a time: as long as there are solutions of the left-hand side that do not satisfy the right-hand side, the game proceeds as usual.

Let it be required to order the nodes in an arbitrary permutation, i.e.

(r3,n1,n2) <⇒> "n1 is n2's left-hand neighbour."

Program:

(r0,y0) and non (r3,?,?) ⇒ (r3,handle,y0);
(r3,?,y0) and non (r3,y0,?) and (r0,y1) and non (r3,?,y1) ⇒ (r3,y0,y1);
() ⇒ non (r3,handle,?)

The constant "handle" has been introduced such that

(r3,?,n) <⇒> "n has been placed in the sequence".

The left-hand side of the second statement is in words: y0 must occur in the sequence without right-hand neighbour and y1 must be a node not in the sequence. Frased that way, it is not too bad.

Here we have our first equation with two y's. It means: select from all solutio --as if they had been x'es!-- exactly one single pair. What about x'es and y's occurring in the same left-hand equation?

Let us have one x and one y in a left-hand side equation. The corresponding solution set could be either:

a) every possible value of x combined with one then possible value of y.
b) one value of y combined with all then possible values of x.

The choice we have is due to the fact that the universal and the existential quantifier do not commute.

We propose to distinguish between the two cases by the way in which we number our unknowns; and to indicate

case a) by x0 y1
case b) by y0 x1.

The set of solutions for an equation in

z0,z1, ... ,zn

where z may either be an x or an y, is given recursively as follows:

We start with one empty k-tuple for k = 0.

If zk = xk,

each k-tuple gives rise to as many k+1-tuples that we can form by extending it with a permissible value for xk; each k-tuple gives rise to at least one k+1-tuple;

If zk = yk,

each k - tuple gives rise to exactly one k+1-tuple that we can form by extending it with an arbitrary permissible value for yk.

The set of n+1-tuples is our set of solutions.

Note 1. The relative ordering of succesive x'es is irrelevant; so is the relative ordering of successive y's. (End of note 1)

Note 2. When in a statement all x'es are replaced by y's, its activity is equal to a possible activity of the original form, because the concurrency when x'es are present is not obligatory. They are therefore identical when the statement with the x'es is deterministic. (End of note 2)

Note 3. We don't expect much use of complicated alternation of x'es and y's in the ordered sequence of unknowns. (End of note 3)

* * *

Let for N different values of n

(r0,n) <=> "n is the name of a node".

Let it be requested to form N-1 associons

(r1,n1,n2) <=> "n1,n2 is a directed edge of an arbitrary rooted spanning tree."

Program:

(r0,y0) and non (r1,?,?) ⇒ (r1,handle,y0);
(r0,x0) and non (r1,?,x0) and (r1,?,y1) ⇒ (r1,y1,x0);
() ⇒ (r1,handle,?)

The middle statement has the property thet many new nodes may be attached simultaneously to the tree, but no one in more than one way: a merging of branches is excluded. Different new nodes may be attached simultaneously to the same old one.

* * *

(To be continued.)