Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

MATHEMATICAL CENTRE TRACTS 76

M. REM

ASSOCIONS AND THE CLOSURE STATEMENT

MATHEMATISCH CENTRUM AMSTERDAM 1976

AMS(MOS) subject classification scheme (1970): 68A30, 68A05

ACM-Computing Reviews-categories 4.22, 5.24

ISBN 90 6196 135 1

CONTENTS

ACKNOWLEDGEMENTS			*vii*
CHAPTER	1.	Prologue	1
	2.	Associons	5
	3.	Characterization of states	7
	4.	An appreciation of the closure statement	15
	5.	Closure of a set of associons	23
	6.	Formal definition of the closure statement	35
	7.	Some small examples	43
	8.	An appreciation of the repetitive construct	55
	9.	Formal definition of the repetitive construct	67
	10.	Some examples	77
	11.	Dynamically created names	86
	12.	Recording the cliques of an undirected graph	94
	13.	On what we have rejected	100
	14.	Epilogue	107
BIBLIOGRAPHY			110
INDEX			112

ACKNOWLEDGEMENTS

The author is very much indebted to Prof.dr. E.W. Dijkstra for bringing the subject of this monograph to his attention and for his many helpful suggestions.

I am also indebted to Prof.dr. F.E.J. Kruseman Aretz, who very accurately read the manuscript, and whose suggestions improved the presentation to a large extent.

I would like to thank the participants of the "tuesday sessions" at Eindhoven University of Technology, drs. R.W. Bulterman, ir. W.H.J. Feijen, ir. A.J. Martin and drs. E.F.M. Steffens, for their contributions in discussions on earlier drafts of the manuscript.

I owe thanks to Prof.dr. S.T.M. Ackermans, drs. H.J.M. Goeman, and drs. C.S. Scholten, who have corrected a number of errors and obscurities in the text.

I thank the Mathematical Centre for the opportunity to publish this monograph in their series Mathematical Centre Tracts and all those at the Mathematical Centre who have contributed to its technical realization.

CHAPTER 1

PROLOGUE

Programming languages enable us to abstract from the machines we are using. It is the purpose of the implementation to map the programming language constructs on the machine instructions. This implementation should be a truthful one, i.e. it should not hide properties, the knowledge of which is indispensable for the construction of correct and efficient programs. A programming language should be such that it allows for a truthful implementation.

Present-day programming languages reflect present-day machine technology. New techniques --associative addressing, large scale integration (LSI)-- are being developed. These new techniques may very well allow for a truthful implementation of radically different programming languages.

In stores that are realized with LSI-techniques the information is (usually) kept in essentially active components. Such a store virtually consists of a large number of little machines, that do nothing else but remembering some value and on command reproducing it or replacing it by some other value. It may well be that someday these little machines will be able to do "more intelligent" work than the mere simulation of a core store. It may well be that the major part of the logical manipulations will take place distributed all through the "store", thus realizing a very high degree of concurrency.

Questions that then arise are: "How can we, when it is desired to program for such a machine, exploit this potential ultraconcurrency?", "Can we think of useful language constructs whose execution may involve such a distributed activity?" and "Can we do this in such a way that the implied programming task remains intellectually manageable?". It is to such questions that this monograph is addressed.

We would like to stress that it is not our intention to design a machine. Our primary concern is the manageability from the programmer's point of view. If so desired, one can, of course, interpret the semantics of our programming language as the functional specifications of such a machine.

We shall write down programs under control of which a highly concurrent activity is possible, but not obligatory. We shall, furthermore, arrange our programs in such a fashion that, in spite of the high potential concurrency, most of the thus far developed techniques for the programming of sequential processes remain applicable. (We maintain the semicolons, but allow more powerful statements in between.)

The basic idea of the research (essentially: the massaging of a set of n-tuples) is due to E.W. Dijkstra. A preliminary design, in the realization of which W.H.J. Feijen and the author participated as well, has been reported in [7] and [8].

* *
*

Most of the syntax of our programming language will be given in *BNF* [13]. We have extended BNF with the convention that the braces "{...}" should be read as "zero or more instances of the enclosed". E.g., the production rule

<statement list> ::= <statement> {;<statement>}

defines <statement list> to denote a sequence of one or more instances of <statement>, separated by semicolons.

There is little sense in introducing a programming language, if its semantics cannot be formalized. There are several reasons for this. A formal definition of the semantics is indispensable if one wants to prove the correctness of a program. But even if one does not intend to prove the correctness of one's programs explicitly, formally defined semantics make it possible to understand and appreciate programs without being forced to think in terms of specific implementations. We, furthermore, wish to design our programs in such a fashion that they are a priori known to meet the requirements of their correctness proofs. By doing so correctness concerns can, by their guiding role in program design, contribute to the alleviation of the programming task.

For the definition of the semantics of our programming language we shall use the concept of the *weakest pre-condition* [6]:

> If S denotes a mechanism (statement list), and R some condition on the state of the system, then "wp(S,R)" denotes the weakest pre-condition for the initial state of the system, such that activation of S is guaranteed to lead to a properly terminating activity, leaving the system in a final state satisfying the post-condition R .

(A *condition on the state* --or simply: a *condition*-- is a boolean function defined on all states.) As in [6], we shall restrict ourselves to "wp's" that satisfy the following three properties for any statement list S and for all states.

PROPERTY 1.1. $wp(S,false) \equiv false$,

PROPERTY 1.2. For any two conditions P and Q

$$(wp(S,P) \wedge wp(S,Q)) \equiv wp(S,P \wedge Q) ,$$

PROPERTY 1.3. For any infinite sequence of conditions $B_0,B_1,B_2,\ldots,$ such that $B_i \Rightarrow B_{i+1}$ $(i \geq 0)$,

$$wp(S,(\exists i: i \geq 0: B_i)) \equiv (\exists i: i \geq 0: wp(S,B_i)) .$$

From the above we can derive the following two properties. For any two conditions P and Q :

PROPERTY 1.4. $P \Rightarrow Q$ implies $wp(S,P) \Rightarrow wp(S,Q)$,

PROPERTY 1.5. $(wp(S,P) \vee wp(S,Q)) \Rightarrow wp(S,P \vee Q)$.

Property 1.4 is proved with Property 1.2. Suppose $P \Rightarrow Q$. Then $P \equiv (P \wedge Q)$, and consequently

$$wp(S,P) \equiv wp(S,P \wedge Q) .$$

According to Property 1.2

$$wp(S,P) \equiv (wp(S,P) \wedge wp(S,Q)) ,$$

from which $wp(S,P) \Rightarrow wp(S,Q)$ follows.

Property 1.5 is proved using Property 1.4:

$P \Rightarrow (P \vee Q)$ implies $wp(S,P) \Rightarrow wp(S,P \vee Q)$ (1)

$Q \Rightarrow (P \vee Q)$ implies $wp(S,Q) \Rightarrow wp(S,P \vee Q)$ (2)

From (1) and (2) it follows that $(wp(S,P) \vee wp(S,Q)) \Rightarrow wp(S,P \vee Q)$.

A mechanism S is said to be *deterministic* if and only if for any two conditions P and Q and for all states

$$wp(S,P \vee Q) \Rightarrow (wp(S,P) \vee wp(S,Q)) .$$

We take the position that we know the semantics of a mechanism S sufficiently well if we know its *predicate transformer*, i.e. if we know how to derive wp(S,R) for any post-condition R .

EXAMPLE 1.1. The assignment statement is the basic statement of most programming languages. Its semantics are given by

$$wp("x:= E",R) \equiv R_E^x ,$$

in which R_E^x denotes a copy of the predicate defining the post-condition R in which each occurence of the variable "x" is replaced by the expression "(E)". This definition is known as the *Axiom of Assignment.*
(End of example.)

EXAMPLE 1.2. With "skip" denoting the empty statement, and with "abort" denoting the statement that cannot terminate properly, we have, for all conditions P ,

$$wp("skip",P) \equiv P ,$$
$$wp("abort",P) \equiv false .$$

(End of example.)

In accordance with our earlier remark that we maintain the semicolons in our programming language, we define the semantics on the *semicolon* as in [6]:

> If S1 and S2 denote arbitrary statement lists, and R some post-condition, then
>
> $$wp("S1;S2",R) \equiv wp(S1,wp(S2,R)) .$$

CHAPTER 2

ASSOCIONS

This chapter is devoted to the recording of states of computations in our "active" store. It is our intention to realize this recording in such a way that at each moment all of the storage contents (so to speak) are involved in the computational process.

To achieve this high degree of concurrency by asking the programmer to synchronize explicitly the co-operation between a huge number of possibly all different concurrent sequential processes, seems a blind alley in the sense that the implied programming task will quickly exceed our abilities. It seems more attractive to look for a simple and systematic instruction repertoire, such that each instruction can interfere in a homogeneous fashion with the total contents of the store.

In conventional stores the components record values of variables. The represented state is changed by altering the value recorded in one --explicitly addressed-- component. In our active store we shall not address the components to be activated in the execution of a state change explicitly. We wish to achieve the above stated homogeneous interference by broadcasting commands through the store, telling that all components of which the contents satisfy a certain condition, should do something. The storage cells are anonymous, they are characterized by their contents only, i.e. we are assuming an associative store.

Having abolished addresses, we have to introduce *names*. The storage cells record *relations* between these names. We assume the machine to be able to test the equality of two names. If all entities to be referred to are identified by mutually distinct names, then any relation between some of these entities can be represented by a relation between their names. If, e.g., x and y stand for names of persons, we could have relations like

fatherof(x,y) meaning " x is the father of y ", and
olderthan(x,y) meaning " x is older than y " .

We then know, for instance, that

fatherof(x,y) ⇒ olderthan(x,y) .

Instead of representing all sorts of relations --such as "fatherof" or "olderthan"-- we choose the more general technique of considering different relations as named entities as well --e.g. named by "fatherof" and "olderthan", respectively--, leaving us with a single *universal relation* --which, therefore, can remain anonymous-- and represent

(fatherof,x,y) and
(olderthan,x,y) .

The knowledge that

(fatherof,x,y) ⇒ (olderthan,x,y)

could be represented by

(implies, fatherof, olderthan) .

Note that what from one point of view was regarded as "the name of a relation", from another point of view can be regarded as "an argument".

Such an ordered n-tuple of names is called an *associon*. We consider the contents of the store to be an unordered set of (different) associons. The presence of an associon in store will be interpreted as the truth of the universal relation applied to the entities denoted by the members of the n-tuple.

CHAPTER 3

CHARACTERIZATION OF STATES

In Chapter 1 we have said that a condition P is a boolean function of the state. If P is applied to the state X --notation "P|X"--, it yields either the value true or the value false . When no confusion seems possible, we may forget to mention the state and write "P" instead of "P|X". This was, as a matter of fact, done in Chapter 1. We shall be explicit in cases where precision is necessary.

When programming for conventional machines, the state of a computation is uniquely defined by a *state vector*, the components of which comprise the values of the individual program variables. The connection between conditions on the state and these state vectors is that names of variables (identifying components of the state vector) may occur in a condition on the state. The predicate P|X is then satisfied if and only if the condition P yields the value true if in P all names of variables are replaced by their value in the state vector (representing the state) X .

When programming for associons, the connection between states and conditions on the state is slightly different. The associons that are present in the store represent (instances of) relations between names. The evaluation of a computation is viewed as the creation of new associons, recording relations that are implied by already existing relations. One relation is fairly universal, it is the truth of "true", this will be recorded by the irrevocable presence in store of the *empty associon* "()". The state of a computation is then uniquely defined by an unordered set of associons, viz. the set of all present associons. If $(a_0,\ldots,a_{i-1})$ --i ≥ 0-- is an associon, then $[a_0,\ldots,a_{i-1}]$ is its corresponding *presence condition*. Presence conditions may (and usually will) occur in conditions on the state. If U denotes a set of associons (e.g. the state of a computation), then the predicate P|U is satisfied if and only if the condition P yields the value true if in P all presence conditions "$[a_0,\ldots,a_{i-1}]$" are replaced by "$(a_0,\ldots,a_{i-1}) \in U$".

REMARK 3.1. From the way in which the predicate P|U is defined, it follows that

1) $\mathrm{true}|U \equiv \mathrm{true}$,
2) $\mathrm{false}|U \equiv \mathrm{false}$,
3) $(P \wedge Q)|U \equiv (P|U \wedge Q|U)$,
4) $(P \vee Q)|U \equiv (P|U \vee Q|U)$,
5) $(\neg P)|U \equiv \neg(P|U)$.

(End of remark.)

Not every name in a presence condition has to be specified. Those names that we do not wish to specify should be replaced by a *question-mark* "?":

If L denotes zero or more names, each followed by a comma, and if M denotes zero or more names and question-marks, each preceded by a comma, then

$[L\ ?\ M]$ denotes $(\exists a\colon [L\ a\ M])$.

NOTE. Above and in the sequel all quantified variables of which the range is not specified, are assumed to be quantified over the (in principle infinite) set of names. (In some formulae --where no confusion seems possible-- over a Cartesian product of the set of names.)
(End of note.)

EXAMPLE 3.1. Let U denote the state $\{(),(a,b,c),(a,b,d),(a,c,c)\}$. Then the following propositions are satisfied.

$[]|U$,
$[?,b,c]|U$,
$(\forall x\colon [a,c,x] \Rightarrow [a,b,x])|U$.

(End of example.)

EXAMPLE 3.2. For any state

$[] \equiv \mathrm{true}$,
$\neg[] \equiv \mathrm{false}$.

(End of example.)

The state of a computation is (represented by) an unordered set of associons, containing the empty associon. The most basic functions on unordered sets seem to be the membership test and the cardinality. Above the member-

ship test has been exploited as a "building block" for conditions on the state. To characterize states by the number of associons present, on the other hand, seems a characterization that bears little fruits, as it tells us nothing about which associons are present. More is to be expected from the number of associons present of a certain type. We shall use *equations* to characterize types of associons. An equation is a kind of condition on the state that may contain *unknowns*. As equations will play a role in the "closure statement" --to be introduced later--, we give the definition of their syntax in BNF.

```
<equation> ::= <unknowns> <term>{∨<term>}
<unknowns> ::= <unknown>{,<unknown>}: | <empty>
<term> ::= <factor>{∧<factor>}
<factor> ::= <primary> | ¬<primary>
<primary> ::= <presence condition> | <nu> = <nu> |
              <nu> ≠ <nu> | (<term>{∨<term>})
<presence condition> ::= [] | [<nuq>{,<nuq>}]
<nu> ::= <name> | <unknown>
<nuq> ::= <nu> | ?
<name> ::= <identifier>
<unknown> ::= <identifier>
```

The logical operators "∧","∨", and "¬" have their usual meaning. We can rewrite an equation into a disjunction of terms in which no parentheses "(" and ")" occur anymore. An occurrence of a presence condition in an equation is called *negative* if, after such a rewrite, it is preceded by "¬", and it is called *positive* otherwise.

An equation E will in general contain unknowns. By substituting names for these unknowns, we obtain a condition on the state, that, for any given state, will either be satisfied or not. The set of all substitution instances for which the resulting condition is satisfied by U , is called the *solution set* of E in U , notation "Z(E,U)":

If E denotes the equation

$$u_0,\ldots,u_{i-1}: D(u_0,\ldots,u_{i-1})$$

($i \geq 0$), and U some set of associons, then "Z(E,U)" denotes the set of all i-tuples of names $a_0,\ldots,a_{i-1}$, such that $D(a_0,\ldots,a_{i-1})|U$.

In order to guarantee that for finite sets U, the set $Z(E,U)$ will be finite as well --to be proven in Theorem 3.1.--, we have to be restrictive as to the way in which unknowns may occur in an equation:

> After the elimination of parentheses, we may only have terms in which for each unknown of the equation there exists at least one positive presence condition in which the unknown occurs.

(We, for instance, do not allow the equation $x\colon [a,x] \vee \neg[b,x]$.)

EXAMPLE 3.3. Let U denote the set $\{(),(r,a,b),(r,b,a),(s,a)\}$, V the set $\{(),(r,a,b),(r,b,a),(s,b)\}$, and let E denote the equation

$$x,y\colon [r,x,y] \wedge \neg[s,x] .$$

Then
$$Z(E,U) = \{(b,a)\} ,$$
$$Z(E,V) = \{(a,b)\} ,$$
$$Z(E,U \cap V) = \{(a,b),(b,a)\} ,$$
$$Z(E,U \cup V) = \emptyset .$$

(End of example.)

EXAMPLE 3.4. Let U denote the set $\{(),(a,b,c)\}$, V the set $\{(),(a,c,c)\}$, and let E denote the equation

$$x\colon [a,?,x] .$$

Then
$$Z(E,U) = \{(c)\} ,$$
$$Z(E,V) = \{(c)\} ,$$
$$Z(E,U \cap V) = \emptyset ,$$
$$Z(E,U \cup V) = \{(c)\} .$$

(End of example.)

Note that a decrease of the number of associons (by taking the intersection) effects an increase of the number of solutions in Example 3.3, and a decrease of the number of solutions in Example 3.4.

THEOREM 3.1. For any equation E, and for any finite set of associons U, $Z(E,U)$ is finite.

PROOF. As a consequence of the above restriction on occurrences of unknowns in equations, only names that are members of associons in U can occur in solutions of equations. As U is finite, the number of names in associons in U is finite as well, say k . If E has i unknowns, then the cardinality of Z(E,U) does not exceed k^i .
(End of proof.)

The solution set Z(E,U) tells us something about the set (the state) U . In particular the cardinality of the solution set is a concept that will be used for characterizing states. We introduce a special notation for it:

If U denotes the state of a computation, and E some equation, then "(N E)|U" denotes the cardinality of Z(E,U) .

EXAMPLE 3.5. Let U denote the state {(),(r,a,b),(r,b,c),(r,c,d)} . Then

(Nx,y: [r,x,y] ∧ [r,y,?])|U = 2 ,
(N [r,a,b])|U = 1 ,
(N [r,b,a])|U = 0 ,
(N [r,?,?])|U = 1 ,
(Nx,y: [r,x,y])|U = 3 .

(End of example.)

(Usually we shall not mention the state explicitly, and write "(N E)" instead of "(N E)|U".)

REMARK 3.2. (N E) is a nonnegative function of the state. As such it will often be used as the variant function in termination proofs.
(End of remark.)

The test whether the solution set Z(E,U) is empty or not --i.e. whether (N E) = 0 -- can yield the value true or the value false , depending on the state U . In the future we shall use this test like the "boolean expression" in classical programming.

* *
*

In the sequel we shall use the terms *set difference* and *symmetric set difference*. If U and V denote sets, then the set difference of U and V, notation "$U \setminus V$", is the set $\{x: x \in U \wedge x \notin V\}$. (With "$\{x: P(x)\}$" we denote the set of all x such that $P(x)$.) The symmetric set difference of U and V, notation "$U \div V$", is the set $(U \setminus V) \cup (V \setminus U)$.

Another concept that will be used is the *match* of an associon and an equation. A presence condition in an equation may contain unknowns. Let a presence condition C contain the unknowns $u_0,\ldots,u_{i-1}$ $(i \geq 0)$. An associon A *fits* the presence condition C if and only if

$$Z("u_0,\ldots,u_{i-1}: C(u_0,\ldots,u_{i-1})", \{A\}) \neq \emptyset .$$

An associon *matches* an equation if and only if it fits at least one presence condition in the equation. The match is called *negative* if the associon fits a negative presence condition, and *positive* otherwise. (This is an asymmetric definition. Negative presence conditions can be viewed as "prohibitive regulations" on the presence of certain associons. We wish to characterize those associons that can --for a given equation-- possibly violate such a "prohibitive regulation".)

EXAMPLE 3.6. Let E denote the equation

$$x,y: [a,x,y] \wedge [b,x,y] \wedge \neg[b,y,?] .$$

The associon (a,b,c) matches E positively. The associon (b,a,c) matches E negatively.
(End of example.)

From Example 3.3 (p. 10) we know that $Z(E,U)$ is not monotonic in U, i.e. it is in general not true that

$$U \subset V \text{ implies } Z(E,U) \subset Z(E,V) .$$

The following property, however, does hold.

PROPERTY 3.1. If E denotes an arbitrary equation, and U and V denote sets of associons, such that $V \setminus U$ does not contain associons that negatively match E, then

$$U \subset V \text{ implies } Z(E,U) \subset Z(E,V) .$$

PROOF. If $Z(E,U)$ is empty, then the assertion of the theorem is trivially satisfied. Otherwise, let a be an arbitrary element of $Z(E,U)$.

Writing E without parentheses "(" and ")", it must contain a term T such that $a \in Z(T,U)$. We prove $a \in Z(T,V)$, and hence, $a \in Z(E,V)$.

Let v denote the list of unknowns of E , and let T contain i ($i \geq 0$) question-marks:

$$T(v,\underbrace{?,?,\ldots,?}_{i}) .$$

Let $b_0,\ldots,b_{i-1}$ be such that $T(a,b_0,\ldots,b_{i-1})|U$. (From $a \in Z(T,U)$ we know that such an i-tuple $b_0,\ldots,b_{i-1}$ must exist.)
For any positive presence condition

$$Cp(v,\underbrace{?,?,\ldots,?}_{i})$$

in T , we know from $Cp(a,b_0,\ldots,b_{i-1})|U$ and $U \subset V$ that

$$Cp(a,b_0,\ldots,b_{i-1})|V . \tag{1}$$

For any negative presence condition $Cn(v,?,?,\ldots,?)$ in T , we know, as it is not fitted by associons in $V \setminus U$,

$$\neg Cn(a,b_0,\ldots,b_{i-1})|(V \setminus U) . \tag{2}$$

We furthermore know

$$\neg Cn(a,b_0,\ldots,b_{i-1})|U , \tag{3}$$

and from (2) and (3)

$$\neg Cn(a,b_0,\ldots,b_{i-1})|V . \tag{4}$$

From (1) and (4) we conclude $T(a,b_0,\ldots,b_{i-1})|V$. Hence $T(a,?,?,\ldots,?)|V$, or $a \in Z(T,V)$.
(End of proof.)

We can now prove the following theorem, that will be used in Chapter 5.

THEOREM 3.2. If E denotes an arbitrary equation, and U and V denote sets of associons, such that $U \div V$ does not contain associons that negatively match E , then

$$Z(E,U \cap V) \subset (Z(E,U) \cap Z(E,V)) .$$

PROOF. Applying Property 3.1 (p. 12), we get

$$Z(E,U \cap V) \subset Z(E,U) \text{ , and}$$
$$Z(E,U \cap V) \subset Z(E,V) \text{ .}$$

The combination of these two yields the desired result.
(End of proof.)

REMARK 3.3. From Example 3.4 (p. 10) we know that the above theorem with the "inclusion" replaced by an "equality" does not hold. (This is a consequence of the occurrence of the question-mark in the equation.)
(End of remark.)

CHAPTER 4

AN APPRECIATION OF THE CLOSURE STATEMENT

We can distinguish two ways of appreciating programs. One way is by regarding a program as "executable code", i.e. as instructions that control the way in which, upon execution of the program, the computation proceeds through its states. The other appreciation is that a program can be viewed as a "predicate transformer", or to be more precise: as the one argument of the function "wp" that, with a predicate on the state as the other argument, yields as its value another predicate on the state. The link between these two conceptions is that the execution of a program S is guaranteed to terminate in a state satisfying the predicate P if and only if it is initiated in a state satisfying the predicate wp(S,P) . In the first --"mechanistic" or "operational"-- interpretation we can talk about implementations of programming languages, and about the efficiency of a program for a given implementation. In the second --"formal"-- interpretation time considerations, and hence efficiency considerations, do not enter the picture.

In this chapter we shall appeal to a mechanistic appreciation of the closure statement. In the two subsequent chapters we shall introduce the closure statement formally. Some of the assertions that are made plausible in this chapter, will be proved there.

In traditional programming the state of a computation is identified by a point in the state space --the earlier mentioned "state vector"--, the state space being the Cartesian product of the value sets of all variables. The basic statement is the assignment statement. It moves the point identifying the current state, parallel to one of the axes of the state space.

When programming with associons the state of a computation is identified by the set of associons present. These associons represent instances of relations between names. We shall introduce one basic statement that can change the set of associons present, viz. the *closure statement*.

From the relations between names that are represented by the associons present we can conclude new relations between these names. We can, for instance, from the knowledge that x is the name of an integer, and that x

is not the name of an even integer, conclude that x is the name of an odd integer.

Suppose that the above knowledge is represented in associons of the *formats* (int,?) and (even,?) , i.e. suppose that for all x

[int,x] ≡ " x is the name of an integer", and
[even,x] ≡ " x is the name of an even integer" ,

and suppose that we wish to represent the conclusion in associons (odd,?) --the target associons of the computation--, such that for all x

[odd,x] ≡ " x is the name of an odd integer" . (1)

Then, in order to establish the truth of (1), which is equivalent to

[odd,x] ≡ ([int,x] ∧ ¬[even,x]) , (2)

we would like to create for all solutions x of the equation

x: [int,x] ∧ ¬[even,x]

the associons (odd,x) . The closure statement that accomplishes this creation will be denoted by

x: [int,x] ∧ ¬[even,x] :⇒ (odd,x) .

(The dotted arrow ":⇒ " is pronounced as "creates".) This statement establishes the truth of the implication

(∀x: ([int,x] ∧ ¬[even,x]) ⇒ [odd,x]) .

(This is not really an implication, but rather a conjunction of implications. We shall apply this abuse of language more often.)

If initially the inverse implication, i.e.

(∀x: [odd,x] ⇒ ([int,x] ∧ ¬[even,x])) , (3)

was satisfied, e.g. because ¬[odd,?] , then (3) should still hold, and we have --by enlarging the set of associons present with target associons of the format (odd,?) -- established a state satisfying (2).

We propose the following syntax for the closure statement.

<closure statement> ::= <left-hand side> :⇒
<target associon format set>
<left-hand side> ::= <equation>
<target associon format set> ::= <target associon format>
{,<target associon format>}
<target associon format> ::= (<nu>{,<nu>})

We shall apply the notational convention that, if A denotes an associon, then $\tilde{A}$ denotes the corresponding presence condition. (The tilde "~" replaces the parentheses by brackets: e.g. if A denotes the associon "(a,b)", then $\tilde{A}$ denotes the condition "[a,b]".) By substituting names for the unknowns in the target associon format set $T(x)$ of a closure statement

$$x\colon E(x) :\Rightarrow T(x)$$

we obtain a set of associons, say $T(a)$. We shall also use the tilde on such sets of associons. $\tilde{T}(a)$ is a condition; for any set U of associons the predicate $\tilde{T}(a)|U$ is satisfied if and only if $T(a) \subset U$. (For instance, if T denotes the set $\{(a,b),(c,d),(e,f)\}$, then $\tilde{T}$ denotes the condition $[a,b] \wedge [c,d] \wedge [e,f]$.)

We then immediately have the following property.

PROPERTY 4.1. For all sets U and V of associons

$$((\tilde{T}|U) \wedge (\tilde{T}|V)) \equiv \tilde{T}|(U \cap V) .$$

Target associons of a closure statement are associons that can be obtained by substituting names for the unknowns in a target associon format.

The intended effect of the execution of the closure statement $x\colon E(x) :\Rightarrow T(x)$ will be that as few target associons as possible are created, in order to establish the truth of the implication

$$(\forall x\colon E(x) \Rightarrow \tilde{T}(x)) . \qquad (4)$$

EXAMPLE 4.1. The execution of the closure statement

S $\quad x\colon [v,x] :\Rightarrow (w,x)$

can cause target associons of the format $(w,?)$ to be created. Which associons $(w,?)$ are created depends on the solution set of the equation $x\colon [v,x]$ in the state in which S is executed.

Let that state be {(),(v,a),(v,b),(v,c),(w,a)} . Then

(∀x: [w,x] ⇒ [v,x])

holds. The effect of the execution of S will then be that the associons (w,b) and (w,c) are created, causing

(∀x: [v,x] ⇒ [w,x]) ,

and consequently

(∀x: [v,x] ≡ [w,x])

to hold.
(End of example.)

From the knowledge that x is greater than y , and that y is greater than z , we may (if "greater than" is transitive) conclude that x is greater than z . The closure statement

S x,y,z: [greater,x,y] ∧ [greater,y,z] :⇒ (greater,x,z)

would record such a conclusion. It establishes the truth of the implication

(∀x,y,z: ([greater,x,y] ∧ [greater,y,z]) ⇒ [greater,x,z]) ,

which is equivalent to

(∀x,z: (∃y: [greater,x,y] ∧ [greater,y,z]) ⇒ [greater,x,z]) .(5)

Due to the positive match of the target associons (greater,?,?) and the equation of S , (5) will (in general) not be established by creating for all initial solutions (x,y,z) of the left-hand side of S the absent associons (greater,x,z) . For these creations can enlarge the solution set of the left-hand side of S again, causing that new target associons have to be created, etc. We call such a closure statement cascading. Or, denoting by a "constant" a "member that is not an unknown or a question-mark",

> a closure statement is said to be *cascading* if and only if it contains a positive presence condition and a target associon format of the same length, and these do not have different constants at corresponding positions.

(Whether a closure statement is a cascading closure statement can hence be established statically.)

EXAMPLE 4.2. *Transitive closure.* Given a finite set W on which a binary relation S is defined, then the *transitive closure* of S , notation "$\overset{*}{S}$", is defined by

1) $(\forall x,y:\ x,y \in W:\ S(x,y) \Rightarrow \overset{*}{S}(x,y))$,
2) $(\forall x,z:\ x,z \in W:\ (\exists y:\ y \in W:\ \overset{*}{S}(x,y) \wedge \overset{*}{S}(y,z)) \Rightarrow \overset{*}{S}(x,z))$,
3) $\overset{*}{S}$ is only true for those arguments for which it is true on account of 1) and 2) .

Let a finite set W and a binary relation S on W be given by

$(\forall x:\ [w,x] \equiv$ " x is the name of an element of W ") $\wedge$
$(\forall x,y:\ [s,x,y] \equiv ([w,x] \wedge [w,y] \wedge S(x,y)))$.

REMARK 4.1. The expression "S(x,y)" in the above formula should be read as " S applied to the arguments of which the names are x and y ". We shall apply this abuse of language more often.
(End of remark.)

Let, furthermore, be given that $\neg[t,?,?]$ holds, and let it be requested to write a program that establishes the truth of the relation

R $\quad (\forall x,y:\ [t,x,y] \equiv ([w,x] \wedge [w,y] \wedge \overset{*}{S}(x,y)))$.

The definition of the transitive closure suggests the program

$x,y:\ [s,x,y] :\Rightarrow (t,x,y)$;
$x,y,z:\ [t,x,y] \wedge [t,y,z] :\Rightarrow (t,x,z)$.

It establishes the implication of the left-hand side of R by the right-hand side of R , under invariance of the initially holding implication in the other direction.
(End of example.)

As it is now, the execution of a closure statement cannot only enlarge the solution set of its left-hand side, but the creation of target associons could also reduce it. This is a very unattractive situation.

Let e.g. the state be $\{(),(r,a,b),(r,b,a)\}$, and let it be requested to establish the truth of

$$(\forall y:\ (\exists x:\ [r,x,y] \wedge \neg[s,x]) \Rightarrow [s,y]) \ , \qquad (6)$$

which is equivalent to

$$(\forall x,y:\ ([r,x,y] \wedge \neg[s,x]) \Rightarrow [s,y])\ .$$

Suppose we would allow the following statement to establish this implication.

$$x,y:\ [r,x,y] \wedge \neg[s,x] :\Rightarrow (s,y)\ .$$

The equation in the left-hand side has two solutions: "x = a, y = b" and "x = b, y = a". For both solutions the corresponding target associons (s,y) are absent. This does not necessarily imply that both (s,a) and (s,b) should be created, as (6) should be established by creating as few associons (s,?) as possible. As the creation of one of them already establishes (6), either (s,a) or (s,b) should be created, but not both. This phenomenon would severely complicate the concurrent creation of target associons, and would as such unsettle our whole design.

We, therefore, wish to avoid that the creation of target associons can reduce the solution set of the left-hand side. From Property 3.1 (p. 12) we know that, if the created target associons do not match the equation in the left-hand side negatively, then its solution set can never shrink.
Hence we forbid this negative match:

> If a closure statement contains a presence condition and a target associon format of the same length, and these do not have different constants at corresponding positions, then the presence condition must be a positive one.

(Whether a closure statement obeys this rule can be established statically.)

By prohibiting this negative match the closure statement becomes a deterministic construct. As a consequence its effect cannot depend on the amount of concurrency in the implementation of its activity. In order to establish the truth of (4) the implementation could, as long as

$$(\exists x:\ (\exists A:\ A \in T(x):\ E(x) \wedge \neg\tilde{A}(x)))\ ,$$

create such an associon A . How much of this is done concurrently, is up to the implementation.

We know that information destruction is essential for all nontrivial computing, as a computation would otherwise merely be a reversible transformation of the initial state. How is information destroyed if the closure statement is our only basic statement? It may sound contradictory --to the novice at least--, but the execution of a closure statement will in general

indeed cause information to be destroyed. The reason for this is that the mapping from initial state to final state is in general not one-to-one. If the execution of a closure statement S in state U0 transforms the state into U1 , then the execution of S in state U1 would also lead to the final state U1 .

Still, one might wish --in order to attach transient meanings to associons-- to destroy associons. We accomplish this by allowing as a statement a *block*, which is a statement list surrounded by the delimiters "loc" and "col". This statement list is the scope of all associons of the formats listed after the delimiter "loc":

<block> ::= loc <associon format> {,<associon format>}:
<statement list> col
<associon format> ::= ({<nq>,} <name> {,<nq>})
<nq> ::= <name> | ?

Upon "block entry" we have for all associons A of the specified format $\neg\tilde{A}$. Upon "block exit" all local associons (all associons of the specified format that have been created during the execution of the statement list) are destroyed. The logical need of local associons will not arise until the introduction of the repetitive construct.

REMARK 4.2. The closure statement can easily be generalized into a *concurrent* closure statement. Such a concurrent closure statement is a set of closure statements in which no target associon can negatively match any of the equations. The concurrent closure statement

$$\{x_0\colon E_0(x_0) :\Rightarrow T_0(x_0),\ldots,x_{n-1}\colon E_{n-1}(x_{n-1}) :\Rightarrow T_{n-1}(x_{n-1})\}$$

establishes --by creating target associons of $T_0,\ldots,T_{n-1}$-- the truth of

$$(\forall x_0\colon E_0(x_0) \Rightarrow \tilde{T}_0(x_0)) \wedge\ldots\wedge (\forall x_{n-1}\colon E_{n-1}(x_{n-1}) \Rightarrow \tilde{T}_{n-1}(x_{n-1})) .$$

In order to achieve this, the different members of the concurrent closure statement may be executed in any order --even concurrently-- and should be executed until all of them have finished, i.e. until for all i $(0 \leq i < n)$

$$(\forall x_i\colon E_i(x_i) \Rightarrow \tilde{T}_i(x_i))$$

holds.

Any closure statement can be implemented by a concurrent closure statement of which all members have a conjunction of two factors --not containing any parentheses "(" and ")"-- as a left-hand side. We shall illustrate this with an example.

An *out-tree* is a directed rooted tree, directed in such a way that each vertex is reachable from the root. (When dealing with graphs we mainly employ the terminology as defined in [9].) If an out-tree T is given by

(∀x: [v,x] ≡ " x is the name of a vertex of T ") ∧
(∀x,y: [s,x,y] ≡ ([v,x] ∧ [v,y] ∧ " T has an arc from x
to y)) ,

then the closure statement

x,y,z: [s,x,y] ∧ [s,y,z] ∧ (¬[s,?,x] ∨ [ev,x]) :⇒ (ev,z) (7)

would record all "even" vertices of T , i.e. all vertices for which the path from the root comprises an even number (zero excluded) of arcs. This statement could be implemented by the following concurrent closure statement, that we have written as a block.

```
loc (h,?,?):
    {x,y,z: [s,x,y] ∧ [s,y,z] :⇒ (h,x,z),
     x,z: [h,x,z] ∧ ¬[s,?,x] :⇒ (ev,z),
     x,z: [h,x,z] ∧ [ev,x] :⇒ (ev,z)}
col
```

(We could have coded statement (7) more elegantly in two or three statements. We have not done so, because we wished to illustrate the systematic translation.)

The fact that any closure statement can be written as a set of "simple" closure statements, should give us confidence in the implementability of arbitrarily complex closure statements. (It is the analogue of the phenomenon that any arithmetic expression can be written as a succession of binary operations.)
(End of remark.)

CHAPTER 5

CLOSURE OF A SET OF ASSOCIONS

In Chapter 4 we have described the closure statement in a mechanistic fashion. In Chapter 6 we shall give the formal definition of the effect of the execution of a closure statement. In this chapter we lay the foundation for the formal definition by studying sets of associons and their relations. In particular shall we study the concept of a closure of a set of associons with respect to a closure statement. This treatment will not depend on the mechanistic appreciation of the closure statement we acquired in Chapter 4. From Chapter 4 we shall only use the knowledge which texts constitute legitimate closure statements.

We apply the notational convention that, if S denotes the closure statement

$$x\colon E(x) :\Rightarrow T(x) \ ,$$

then $\hat{S}$ denotes the condition

$$(\forall x\colon E(x) \Rightarrow \widetilde{T}(x)) \ .$$

(If, for instance, S denotes the statement $x\colon [v,x] :\Rightarrow (w,x)$, then $\hat{S}$ denotes the condition $(\forall x\colon [v,x] \Rightarrow [w,x])$.)

PROPERTY 5.1. If S denotes the closure statement $x\colon E(x) :\Rightarrow T(x)$, and U some set of associons, then

$$\hat{S}|U \equiv (\forall x\colon x \in Z(E,U)\colon T(x) \subset U) \ .$$

PROOF. By definition,

$$\hat{S}|U \equiv (\forall x\colon E(x) \Rightarrow \widetilde{T}(x))|U \ .$$

As in Remark 3.1 (p. 7), we distribute the U , yielding

$$\hat{S}|U \equiv (\forall x\colon (E(x)|U) \Rightarrow (\widetilde{T}(x)|U)) \ .$$

According to the definition of the solution set

$$E(x)|U \equiv x \in Z(E,U) \ ,$$

which, together with

$$\widetilde{T}(x)|U \equiv (T(x) \subset U)$$

(cf. p. 17), proves the property to hold.
(End of proof.)

THEOREM 5.1. If S denotes an arbitrary closure statement, and U and V denote sets of associons, such that $U \div V$ does not contain associons that negatively match the left-hand side of S, then

$$((\hat{S}|U) \wedge (\hat{S}|V)) \Rightarrow \hat{S}|(U \cap V) .$$

PROOF. Let S denote the closure statement $x\colon E(x) :\Rightarrow T(x)$. We assume

$$(\hat{S}|U) \wedge (\hat{S}|V) , \tag{1}$$

and we derive $\hat{S}|(U \cap V)$. From (1) and Property 5.1 (p. 23) we conclude

$$(\forall x\colon x \in Z(E,U)\colon T(x) \subset U) \wedge (\forall x\colon x \in Z(E,V)\colon T(x) \subset V) ,$$

which implies

$$(\forall x\colon x \in (Z(E,U) \cap Z(E,V))\colon (T(x) \subset U) \wedge (T(x) \subset V)) ,$$

or

$$(\forall x\colon x \in (Z(E,U) \cap Z(E,V))\colon T(x) \subset (U \cap V)) . \tag{2}$$

As $U \div V$ does not contain associons that negatively match E, we may apply Theorem 3.2 (p. 13), yielding

$$Z(E,U \cap V) \subset (Z(E,U) \cap Z(E,V)) . \tag{3}$$

From (2) and (3) we conclude

$$(\forall x\colon x \in Z(E,U \cap V)\colon T(x) \subset (U \cap V)) ,$$

or (apply Property 5.1 (p. 23))

$$\hat{S}|(U \cap V) .$$

(End of proof.)

The following lemma is a consequence of the above theorem.

LEMMA 5.1. If U denotes some finite set of associons, and S an arbitrary closure statement, and if $W1$ and $W2$ are sets W of associons satisfying

1) $U \subset W$,
2) $W \setminus U$ does not contain associons that negatively match the left-hand side of S,
3) $\hat{S}|W$,

then $W1 \cap W2$ is also a set W of associons satisfying these properties.

PROOF.

1) $(U \subset W1) \wedge (U \subset W2)$ implies $U \subset (W1 \cap W2)$.

2) $(W1 \cap W2) \setminus U = (W1 \setminus U) \cap (W2 \setminus U)$.

3) As

$$W1 \div W2 \subset (W1 \setminus U) \cup (W2 \setminus U) ,$$

the set $W1 \div W2$ does not contain associons that negatively match the left-hand side of S . This allows us to apply Theorem 5.1 (p. 24), yielding

$$(\hat{S}|W1 \wedge \hat{S}|W2) \Rightarrow \hat{S}|(W1 \cap W2) .$$

(End of proof.)

LEMMA 5.2. If U denotes some finite set of associons, and S an arbitrary closure statement, then there exists a finite set $W0$, satisfying properties 1), 2), and 3) of Lemma 5.1 (p. 24).

PROOF. Let V denote the set of all names occurring in associons of U or occurring (as constants) in the target associon formats of S . As U is finite, V will be finite as well.

$W0$ is defined as the union of U and the set of all target associons of S that can be obtained by substituting names of V for the unknowns in the target associon formats. Then $W0$ is finite and it satisfies property 1). As target associons do not negatively match the left-hand side, $W0$ also satisfies property 2).

We still have to prove $\hat{S}|W0$, or, with S denoting the closure statement $x\colon E(x) :\Rightarrow T(x)$, (apply Property 5.1 (p. 23))

$$(\forall x\colon x \in Z(E,W0)\colon T(x) \subset W0) .$$

If $x \in Z(E,W0)$, then --cf. the proof of Theorem 3.1 (p. 10)-- x contains only names occurring in associons of $W0$, i.e. names of V . But then $T(x) \subset W0$.

(End of proof.)

A consequence of the above two lemmata is that for any closure statement S there exists, for any finite set of associons U , a unique smallest set W of associons satisfying properties 1), 2), and 3) of Lemma 5.1, viz. their intersection. This unique smallest set is called the

closure of U *with respect to* S , notation "C(S,U)". We immediately have

THEOREM 5.2. If U denotes some finite set of associons, and S an arbitrary closure statement, then C(S,U) is finite.

PROPERTY 5.2. If U denotes some finite set of associons, S an arbitrary closure statement, and p some presence condition, then

$$p|U \Rightarrow p|C(S,U) .$$

PROOF. $U \subset C(S,U)$.
(End of proof.)

PROPERTY 5.3. If U denotes some finite set of associons, and S the closure statement x: E(x) :⇒ T(x) , then for all x

$$E(x)|U \Rightarrow E(x)|C(S,U) .$$

PROOF. $U \subset C(S,U)$. As $C(S,U) \setminus U$ does not contain associons that negatively match E , we may apply Property 3.1 (p. 12), yielding

$$Z(E,U) \subset Z(E,C(S,U)) .$$

According to the definition of the solution set, this is equivalent to the property to be proved.
(End of proof.)

PROPERTY 5.4. If U denotes some finite set of associons, and S an arbitrary closure statement, then

$$\hat{S}|C(S,U) .$$

PROOF. Consequence of the definition of C(S,U) .
(End of proof.)

PROPERTY 5.5. If S denotes some closure statement, and U and V denote sets of associons, such that $U \subset V$ and $V \setminus U$ does not contain associons that negatively match the left-hand side of S , then

$$C(S,U) \subset C(S,V) .$$

PROOF. We prove that C(S,V) satisfies properties 1), 2), and 3) of Lemma 5.1 (p. 24). This lemma then learns us that the set $C(S,U) \cap C(S,V)$ satisfies them too. With C(S,U) being the smallest such set, we then have

$$C(S,U) \subset C(S,V) \ .$$

1) $(U \subset V) \wedge (V \subset C(S,V))$ implies $U \subset C(S,V)$.

2) $$C(S,V) \setminus U = (C(S,V) \setminus V) \cup (V \setminus U) \ .$$

By definition, $C(S,V) \setminus V$ does not contain associons that negatively match the left-hand side of S . It is given that $V \setminus U$ does not contain them either. Therefore, $C(S,V) \setminus U$ does not contain associons that negatively match the left-hand side of S .

3) According to Property 5.4 (p. 26), $\hat{S}|C(S,V)$.

(End of proof.)

PROPERTY 5.6. If U denotes some finite set of associons, and S some closure statement, then

$$\hat{S}|U \Rightarrow C(S,U) = U \ .$$

PROOF. By definition, $U \subset C(S,U)$. $\hat{S}|U$ implies that U satisfies properties 1), 2), and 3) of Lemma 5.1 (p. 24). As $C(S,U)$ is the smallest such set, we have $U = C(S,U)$.

(End of proof.)

PROPERTY 5.7. If U denotes some finite set of associons, and S an arbitrary closure statement, then

$$C(S,C(S,U)) = C(S,U) \ .$$

PROOF. Consequence of Properties 5.4 (p. 26) and 5.6.

(End of proof.)

PROPERTY 5.8. If U denotes some finite set of associons, and S the closure statement $x\colon E(x) :\Rightarrow T(x)$, then

$$C(S,U) = U \cup \{A\colon (\exists x\colon x \in Z(E,C(S,U))\colon A \in T(x))\} \ .$$

PROOF. Let V denote the set

$$\{A\colon (\exists x\colon x \in Z(E,C(S,U))\colon A \in T(x))\} \ .$$

We first prove $C(S,U) \subset (U \cup V)$ by a reductio ad absurdum. Suppose there exists an associon A , such that $A \in C(S,U) \wedge A \notin U \wedge A \notin V$. Let W denote the set $C(S,U) \setminus \{A\}$. If we can show that W satisfies properties 1), 2), and 3) of Lemma 5.1 (p. 24), then we have found a smaller set than $C(S,U)$ satisfying these properties, and we have derived a contra-

diction.

1) From $U \subset C(S,U)$ follows $U \setminus \{A\} \subset C(S,U) \setminus \{A\}$, or $U \subset W$.

2) As $C(S,U) \setminus U$ does not contain associons that negatively match E, and $(W \setminus U) \subset (C(S,U) \setminus U)$, the set $W \setminus U$ does not contain them either.

3) From $A \notin V$ and the definition of V we conclude

$$(\forall x: x \in Z(E,C(S,U)): A \notin T(x)) . \tag{4}$$

From $\hat{S}|C(S,U)$ we may, according to Property 5.1 (p. 23), conclude

$$(\forall x: x \in Z(E,C(S,U)): T(x) \subset C(S,U)) . \tag{5}$$

From (4), (5), and the definition of W we conclude

$$(\forall x: x \in Z(E,C(S,U)): T(x) \subset W) . \tag{6}$$

The set $C(S,U) \setminus W$ contains the associon A only. As $A \in C(S,U) \setminus U$, A does not negatively match E. This allows us to apply Property 3.1 (p. 12) on W and $C(S,U)$, yielding

$$Z(E,W) \subset Z(E,C(S,U)) . \tag{7}$$

From (6) and (7) we conclude

$$(\forall x: x \in Z(E,W): T(x) \subset W) ,$$

or (apply Property 5.1 (p. 23)) $\hat{S}|W$.

Next we prove

$$(U \cup V) \subset C(S,U) .$$

As $U \subset C(S,U)$, we only have to prove

$$V \subset C(S,U) .$$

If V is empty, this is trivially satisfied. Otherwise, let A be an arbitrary element of V. We prove $A \in C(S,U)$. From the definition of V we conclude the existence of an $x \in Z(E,C(S,U))$, such that

$$A \in T(x) . \tag{8}$$

From $\hat{S}|C(S,U)$ and $x \in Z(E,C(S,U))$ we deduct, by applying Property 5.1 (p. 23),

$$T(x) \subset C(S,U) . \tag{9}$$

From (8) and (9) follows

$$A \in C(S,U) \ .$$

(End of proof.)

The above property shows that $C(S,U) \setminus U$ contains target associons only. The following property is a direct consequence.

PROPERTY 5.9. If U denotes some finite set of associons, p some presence condition, and S a closure statement of which no target associon fits p, then

$$p|C(S,U) \equiv p|U \ .$$

PROPERTY 5.10. If U denotes some finite set of associons, and S the noncascading closure statement $x\colon E(x) :\Rightarrow T(x)$, then

$$C(S,U) = U \cup \{A\colon (\exists x\colon x \in Z(E,U)\colon A \in T(x))\} \ .$$

PROOF. In a noncascading closure statement target associons do not match E. Hence, we have

$$Z(E,U) = Z(E,C(S,U)) \ .$$

The result now follows directly from Property 5.8 (p. 27).
(End of proof.)

The following property will turn out to be important in Chapter 6. It will then give rise to the invariance theorem for closure statements.

PROPERTY 5.11. Let S denote the closure statement $x\colon E(x) :\Rightarrow T(x)$, A an associon, and U some finite set of associons. If

$$A \in U \Rightarrow (\exists x\colon x \in Z(E,U)\colon A \in T(x)) \qquad (10)$$

then

$$A \in C(S,U) \Rightarrow (\exists x\colon x \in Z(E,C(S,U))\colon A \in T(x)) \ . \qquad (11)$$

PROOF. We assume (10) and derive (11). According to Property 5.3 (p. 26)

$$Z(E,U) \subset Z(E,C(S,U)) \ . \qquad (12)$$

From (10) and (12) we conclude

$$A \in U \Rightarrow (\exists x\colon x \in Z(E,C(S,U))\colon A \in T(x)) \ . \qquad (13)$$

According to Property 5.8 (p. 27)

$$A \in C(S,U) \Rightarrow ((A \in U) \vee (\exists x\colon x \in Z(E,C(S,U))\colon A \in T(x))) \ . \quad (14)$$

From (13) and (14) follows (11).
(End of proof.)

REMARK 5.1. For noncascading closure statements we have $Z(E,U) = Z(E,C(S,U))$. As $U \subset C(S,U)$, we may then conclude that (11) implies (10). For noncascading closure statements Property 5.11 (p. 29) can, consequently, be strengthened into: (10) if and only if (11).
(End of remark.)

Property 5.10 (p. 29) gives a constructive characterization of $C(S,U)$ for noncascading closure statements. We shall generalize this into a constructive characterization of $C(S,U)$ for arbitrary closure statements. The equivalence of this characterization and our earlier definition will be proved in Theorem 5.3 (p. 31).

We first define, for a finite set U of associons, and for an arbitrary closure statement $x\colon E(x) :\Rightarrow T(x)$, sets F_i of associons ($i \geq 0$) as follows.

$$F_0 = U \ ,$$

$$F_{i+1} = F_i \cup \{A\colon (\exists x\colon x \in Z(E,F_i)\colon A \in T(x))\} \ .$$

LEMMA 5.3. If U denotes some finite set of associons, and S the closure statement $x\colon E(x) :\Rightarrow T(x)$, then for all i ($i \geq 0$)

$$F_i \subset C(S,U) \ .$$

PROOF. We prove $F_i \subset C(S,U)$ by mathematical induction. Obviously, $F_0 \subset C(S,U)$. Suppose $F_k \subset C(S,U)$ ($k \geq 0$). We prove $F_{k+1} \subset C(S,U)$.

By definition,

$$F_{k+1} = F_k \cup \{A\colon (\exists x\colon x \in Z(E,F_k)\colon A \in T(x))\} \ . \quad (15)$$

From Property 5.3 (p. 26) we know

$$Z(E,F_k) \subset Z(E,C(S,F_k)) \ . \quad (16)$$

From (15) and (16) we conclude

$$F_{k+1} \subset (F_k \cup \{A\colon (\exists x\colon x \in Z(E,C(S,F_k))\colon A \in T(x))\}) \ ,$$

or (apply Property 5.8 (p. 27))

$$F_{k+1} \subset C(S,F_k) \ . \tag{17}$$

As $U \subset F_k$, we have

$$(C(S,U) \setminus F_k) \subset (C(S,U) \setminus U) \ .$$

$C(S,U) \setminus F_k$, consequently, does not contain associons that negatively match E . This allows us to apply Property 5.5 (p. 26), yielding

$$C(S,F_k) \subset C(S,C(S,U)) \ ,$$

or (apply Property 5.7 (p. 27))

$$C(S,F_k) \subset C(S,U) \ . \tag{18}$$

From (17) and (18) we conclude

$$F_{k+1} \subset C(S,U) \ .$$

(End of proof.)

From $F_i \subset F_{i+1}$ and $F_i \subset C(S,U)$ we conclude, as $C(S,U)$ is finite, that $\lim_{i\to\infty} F_i$ exists and is finite. The following theorem expresses that this limit is exactly $C(S,U)$.

THEOREM 5.3. If U denotes some finite set of associons, and S an arbitrary closure statement, then

$$C(S,U) = \lim_{i\to\infty} F_i \ .$$

PROOF. Let S denote the closure statement $x\colon E(x) :\Rightarrow T(x)$, and let j be such that $\lim_{i\to\infty} F_i = F_j$. Then $F_j = F_{j+1}$. We prove that F_j satisfies properties 1), 2), and 3) of Lemma 5.1 (p. 24). Obviously, F_j satisfies properties 1) and 2). From $F_j = F_{j+1}$ and the definition of F_{j+1} , i.e.

$$F_{j+1} = F_j \cup \{A\colon (\exists x\colon x \in Z(E,F_j)\colon A \in T(x))\} \ ,$$

we conclude

$$\{A\colon (\exists x\colon x \in Z(E,F_j)\colon A \in T(x))\} \subset F_j \ .$$

or

$$(\forall x\colon x \in Z(E,F_j)\colon T(x) \subset F_j) \ ,$$

or (apply Property 5.1 (p. 23)) $\hat{S}|F_j$. Hence, F_j satisfies property 3) as well. From Lemma 5.3 (p. 30) we know $F_j \subset C(S,U)$. As $C(S,U)$ is the smallest set satisfying properties 1), 2), and 3), we have $C(S,U) = F_j$.
(End of proof.)

PROPERTY 5.12. If U denotes some finite set of associons, and $S1$ and $S2$ denote arbitrary closure statements, then

$$C(S2,C(S1,U)) = C(S2,U) \tag{19}$$

if and only if

$$C(S1,U) \subset C(S2,U) \ . \tag{20}$$

PROOF. The fact that (19) implies (20) is an immediate consequence of the definition of $C(S,U)$. Next we assume (20) and derive (19). As

$$(C(S2,U) \setminus C(S1,U)) \subset (C(S2,U) \setminus U) \ ,$$

$C(S2,U) \setminus C(S1,U)$ does not contain associons that negatively match the left-hand side of $S2$. This allows us to apply Property 5.5 (p. 26) on $C(S1,U)$ and $C(S2,U)$, yielding

$$C(S2,C(S1,U)) \subset C(S2,C(S2,U)) \ ,$$

or (apply Property 5.7 (p. 27))

$$C(S2,C(S1,U)) \subset C(S2,U) \ . \tag{21}$$

As

$$(C(S1,U) \setminus U) \subset (C(S2,U) \setminus U) \ ,$$

we may apply Property 5.5 (p. 26) on U and $C(S1,U)$ as well, yielding

$$C(S2,U) \subset C(S2,C(S1,U)) \ . \tag{22}$$

From (21) and (22) follows (19).
(End of proof.)

One might wonder whether the mapping $U \rightarrow C(S,U)$ is a closure operator as defined in [12]. This is not the case. It does not satisfy the fourth Kuratowski closure axiom, requiring that the closure of the union of two sets equals the union of their closures. (Take, e.g., $U = \{(u)\}$, $V = \{(v)\}$, and S: $[u] \wedge [v] :\Rightarrow (w)$. Then $C(S,U) \cup C(S,V) \neq C(S,U \cup V)$.) It is more "powerful" than the closure of a finite set under a binary relation --for definition see below--, which does satisfy the Kuratowski clo-

sure axioms. The closure of a set under a binary relation can be expressed in terms of C(S,U) --see the following example--, the converse is not true.

EXAMPLE 5.1. Let V denote a finite set on which a binary relation R is defined, and let D denote some subset of V . Let, furthermore, sets G_i ($i \geq 0$) be given by the recurrence relation

$$G_0 = D ,$$

$$G_{i+1} = G_i \cup \{x: x \in V: (\exists y: y \in G_i: R(y,x))\} .$$

Then the *closure of* D *under* R , notation "D^R", is defined by

$$D^R = \lim_{i\to\infty} G_i .$$

(The above limit exists as for all i ($i \geq 0$) $G_i \subset G_{i+1} \subset V$.)

Let V , D , and R be given. Let the state U satisfy

$$(\forall x: [v,x]|U \equiv x \in V) \wedge$$
$$(\forall x: [d,x]|U \equiv x \in D) \wedge$$
$$(\forall x,y: [r,x,y]|U \equiv R(x,y)) .$$

Let, furthermore, S denote the closure statement

$$x,y: [d,y] \wedge [r,y,x] :\Rightarrow (d,x) .$$

We shall prove

$$(\forall x: [d,x]|C(S,U) \equiv x \in D^R) . \tag{23}$$

We prove, by mathematical induction, that for all x and y

$$([d,x]|F_i \equiv x \in G_i) \wedge ([r,y,x]|F_i \equiv R(y,x)) . \tag{24}$$

Then, as $\lim_{i\to\infty} F_i = C(S,U)$ and $\lim_{i\to\infty} G_i = D^R$, (23) follows immediately.

Relation (24) is satisfied for $i = 0$. Suppose, for all x and y

$$([d,x]|F_k \equiv x \in G_k) \wedge ([r,y,x]|F_k \equiv R(y,x)) \tag{25}$$

($k \geq 0$) . We derive

$$([d,x]|F_{k+1} \equiv x \in G_{k+1}) \wedge ([r,y,x]|F_{k+1} \equiv R(y,x)) . \tag{26}$$

By definition,

$$x \in G_{k+1} \equiv (x \in G_k \vee (\exists y: y \in G_k: R(y,x))) \ . \tag{27}$$

By applying the definition of F_i we obtain

$$\begin{aligned} &[d,x]|F_{k+1} \equiv ([d,x]|F_k \vee (\exists y: [d,y] \wedge [r,y,x])|F_k) \wedge \\ &[r,y,x]|F_{k+1} \equiv [r,y,x]|F_k \ . \end{aligned} \tag{28}$$

From (25), (27), and (28) follows (26).
(End of example.)

REMARK 5.2. The transitive closure, as defined in Example 4.2 (p. 19), can be expressed as a closure of a set under a binary relation: Let S be a binary relation on a finite set W. Define a binary relation R on $W \times W$ as

$$R((x_0,x_1),(y_0,y_1)) \equiv (x_0 = y_0 \wedge S(x_1,y_1)) \ ,$$

and define the set D to be $\{(x,y): x,y \in W: S(x,y)\}$. D is then a subset of $W \times W$, and we state without proof

$$\overset{*}{S}(x,y) \equiv (x,y) \in D^R \ .$$

(End of remark.)

CHAPTER 6

FORMAL DEFINITION OF THE CLOSURE STATEMENT

In the preceding chapter we have laid the foundation for the formal definition of the semantics of the closure statement. It is our intention to have the closure statement S transform the state U into the state C(S,U) . What in effect should happen is the assignment "U:= C(S,U)". We, therefore, define, in analogy to the Axiom of Assignment (vide Example 1.1 (p. 4)), the weakest pre-condition of the closure statement as follows.

If S denotes an arbitrary closure statement, and P some condition on the state, then for all states U

$$wp(S,P)|U \equiv P|C(S,U) \ .$$

(Like the assignment statement, it satisfies Properties 1.1, 1.2, and 1.3 (p. 3) for predicate transformers.)

PROPERTY 6.1. If S denotes an arbitrary closure statement, and P some condition on the state, then

$$\neg wp(S,P) \equiv wp(S,\neg P) \ .$$

PROOF. We use Remark 3.1 (p. 7). For any state U

$$\begin{aligned} (\neg wp(S,P))|U &\equiv \neg(wp(S,P)|U) \\ &\equiv \neg(P|C(S,U)) \\ &\equiv (\neg P)|C(S,U) \\ &\equiv wp(S,\neg P)|U \ . \end{aligned}$$

(End of proof.)

By substituting "false" --or "true"-- for "P" in the above property, we obtain

$$wp(S,true) \equiv true \ ,$$

which is interpreted as the guaranteed termination of closure statements.

THEOREM 6.1. The finiteness of the set of associons that characterizes the state, is an invariant of closure statements.

PROOF. Consequence of Theorem 5.2 (p. 26).
(End of proof.)

THEOREM 6.2. The closure statement is a deterministic statement.

PROOF. According to the definition in Chapter 1, a statement S is deterministic if and only if for any two conditions P and Q , and for all states U

$$wp(S,P \vee Q)|U \Rightarrow (wp(S,P) \vee wp(S,Q))|U \ .$$

Let S denote an arbitrary closure statement. We use Remark 3.1 (p. 7) and the definition of the closure statement to obtain

$$\begin{aligned} wp(S,P \vee Q)|U &\equiv (P \vee Q)|C(S,U) \\ &\equiv (P|C(S,U) \vee Q|C(S,U)) \\ &\equiv (wp(S,P)|U \vee wp(S,Q)|U) \\ &\equiv (wp(S,P) \vee wp(S,Q))|U \ . \end{aligned}$$

(End of proof.)

From the properties of closures, as proved in Chapter 5, we can derive equivalent properties of closure statements.

PROPERTY 6.2. If S denotes an arbitrary closure statement, and p some presence condition, then

$$p \Rightarrow wp(S,p) \ .$$

PROOF. Consequence of Property 5.2 (p. 26).
(End of proof.)

PROPERTY 6.3. If S denotes an arbitrary closure statement, and p a presence condition, such that no target associon of S fits p , then

$$p \equiv wp(S,p) \ .$$

PROOF. Consequence of Property 5.9 (p. 29).
(End of proof.)

Property 6.2 expresses that the execution of a closure statement does not destroy associons, Property 6.3 expresses that only target associons are created.

PROPERTY 6.4. If S denotes the closure statement x: E(x) :⇒ T(x) , then for all x

$$E(x) \Rightarrow wp(S,E(x)) \ .$$

PROOF. Consequence of Property 5.3 (p. 26).
(End of proof.)

PROPERTY 6.5. If S denotes an arbitrary closure statement, then

$$wp(S,\hat{S}) \equiv \text{true} .$$

PROOF. Consequence of Property 5.4 (p. 26).
(End of proof.)

PROPERTY 6.6. If S denotes an arbitrary closure statement and P some condition on the state, then

$$\hat{S} \text{ implies } wp(S,P) \equiv P .$$

PROOF. Let U denote an arbitrary state. We assume $\hat{S}|U$. Then, according to Property 5.6 (p. 27),

$$C(S,U) = U . \tag{1}$$

By definition,

$$wp(S,P)|U \equiv P|C(S,U) . \tag{2}$$

From (1) and (2) we conclude

$$wp(S,P)|U \equiv P|U .$$

(End of proof.)

Property 6.5 expresses that a closure statement S establishes the truth of $\hat{S}$. Property 6.6 expresses that, if $\hat{S}$ was true to start with, then S is equivalent to the statement "skip".

PROPERTY 6.7. If S denotes an arbitrary closure statement, and P some condition on the state, then

$$wp("S;S",P) \equiv wp(S,P) .$$

PROOF. By definition,

$$wp("S;S",P) \equiv wp(S,wp(S,P)) .$$

Therefore, for any state U ,

$$\begin{aligned} wp("S;S",P)|U &\equiv wp(S,wp(S,P))|U \\ &\equiv wp(S,P)|C(S,U) \\ &\equiv P|C(S,C(S,U)) . \end{aligned}$$

According to Property 5.7 (p. 27) $C(S,C(S,U)) = C(S,U)$, consequently

$$wp("S;S",P)\,|\,U \equiv P\,|\,C(S,U)$$
$$\equiv wp(S,P)\,|\,U \ .$$

(End of proof.)

PROPERTY 6.8. If A denotes an arbitrary associon, and S the noncascading closure statement $x\colon E(x) :\Rightarrow T(x)$, then

$$wp(S,\tilde{A}) \equiv (\tilde{A} \vee (\exists x\colon E(x)\colon A \in T(x))) \ .$$

PROOF. Consequence of Property 5.10 (p. 29).
(End of proof.)

EXAMPLE 6.1. If S denotes the closure statement

$$x\colon [v,x] :\Rightarrow (w,x) \ .$$

then, for any name u ,

$$wp(S,[v,u]) \equiv [v,u] \ , \quad \text{(Apply Property 6.3 (p. 36),)}$$

and

$$wp(S,[w,u]) \equiv ([w,u] \vee [v,u]) \ .$$

(According to Property 6.8.

$$wp(S,[w,u]) \equiv ([w,u] \vee (\exists x\colon [v,x]\colon (w,u) \in \{(w,x)\}))$$
$$\equiv ([w,u] \vee [v,u]) \ . \ )$$

(End of example.)

PROPERTY 6.9. If S1 and S2 denote arbitrary closure statements, and P denotes some condition on the state, then

$$wp(S1,wp(S2,R)) \equiv wp(S2,R)$$

if and only if for all associons A

$$wp(S1,\tilde{A}) \Rightarrow wp(S2,\tilde{A}) \ .$$

PROOF. Consequence of Property 5.12 (p. 32).
(End of proof.)

From Property 6.3 (p. 36) and Property 6.5 (p. 37) we know that the execution of a closure statement S establishes the truth of $\hat{S}$ by creating target associons of S . The following theorem expresses that not too

many associons are created. More precisely: if we call a target associon A of the statement $x: E(x) :\Rightarrow T(x)$ "wrong" in state U when $\neg(\exists x: x \in Z(E,U): A \in T(x))$, then the theorem expresses --besides the fact that only target associons are created-- that if prior to the execution of S there is no wrong target associon present, then after the execution of S there will be no wrong target associon present either.

THEOREM 6.3. *Invariance theorem for closure statements.*
Let S denote the closure statement $x: E(x) :\Rightarrow T(x)$, A an arbitrary associon, and P the implication

$$\tilde{A} \Rightarrow (\exists x: E(x): A \in T(x)) ,$$

then

$$P \Rightarrow wp(S,P) .$$

PROOF. Consequence of Property 5.11 (p. 29).
(End of proof.)

EXAMPLE 6.2. We choose for S the closure statement

$$x: [v,x] :\Rightarrow (w,x) .$$

As a first application of the invariance theorem we choose for A an associon that is not a target associon of S . Let u denote an arbitrary name. We choose for A the associon (v,u) . As for all names x only the associon (w,x) is contained in $T(x)$, there exists no x such that $A \in T(x)$. P , consequently, is equivalent to $\neg[v,u]$. The invariance theorem then expresses

$$\neg[v,u] \Rightarrow wp(S,\neg[v,u]) ,$$

i.e. $\neg[v,u]$ is an invariant of S .

We could also have chosen for A an arbitrary target associon (w,u) . Then there is indeed a value of x such that $A \in T(x)$, viz. the value u . P then becomes

$$[w,u] \Rightarrow E(u) ,$$

or

$$[w,u] \Rightarrow [v,u] .$$

As u denotes an arbitrary name, we have derived the invariance of the im-

plication

$$(\forall x\colon [w,x] \Rightarrow [v,x]) \ . \qquad (3)$$

According to Property 6.5 (p. 37), the execution of S establishes the truth of the implication

$$(\forall x\colon [v,x] \Rightarrow [w,x]) \ .$$

If (3) is satisfied to start with, it will remain satisfied, and the execution of S establishes the truth of the equivalence

$$(\forall x\colon [w,x] \equiv [v,x]) \ .$$

(End of example.)

Suppose we have to write a program that establishes the truth of a relation R . We then have to find an implication $\hat{S}$ and a relation P , such that initially P holds, $P \Rightarrow wp(S,P)$, and $(P \wedge \hat{S}) \Rightarrow R$. Often one can write R as an equivalence, that initially holds in one direction. We then try to establish, under invariance of that initially satisfied implication, the implication in the other direction as well. An instance of this we encountered in the second part of the above example.

The invariance theorem has been formulated in such a fashion that it may provide a practicable tool for proving the invariance of implications. We shall demonstrate this application of the invariance theorem in two more examples. In the above example we obtained the invariant "inverse" implication by simply reversing the implication sign in $\hat{S}$. We have chosen the following two examples to demonstrate two cases in which more care is required.

EXAMPLE 6.3. This is an example in which not all of the unknowns occur in each target associon format. The closure statement

S $\qquad x,y\colon [d,y] \wedge [r,y,x] \ :\Rightarrow \ (d,x)$

establishes the truth of the implication

$\hat{S}$ $\qquad (\forall x,y\colon ([d,y] \wedge [r,y,x]) \Rightarrow [d,x]) \ .$

If we reverse the implication sign in $\hat{S}$, we obtain a relation that is never satisfied. Although it is an invariant of S , it is of little use. We can write $\hat{S}$ as

$$(\forall x\colon (\exists y\colon [d,y] \wedge [r,y,x]) \Rightarrow [d,x]) \ . \qquad (4)$$

We shall prove that the "inverse" of (4), i.e.

$$(\forall x\colon [d,x] \Rightarrow (\exists y\colon [d,y] \wedge [r,y,x])) \ , \tag{5}$$

is kept invariant by S .

We choose for A the associon (d,u) , in which u denotes an arbitrary name. The invariance theorem then expresses the invariance of

$$[d,u] \Rightarrow (\exists x,y\colon [d,y] \wedge [r,y,x]\colon (d,u) \in \{(d,x)\}) \ ,$$

or

$$[d,u] \Rightarrow (\exists y\colon [d,y] \wedge [r,y,u]) \ .$$

As u denotes an arbitrary name, this is exactly (5). If initially (5) is satisfied, then S establishes the truth of the equivalence

$$(\forall x\colon [d,x] \equiv (\exists y\colon [d,y] \wedge [r,y,x])) \ .$$

(End of example.)

EXAMPLE 6.4. This is an example in which for some target associon A there exists more than one x such that $A \in T(x)$. Let a set V of integers, and the greater-than relation ">" on these integers, be given by

$$(\forall x\colon [v,x] \equiv \text{" } x \text{ is the name of an element of } V \text{ "}) \wedge$$
$$(\forall x,y\colon [gr,x,y] \equiv ([v,x] \wedge [v,y] \wedge x > y)) \ .$$

Suppose, furthermore, that we wish to record in associons (d,?,?) the names of mutually distinct integers in V , i.e. we wish to establish the truth of

$$(\forall x,y\colon [d,x,y] \equiv ([gr,x,y] \vee [gr,y,x])) \ . \tag{6}$$

The closure statement

S $\quad x,y\colon [gr,x,y] :\Rightarrow (d,x,y),(d,y,x)$

establishes the truth of the implication

$\hat{S}$ $\quad (\forall x,y\colon [gr,x,y] \Rightarrow ([d,x,y] \wedge [d,y,x])) \ .$

If we simply reverse the implication sign in $\hat{S}$, we obtain a relation that is surely not an invariant of S . The relation $\hat{S}$ is equivalent to

$$(\forall x,y\colon ([gr,x,y] \vee [gr,y,x]) \Rightarrow [d,x,y]) \ . \tag{7}$$

We shall prove that the "inverse" of (7), i.e.

$$(\forall x,y\colon [d,x,y] \Rightarrow ([gr,x,y] \vee [gr,y,x])) \ , \tag{8}$$

is kept invariant by S . If (8) is satisfied to start with --e.g. because ¬[d,?,?] --, then S establishes the truth of (6).

We choose for A the associon (d,u,v) , in which u and v denote arbitrary names. The invariance theorem then expresses the invariance of

$$[d,u,v] \Rightarrow (\exists x,y\colon [gr,x,y]\colon (d,u,v) \in \{(d,x,y),(d,y,x)\}) \ .$$

The condition $(d,u,v) \in \{(d,x,y),(d,y,x)\}$ has two solutions for (x,y) , viz. (u,v) and (v,u) . We thus obtain

$$[d,u,v] \Rightarrow ([gr,u,v] \vee [gr,v,u]) \ ,$$

which is exactly (8).
(End of example.)

CHAPTER 7

SOME SMALL EXAMPLES

EXAMPLE 7.1. *Convex hull.*

A finite collection of (at least three) points in the plane, with no three points on the same line, is given by

$$(\forall x: [p,x] \equiv \text{" } x \text{ is the name of a point"}) \wedge$$
$$(\forall x,y,z: [t,x,y,z] \equiv ([p,x] \wedge [p,y] \wedge [p,z] \wedge \text{"triangle } xyz \text{ is a clockwise triangle"})) .$$

It is, furthermore, given that $\neg[h,?,?]$ holds, and it is requested to write a program that establishes the truth of the relation

R $\quad (\forall x,y: [h,x,y] \equiv$ "edge xy is an edge of the clockwise convex hull")

The condition "edge xy is an edge of the clockwise convex hull" is equivalent to "there exists a point v such that triangle xyv is a clockwise triangle, and there does not exist a point w such that triangle yxw is a clockwise triangle". Hence, we can write R as

$$(\forall x,y: [h,x,y] \equiv ([t,x,y,?] \wedge \neg[t,y,x,?])) .$$

We prove $[h,?,?] \Rightarrow wp(S,R)$ for the program

S $\quad x,y: [t,x,y,?] \wedge \neg[t,y,x,?] :\Rightarrow (h,x,y)$.

We split R into two conditions:

P $\quad (\forall x,y: [h,x,y] \Rightarrow ([t,x,y,?] \wedge \neg[t,y,x,?]))$,

$\hat{S}$ $\quad (\forall x,y: ([t,x,y,?] \wedge \neg[t,y,x,?]) \Rightarrow [h,x,y])$.

Then

$$R \equiv (P \wedge \hat{S}) . \tag{1}$$

From Property 6.5 (p. 37) we know

$$wp(S,\hat{S}) . \tag{2}$$

From $\neg[h,?,?]$ we conclude P . From the invariance theorem (p. 39) we conclude

$$P \Rightarrow wp(S,P) . \tag{3}$$

From P and (3) we conclude

$$wp(S,P) \ . \qquad (4)$$

From (2), (4), and Property 1.2 (p. 3) we conclude $wp(S,\hat{S} \wedge P)$, or --with (1)-- $wp(S,R)$.

EXAMPLE 7.2. *Library administration.*

A library is a collection of books. Each book has an author, and each author has at least one book in the library. Books are either on loan or in stock. Each book on loan has a borrower. The library is given by

$(\forall x,y: [b,x,y] \equiv$ " x is the name of a book and y is the name of the author of that book") $\wedge$

$(\forall x,y: [l,x,y] \equiv$ " x is the name of a book that is on loan and y is the name of the borrower of that book") .

It is, furthermore, given that $\neg[r,?] \wedge \neg[s,?]$ holds, and it is requested to write two programs. One program should record in associons (r,?) the names of authors of whom there is a book on loan to another author. The other program should record in associons (s,?) the names of authors of whom all books are on loan. To be more precise: the one program should establish the truth of the relation

R1 $\quad (\forall x: [r,x] \equiv (\exists y,z: [b,y,x] \wedge [l,y,z] \wedge [b,?,z] \wedge x \neq z))$,

and the other program should establish the truth of

R2 $\quad (\forall x: [s,x] \equiv ([b,?,x] \wedge (\forall y: [b,y,x] \Rightarrow [l,y,?])))$.

Initially the left-hand side of R1 implies the right-hand side. The program

$$x,y,z: [b,y,x] \wedge [l,y,z] \wedge [b,?,z] \wedge x \neq z \ :\Rightarrow (r,x)$$

establishes also the implication in the other direction, and hence R1 .

The relation R2 can be written as

$$(\forall x: [s,x] \equiv ([b,?,x] \wedge \neg(\exists y: [b,y,x] \wedge \neg[l,y,?]))) \ .$$

We introduce associons (h,?) , in which we record the solution set of the existentially quantified condition, i.e. we establish the truth of the relation

$$(\forall x: [h,x] \equiv (\exists y: [b,y,x] \wedge \neg[l,y,?])) \ . \qquad (5)$$

This can be accomplished by the closure statement

$$x,y\colon [b,y,x] \wedge \neg[l,y,?] :\Rightarrow (h,x) \ .$$

(In associons (h,?) we then have recorded the names of the authors of whom there is a book in stock.) If (5) holds, the relation R2 is equivalent to

$$(\forall x\colon [s,x] \equiv ([b,?,x] \wedge \neg[h,x])) \ ,$$

the truth of which can be established by the closure statement

$$x\colon [b,?,x] \wedge \neg[h,x] :\Rightarrow (s,x) \ .$$

As the execution of this statement does not violate the truth of (5) --apply Property 6.3 (p. 36)-- we have also established the truth of R2 .

Program:

```
loc (h,?):
    x,y: [b,y,x] ∧ ¬[l,y,?] :⇒ (h,x);
    x: [b,?,x] ∧ ¬[h,x] :⇒ (s,x)
col
```

The occurrence of the universal quantifier in the right-hand side of R2 necessitated the introduction of the associons (h,?) . We shall encounter this theme more often.

EXAMPLE 7.3. *Missionaries and cannibals.*

Three missionaries and three cannibals wish to cross a river from the left bank to the right bank. For this purpose a boat is situated at the left bank. The boat can sail from one bank to another with either one or two passengers. On either bank the missionaries will be eaten by the cannibals if there are more cannibals than missionaries. We are requested to write a program that creates the (initially absent) associon (yes) if and only if the crossing is possible.

We only consider the moments at which the boat is at either bank. If at those moments the situation is legitimate (in the sense that no missionary will be eaten), then the situation during the intermediate time intervals will be legitimate as well. We represent each situation by a pair of quaternary digits (x,y) , with x and y being respectively the number of missionaries on the right bank and the number of cannibals on the right bank.

(For the time being we do not represent the position of the boat.) The initial situation is then (zero,zero) , and the desired final situation is (three,three) . The quaternary digits and their successor relation can be recorded by the closure statement

```
[] :⇒ (q,zero),(q,one),(q,two),(q,three),
      (suc,zero,one),(suc,one,two),(suc,two,three) .
```

We wish to record in associons (r,?,?,?,?) the possible transformations (between legitimate situations) that a crossing of the boat can effect, i.e. we wish to establish the truth of the relation

```
(∀x1,y1,x2,y2: [r,x1,y1,x2,y2] ≡ "situation (x1,y1)
can by a crossing from left to right be transformed          (6)
into (x2,y2) ") .
```

We then also have characterized the crossings from right to left, as (6) implies

```
(∀x1,y1,x2,y2: [r,x1,y1,x2,y2] ≡ "situation (x2,y2)
can by a crossing from right to left be transformed
into (x1,y1) ") .
```

And hence we have

```
(∀x1,y1,x3,y3: (∃x2,y2: [r,x1,y1,x2,y2] ∧ [r,x3,y3,x2,y2]) ≡
"situation (x1,y1) with the boat at the left can be trans-
 formed into (x3,y3) with the boat again at the left") .
```

Recording the reachable situations in associons (d,?,?) --cf. Example 5.1 (p. 33)-- the program will get the following structure.

```
loc (q,?),(suc,?,?),(d,?,?),(r,?,?,?,?):
    [] :⇒ (q,zero),(q,one),(q,two),(q,three),
          (suc,zero,one),(suc,one,two),(suc,two,three);
    "establish (6)";
    [] :⇒ (d,zero,zero);
    x1,y1,x2,y2,x3,y3: [d,x1,y1] ∧ [r,x1,y1,x2,y2]
                       ∧ [r,x3,y3,x2,y2] :⇒ (d,x3,y3);
    x,y: [d,x,y] ∧ [r,x,y,three,three] :⇒ (yes)
col
```

In the refinement of "establish (6)" we first (temporarily) forget the fact that some pairs of quaternary digits represent illegitimate situations,

i.e. we first establish the truth of

(∀x1,y1,x2,y2: [hh,x1,y1,x2,y2] ≡ "the (possibly illegitimate) situation (x1,y1) can by a crossing from left to right be transformed into the (possibly illegitimate) situation (x2,y2) ") . (7)

This can be accomplished by

```
x1,x2,y: [suc,x1,x2] ∧ [q,y] :⇒ (h,x1,y,x2,y),(h,y,x1,y,x2),
                                 (hh,x1,y,x2,y),(hh,y,x1,y,x2);
x1,y1,x2,y2,x3,y3: [h,x1,y1,x2,y2] ∧ [h,x2,y2,x3,y3]
                                 :⇒ (hh,x1,y1,x3,y3) .
```

(The associons (h,?,?,?,?) record the transformations under a one-man crossing.)

Next we want to restrict (7) to legitimate situations only. Legitimate situations are those in which (on either bank) the number of missionaries equals the number of cannibals, and those in which all missionaries are on one bank. We record the legitimate situations in associons (l,?,?) .

"establish (6)":

```
loc (h,?,?,?,?),(hh,?,?,?,?),(l,?,?):
    x1,x2,y: [suc,x1,x2] ∧ [q,y] :⇒ (h,x1,y,x2,y),(h,y,x1,y,x2),
                                    (hh,x1,y,x2,y),(hh,y,x1,y,x2);
    x1,y1,x2,y2,x3,y3: [h,x1,y1,x2,y2] ∧ [h,x2,y2,x3,y3]
                                    :⇒ (hh,x1,y1,x3,y3);
    x: [q,x] :⇒ (l,x,x),(l,three,x),(l,zero,x);
    x1,y1,x2,y2: [hh,x1,y1,x2,y2] ∧ [l,x1,y1] ∧ [l,x2,y2] :⇒ (r,x1,y1,x2,y2)
col
```

Instead of applying fancy search techniques, we have programmed a tree search by (concurrently?) generating the whole tree.

EXAMPLE 7.4. *Trap of a directed graph.*

We first give some nomenclature on directed graphs. A *walk* (from v_0 to v_n) in a directed graph is an alternating sequence of vertices and arcs $v_0,a_1,v_1,\ldots,a_n,v_n$ ($n \geq 0$) in which each arc a_i is $v_{i-1}v_i$ ($1 \leq i \leq n$). The *length* of such a walk is n . A *nontrivial* walk is a walk that contains at least one arc. A *path* is a walk in which all vertices are

distinct. A *cycle* is a nontrivial walk in which all vertices except the last are mutually distinct, and in which the last vertex equals the first. An *acyclic* graph is a graph that does not contain a cycle. A vertex v is *reachable* from a vertex u if and only if there exists a path from u to v . A *terminal* vertex is a vertex that does not have an outgoing arc.

It is requested to write a program that determines the trap of a given finite directed graph G . A vertex belongs to the *trap* if and only if there does not exist a walk from that vertex to a vertex in a cycle. (The trap of an acyclic graph, consequently, equals the set of all vertices.)

Of the graph G is given

(∀x: [v,x] ≡ " x is the name of a vertex of G ") ∧
(∀x,y: [s,x,y] ≡ " G has an arc from vertex x to vertex y ").

Let, furthermore, be given that ¬[t,?] holds, and let it be requested to establish the truth of the relation

R (∀x: [t,x] ≡ "vertex x in the trap of G ") .

We first establish the truth of

(∀x,y: [w,x,y] ≡ "there exists a nontrivial walk
from vertex x to vertex y ") .

(Which is nothing else than the generation of the transitive closure of the relation "connected by an arc", vide Example 4.2 (p. 19).) As we then also have

(∀x: [w,x,x] ≡ "vertex x in a cycle") ,

the relation R can be written as

(∀x: [t,x] ≡ ([v,x] ∧ ¬(∃y: [w,x,y] ∧ [w,y,y]))) .

We introduce associons (h,?) in which we record the solution set of the existentially quantified condition, i.e. we establish the truth of the relation

(∀x: [h,x] ≡ (∃y: [w,x,y] ∧ [w,y,y])) .

(The associons (h,?) record the names of the vertices for which there exists a walk to a vertex in a cycle.) The relation R is then equivalent to

(∀x: [t,x] ≡ ([v,x] ∧ ¬[h,x])) .

Program:

```
loc (w,?,?),(h,?):
    x,y: [s,x,y] :⇒ (w,x,y);
    x,y,z: [w,x,y] ∧ [w,y,z] :⇒ (w,x,z);
    x,y: [w,x,y] ∧ [w,y,y] :⇒ (h,x);
    x: [v,x] ∧ ¬[h,x] :⇒ (t,x)
col
```

EXAMPLE 7.5. *Kernel of a directed graph.*

The *kernel* of a directed graph is defined as the set of all vertices from which no terminal vertex can be reached. (The kernel and the trap are, consequently, disjoint.)

It is requested to write a program that determines the kernel of a finite directed graph G , given as in Example 7.4 (p. 47), while initially ¬[k,?] holds. The program should do so by establishing the truth of the relation

R (∀x: [k,x] ≡ "vertex x in the kernel of G ") ,

which is equivalent to

(∀x: [k,x] ≡ ([v,x] ∧ ¬"from vertex x a terminal vertex can be reached")) .

If the truth of

(∀x: [h,x] ≡ "from vertex x a terminal vertex can be reached")

has been established, then R can be written as

(∀x: [k,x] ≡ ([v,x] ∧ ¬[h,x])) .

Program:

```
loc (h,?):
    x: [v,x] ∧ ¬[s,x,?] :⇒ (h,x);
    x,y: [s,x,y] ∧ [h,y] :⇒ (h,x);
    x: [v,x] ∧ ¬[h,x] :⇒ (k,x)
col
```

EXAMPLE 7.6. *Determining whether a directed graph is acyclic.*

Let a finite directed graph G be given as in Example 7.4 (p. 47). Let, furthermore, be given that ¬[c] holds, and let it be requested to write a program that establishes the truth of the relation

R [c] ≡ " G contains a cycle" .

After establishing the truth of

(∀x,y: [w,x,y] ≡ "there exists a nontrivial walk from vertex x to vertex y ") ,

(cf. Example 7.4 (p. 47)) the relation R is equivalent to

[c] ≡ (∃x: [w,x,x]) .

Program:

```
loc (w,?,?):
    x,y: [s,x,y] :⇒ (w,x,y);
    x,y,z: [w,x,y] ∧ [w,y,z] :⇒ (w,x,z);
    x: [w,x,x] :⇒ (c)
col
```

EXAMPLE 7.7. *Determining whether an undirected graph is acyclic.*

We first give some nomenclature on undirected graphs. *Adjacent vertices* are vertices that are joined by an edge. A *loop* is an edge joining a vertex to itself. If two vertices are joined by more than one edge, then these edges are called *multiple edges*. A *walk* and a *path* are defined as for directed graphs. An undirected graph is *connected* if and only if every two vertices are joined by a path. A *cycle* in an undirected graph without loops or multiple edges is a walk of at least three edges in which all vertices except the last are mutually distinct, and in which the last vertex equals the first.

A finite undirected graph G , without loops or multiple edges, is given by

(∀x: [v,x] ≡ " x is the name of a vertex of G ") ∧
(∀x,y: [a,x,y] ≡ "vertices x and y are adjacent") .

(Every edge gives rise to the presence of two associons (a,?,?) .) Let, furthermore, be given that ¬[c] holds, and let it be requested to write

a program that establishes the truth of the relation

R [c] ≡ " G contains a cycle" .

In order to find a cycle in G , we look for a "proper triangle of vertices", i.e. three mutually distinct vertices x , y , and z , such that there exists a walk between x and y (not through z), a walk between y and z (not through x), and a walk between z and x (not through y). We, therefore, establish the truth of

(∀x,y,z: [w,x,y,z] ≡ "there exists a walk between vertex y and vertex z (y ≠ z) that does not contain vertex x ") .

The relation R is then equivalent to

[c] ≡ (∃x,y,z: [w,z,x,y] ∧ [w,x,y,z] ∧ [w,y,z,x]) .

Program:

```
loc (w,?,?,?):
    x,y,z: [a,x,y] ∧ [v,z] ∧ x ≠ z ∧ y ≠ z :⇒ (w,z,x,y);
    x,y,z,u: [w,u,x,y] ∧ [w,u,y,z] ∧ x ≠ z :⇒ (w,u,x,z);
    x,y,z: [w,z,x,y] ∧ [w,x,y,z] ∧ [w,y,z,x] :⇒ (c)
col
```

EXAMPLE 7.8. *Safely connected vertices.*

Let G be a finite undirected graph, possibly containing loops and multiple edges. Two paths in G that have no edge in common are called *edge-disjoint*. Two vertices are *safely connected* if and only if they are connected by (at least) two edge-disjoint paths.

The graph G is given by

(∀x: [v,x] ≡ " x is the name of a vertex of G ") ∧
(∀x: [e,x] ≡ " x is the name of an edge of G ") ∧
(∀x,y,z: [j,x,y,z] ≡ "edge x joins vertices y and z ") .

Let, furthermore, be given that ¬[c,?,?] holds, and let it be requested to write a program that establishes the truth of the relation

R (∀x,y: [c,x,y] ≡ "vertices x and y are safely connected") .

One property of safely connected vertices is that if vertices u and v are safely connected, then any two vertices on the edge-disjoint paths

between u and v are safely connected as well.

Another property is that safe connectedness is an equivalence relation. It is easy to see that the relation is symmetric and reflexive (for reflexivity: choose twice the path consisting of the vertex only). The transitivity requires an additional exploration:

Safely connected vertices are those vertices that can not be separated by the removal of one edge from the graph. (It is obvious that two safely connected vertices cannot be separated by the removal of one edge; the inverse assertion, i.e. that two vertices that cannot be separated by the removal of one edge are safely connected, can be proved by induction on the length of their shortest connecting path.) Suppose now that the vertices x and y are safely connected, and that the vertices y and z are safely connected. If we remove one edge from the graph, then x and y remain connected, and y and z remain connected, and, therefore, x and z remain connected too. We conclude that x and z are also safely connected.

From the above two properties we conclude that we can find all safely connected vertices by first looking for adjacent vertices that are safely connected, and then generating the transitive (and reflexive) closure of that relation.

We first establish the truth of

(∀x,y,z: [w,x,y,z] ≡ "there exists a nontrivial walk between vertex y and vertex z that does not contain edge x ") .

Next we establish the truth of

(∀x,y: [c,x,y] ≡ "the adjacent vertices x and y are safely connected") , (8)

after which the truth of R is established by extending the relation of (8) to its transitive and reflexive closure.

Program:

```
loc (w,?,?,?):
    x,y,z,u: [j,x,y,z] ∧ [e,u] ∧ x ≠ u :⇒ (w,u,y,z);
    x,y,z,u: [w,x,y,z] ∧ [w,x,z,u] :⇒ (w,x,y,u);
    x,y,z: [j,x,y,z] ∧ [w,x,y,z] :⇒ (c,y,z);
    x,y,z: [c,x,y] ∧ [c,y,z] :⇒ (c,x,z);
    x: [v,x] :⇒ (c,x,x)
col
```

EXAMPLE 7.9. *Minimal transition pair.*

If a directed graph has an arc from vertex u to vertex v, then u is called a *predecessor* of v, and v is called a *successor* of u. Let K be some vertex of a directed graph G. The *minimal transition pair* with K as *initial vertex*, notation "$(M_{(K)},N_{(K)})$", is the unique smallest pair --i.e. the pair for which the sum of the cardinalities of the components is minimal-- (M,N) such that

1) M and N are sets of vertices of G,
2) $K \in M$,
3) all successors of vertices in M are in N,
4) all predecessors of vertices in N are in M.

Such a unique smallest pair exists, because --if V denotes the set of vertices of G -- (V,V) satisfies the properties 1), 2), 3), and 4), and if the pairs (M0,N0) and (M1,N1) satisfy them, then the pair $(M0 \cap M1, N0 \cap N1)$ satisfies them too.

Let a finite directed graph G be given as in Example 7.4 (p. 47) and let k be the name of the initial vertex K. Let, furthermore, be given that $\neg[m,?] \wedge \neg[n,?]$ holds, and let it be requested to write a program that establishes the truth of the relation

R $\quad (\forall x: [m,x] \equiv x \in M_{(K)}) \wedge (\forall x: [n,x] \equiv x \in N_{(K)})$.

We can write the properties 1), 2), 3), and 4) as follows.

R1 $\quad (\forall x: ([m,x] \vee [n,x]) \Rightarrow [v,x])$,

R2 $\quad [m,k]$,

R3 $\quad (\forall x,y: ([s,x,y] \wedge [m,x]) \Rightarrow [n,y])$,

R4 $\quad (\forall x,y: ([s,x,y] \wedge [n,y]) \Rightarrow [m,x])$.

We are requested to write a program that establishes R1 ∧ R2 ∧ R3 ∧ R4 by creating as few associons (m,?) and (n,?) as possible. R3 ∧ R4 is equivalent to

$$(\forall x,y: ([s,x,y] \wedge ([m,x] \vee [n,y])) \Rightarrow ([m,x] \wedge [n,y])) .$$

Program:

```
[] :⇒ (m,k);
x,y: [s,x,y] ∧ ([m,x] ∨ [n,y]) :⇒ (m,x),(n,y)
```

Upon termination R1 (still) holds. R2 is established by the first statement. The second statement establishes R3 ∧ R4 . From the definition of C(S,U) we conclude that the minimal pair is generated.

CHAPTER 8

AN APPRECIATION OF THE REPETITIVE CONSTRUCT

In Chapter 4 we stated that it is the purpose of a computation to make equivalences true. We were, for instance, requested to establish the truth of the equivalence

$$(\forall x\colon [odd,x] \equiv ([int,x] \wedge \neg[even,x])) \ , \qquad (1)$$

by creating associons (odd,?) , while initially the implication

$$(\forall x\colon [odd,x] \Rightarrow ([int,x] \wedge \neg[even,x])) \qquad (2)$$

was satisfied. The closure statement

$$x\colon [int,x] \wedge \neg[even,x] :\Rightarrow (odd,x)$$

establishes the implication

$$(\forall x\colon ([int,x] \wedge \neg[even,x]) \Rightarrow [odd,x]) \ ,$$

under invariance of its "inverse" implication (2), thus establishing the truth of the equivalence (1). This is the general pattern of associon computations: the equivalence to be established holds initially in one direction; under invariance of that implication the computation establishes the implication in the other direction as well, thus ensuring the desired equivalence to hold. Theorem 6.3 (p. 39) implies that the closure statement does indeed establish implications under invariance of their inverse implications. And as such we have used the closure statement in our examples to achieve the desired equivalences. A problem arises, however, as soon as the antecedent of the implication to be established is more complicated than we can express in the left-hand side of a closure statement. In particular may we expect problems in the case that the antecedent contains a universal (or a negated existential) quantifier.

As an example we shall focus our attention on the generation of an arbitrary clique of an undirected graph.

We first give some more nomenclature on undirected graphs. A *subgraph* of a graph G is a graph having all its vertices and edges in G . G is a *supergraph* of G' if and only if G' is a subgraph of G . A *spanning subgraph* of G is a subgraph containing all the vertices of G . A *complete graph* is a graph (without loops or multiple edges) in which all vertices are

adjacent. A *clique* of a graph is a complete subgraph that is not a proper subgraph of a complete subgraph.

Let an undirected graph G (possibly containing loops) be given as in Example 7.7 (p. 50). Let, furthermore, be given that ¬[c,?] holds, and let it be requested to write a program that establishes --by creating associons (c,?) -- the truth of the relation

R (∀x: [c,x] ≡ ([v,x] ∧ (∀y: y ≠ x: [c,y] ⇒ [a,x,y]))) .

(We generate an arbitrary clique, recording the vertices of the clique in associons (c,?) . As a clique is a complete graph, it can be characterized by its vertex set.)

REMARK 8.1. The generation of an arbitrary clique is a nondeterministic affair. If we want our program to be such that it may establish any of the permissible final states --which seems a laudable goal--, then it should not surprise us that we shall need other constructs besides the (deterministic) closure statement.
(End of remark.)

The implication in the one direction

P1 (∀x: [c,x] ⇒ ([v,x] ∧ (∀y: y ≠ x: [c,y] ⇒ [a,x,y])))

holds initially. P1 expresses that no "wrong" associons are present. Pronouncing "[c,x]" as "x in the clique", it expresses that for all x in the clique, x is the name of a vertex that is adjacent to all other vertices in the clique. Under invariance of P1 the truth of the implication in the other direction, i.e.

(∀x: ([v,x] ∧ (∀y: y ≠ x: [c,y] ⇒ [a,x,y])) ⇒ [c,x]) , (3)

has to be established. Relation (3) expresses that "sufficiently many" associons have been created. It expresses that all vertices x that are adjacent to all other vertices in the clique, are in the clique as well. Relation (3) is equivalent to

(∀x: ([v,x] ∧ ¬(∃y: y ≠ x: [c,y] ∧ ¬[a,x,y])) ⇒ [c,x]) . (4)

We are faced with the situation that we have to establish the truth of an implication in which the antecedent contains a negated existential quantifier.

If all quantified variables of the negated existential quantifier --in (4) the "y" only-- had occurred only once in the quantified condition, then the question mark "?" would have provided a simple solution. We have met this case in Example 7.1 (Convex Hull (p. 43)), in which the implication was

$$(\forall x,y\colon ((\exists z\colon [t,x,y,z]) \wedge \neg(\exists w\colon [t,y,x,w])) \Rightarrow [h,x,y]) \ ,$$

the truth of which could be established by the program

$$x,y,z\colon [t,x,y,z] \wedge \neg[t,y,x,?] :\Rightarrow (h,x,y) \ .$$

(The "z" may be replaced by a question-mark as well, but that transformation is less essential.)

If some quantified variable of the negated existential quantifier occurs more than once in the quantified condition --as the "y" in (4)--, then we introduce new associons in which we record the solution set of the existentially quantified condition.

We have done so, for instance, in Example 7.2 (Library Administration (p. 44)), in which the (second) implication to be established was

$$(\forall x\colon ([b,?,x] \wedge \neg(\exists y\colon [b,y,x] \wedge \neg[l,y,?])) \Rightarrow [s,x]) \ . \qquad (5)$$

We introduced new associons $(h,?)$ and established the truth of

$$(\forall x\colon [h,x] \equiv (\exists y\colon [b,y,x] \wedge \neg[l,y,?])) \ . \qquad (6)$$

As (6) is not matched by the target associons $(s,?)$, it is an invariant of the closure statement

$$x\colon [b,?,x] \wedge \neg[h,x] :\Rightarrow (s,x) \ .$$

The execution of this statement, consequently, establishes the truth of (5).

We call this technique "recording the quantified condition". In the case of the arbitrary clique, this technique gives rise to the introduction of the extra invariant

P2 $\quad (\forall x\colon [h,x] \equiv ([v,x] \wedge (\exists y\colon y \neq x\colon [c,y] \wedge \neg[a,x,y]))) \ .$

If P2 is satisfied, P1 can be written as

P1' $\quad (\forall x\colon [c,x] \Rightarrow ([v,x] \wedge \neg[h,x])) \ ,$

and (4) can be written as

$$(\forall x\colon ([v,x] \wedge \neg[h,x]) \Rightarrow [c,x]) \ . \qquad (7)$$

We are requested to establish the truth of (7) under invariance of
P1' ∧ P2 .

So far we have followed the same pattern as in the example of the Library Administration. There is, however, one important difference: on account of the match of the target associons (c,?) and P2 the creation of target associons can violate the truth of P2 . As this match is a positive one, the creation of associons (c,?) will in general require new associons (h,?) to be created. According to P1' the presence of an associon (h,x) prohibits the associon (c,x) to be created. We, therefore, have to reestablish P2 --i.e. create the missing associons (h,?) -- after every single creation of a target associon (c,?) . Consequently, the program should (besides the initialization) be a repetition of

"create for one arbitrary solution of
x: [v,x] ∧ ¬[h,x] ∧ ¬[c,x] the associon (c,x) " ; (8)
"establish P2 "

As ¬[c,?] ∧ ¬[h,?] holds initially, the invariant P1' ∧ P2 is satisfied to start with, and the initialization of the repetition is empty.

Program (8) should be repeated until the solution set of the equation

x: [v,x] ∧ ¬[h,x] ∧ ¬[c,x]

is empty, thus ensuring (7), the implication that should be established, to hold.

The nonemptiness of some solution set is the criterion we introduce as "guarding boolean expression" of the repetitive construct. The way in which guarded commands constitute a repetitive construct is the same as in [6]:

<repetitive construct> ::= do <guarded command>
{[] <guarded command>} od
<guarded command> ::= <guard> → <guarded list>
<guard> ::= <equation>
<guarded list> ::= <statement list>

The repetitive construct terminates in a state in which the solution sets of all guards are empty. When initially, or upon the completed execution of a selected guarded list, the solution sets of one or more guards are nonempty, one of these guards is selected, and its guarded list is executed.

In program (8) we wish to create for one (arbitrary) solution "x" of the (nonempty) guarding equation the associon (c,x) . In order to be able to achieve this, we postulate --in analogy to [11]-- that the selection of a guarded list for execution, has as a side effect that an arbitrary element of the --then nonempty-- solution set of the guarding equation is assigned to its unknowns. These unknowns --that do now possess values-- can be used as "initialized constants" in the guarded list.

REMARK 8.2. In the indented paragraphs on pages 18 and 20 the word "constant" should be read as "constant that is not an initialized constant". According to the indented paragraph on page 20 the following program is not allowed.

```
do z1,z2: [eq,z1,z2] ∧ ¬[s,?] →
   x,y: [r,x,y] ∧ ¬[z1,x] :⇒ (z2,y)
od
```

Allowing it would bring the very problems of Chapter 4 back. (Take, e.g., {(),(r,a,b),(r,b,a),(eq,s,s)} as an initial state.)
(End of remark.)

The program that records an arbitrary clique now becomes (cf. program (8))

```
loc (h,?):
    do z: [v,z] ∧ ¬[h,z] ∧ ¬[c,z] →
       [] :⇒ (c,z);
       x,y: [v,x] ∧ y ≠ x ∧ [c,y] ∧ ¬[a,x,y] :⇒ (h,x)
    od
col .
```

Only permissible final states are possible, and each permissible final state is possible. As per execution of the guarded list only one associon (c,?) , viz. (c,z) , is created, we can simplify its second statement --"establish P2" -- into

```
x: [v,x] ∧ z ≠ x ∧ ¬[a,x,z] :⇒ (h,x) .
```

The program terminates, as per execution of the guarded list the integer expression

```
(Nx: [v,x] ∧ ¬[c,x])
```

decreases. The fact that this expression is nonnegative and finite, guaran-

tees termination in a finite number of steps. We call the expression $(N x\colon [v,x] \wedge \neg[c,x])$ a *variant function* of the repetitive construct.

According to the invariant P2 the associons (h,?) record those vertices in the complement of the clique, for which there is a vertex in the clique to which they are not adjacent. We can simplify the program by recording in the associons (h,?) the vertices that are not candidates for admittance to the clique, either because they are already recorded to be in the clique, or because there is a nonadjacent vertex in the clique, in formula:

P2' $(\forall x\colon [h,x] \equiv ([c,x] \vee ([v,x] \wedge (\exists y\colon y \neq x\colon [c,y] \wedge \neg[a,x,y])))) .$

As for all x

$$([v,x] \wedge [c,x] \wedge \neg[a,x,x]) \Rightarrow [c,x] ,$$

P2' is equivalent to

$$(\forall x\colon [h,x] \equiv ([c,x] \vee ([v,x] \wedge (\exists y\colon [c,y] \wedge \neg[a,x,y])))) .$$

Thus we obtain the program

```
loc (h,?):
    do z: [v,z] ∧ ¬[h,z] →
       [] :⇒ (c,z),(h,z);
       x: [v,x] ∧ ¬[a,x,z] :⇒ (h,x)
    od
col .
```

(The correctness of this program is formally proved in Example 9.1 (p. 73).)

REMARK 8.3. If it is given that the graph G does not contain loops, then --as [v,z] implies ¬[a,z,z] -- the program may be changed into

```
loc (h,?):
    do z: [v,z] ∧ ¬[h,z] →
       [] :⇒ (c,z);
       x: [v,x] ∧ ¬[a,x,z] :⇒ (h,x)
    od
col .
```

(End of remark.)

We shall now treat two examples that can be expressed as the generation of an arbitrary clique. They, consequently, result in analogous programs.

EXAMPLE 8.1. *Recording of an arbitrary unilateral component of a directed graph.* We first give some more nomenclature on directed graphs. A directed graph is *strongly connected*, or *strong*, if and only if every two vertices are mutually reachable. A directed graph is *unilaterally connected*, or *unilateral*, if and only if for any two vertices at least one is reachable from the other. A *strong component* of a directed graph is a strong subgraph that is not a proper subgraph of a strong subgraph. A *unilateral component* is a unilateral subgraph that is not a proper subgraph of a unilateral subgraph. Each vertex is in exactly one strong component and in at least one unilateral component.

Let a finite directed graph G be given as in Example 7.4 (p. 47). Let, furthermore, be given that ¬[u,?] holds, and let it be requested to record --in associons (u,?) -- the vertices of an arbitrary unilateral component of G .

We define a new graph G' with the same vertex set as G . The graph G' is undirected. Two vertices u and v are adjacent in G' if and only if there exists in G a nontrivial walk from u to v or vice versa. (The graph G' may, consequently, contain loops.) The cliques of G' are then precisely the unilateral components of G .

We generate the edges of G' by first establishing the truth of

(∀x,y: [w,x,y] ≡ "there exists a nontrivial walk from
vertex x to vertex y in G ") ,

as we have done in Example 7.4 (p. 47). We then establish the truth of

(∀x,y: [a,x,y] ≡ "vertices x and y are adjacent in G' ") ,

which is equivalent to

(∀x,y: [a,x,y] ≡ ([w,x,y] ∨ [w,y,x])) .

The recording of an arbitrary unilateral component of G is now the same as the recording of an arbitrary clique of G' .

Program:

```
loc (w,?,?),(a,?,?):
    x,y: [s,x,y] :⇒ (w,x,y);
    x,y,z: [w,x,y] ∧ [w,y,z] :⇒ (w,x,z);
    x,y: [w,x,y] :⇒ (a,x,y),(a,y,x);
    loc (h,?):
        do z: [v,z] ∧ ¬[h,z] →
           [] :⇒ (u,z),(h,z);
           x: [v,x] ∧ ¬[a,x,z] :⇒ (h,x)
        od
    col
col
```

(End of example.)

EXAMPLE 8.2. *One characteristic element per equivalence class.* Let a finite set V and an equivalence relation E on the elements of V be given by

$$(\forall x\colon [v,x] \equiv \text{" } x \text{ is the name of an element of } V \text{ "}) \wedge$$
$$(\forall x,y\colon [e,x,y] \equiv ([v,x] \wedge [v,y] \wedge E(x,y))) \ .$$

We are requested to write a program that selects exactly one element out of every equivalence class of E , i.e. we are requested to establish (if initially $\neg[c,?]$ holds) the truth of

$$(\forall x\colon [c,x] \equiv ([v,x] \wedge (\forall y\colon y \neq x\colon [c,y] \Rightarrow \neg[e,x,y]))) \ .$$

We can associate with V and E an undirected graph G . Each element of V corresponds (in a one-to-one fashion) to a vertex of G . Two vertices of G are adjacent if and only if their corresponding elements in V do not satisfy the relation E . (The graph G , consequently, does not contain loops.) The recording of one characteristic element per equivalence class of E is now the same as the recording of an arbitrary clique of G .

Program:

```
loc (h,?):
    do z: [v,z] ∧ ¬[h,z] →
       [] :⇒ (c,z);
       x: [e,x,z] :⇒ (h,x)
    od
col
```

(End of example.)

* *
*

In the example of the arbitrary clique we have demonstrated the need of repetition. We shall now treat an example in which the choice of the invariant forces us to introduce associons that are local to the guarded list. We shall design a program for the recording of balanced vertices in an acyclic directed graph.

For each vertex v of an acyclic directed graph there exists a (not necessarily unique) longest walk starting in v. We shall write $L(v)$ for the length of such a walk. Two vertices u and v with $L(u) = L(v)$ are called *balanced*.

Let a finite acyclic directed graph G be given as in Example 7.4 (p. 47). Let, furthermore, be given that $\neg[b,?,?]$ holds, and let it be requested to write a program that establishes the truth of

R $\quad (\forall x,y: [b,x,y] \equiv$ "vertices x and y are balanced") .

If a finite (nonempty) directed graph D is acyclic, then it contains at least one terminal vertex. (For assume that all vertices of D have at least one successor. Then we can make an arbitrary long walk in D. As the number of vertices in D is finite, a long enough walk must contain some vertex more than once. But then D contains a cycle.) Obviously, all terminal vertices are mutually balanced, and no terminal vertex is balanced with a nonterminal one. We can, consequently, record the balanced vertices of our graph G by first recording all terminal vertices to be mutually balanced, and then recording the balanced vertices in the remainder of G, which is a smaller acyclic directed graph. (A maximal path in the remainder always leads to a vertex with a terminal successor. The balanced vertices in the remainder are, consequently, exactly the nonterminal balanced vertices of G.)

We thus have the following invariant (writing $V(D)$ for the vertex set of a graph D).

P $\quad B \subset V(G) \wedge$
"all balanced vertices in B have been recorded" $\wedge$
$(\forall u,v: u,v \in V(G): (u \in B \wedge v \notin B) \Rightarrow$ " u and v not balanced")

The choice of P leads to a program of the following structure.

```
B:= ∅;
do B ≠ V(G) → "record all terminal vertices in G \ B to be
                mutually balanced, and extend B with these
                vertices"
od
```

As the graph G \ B is acyclic, it must contain a terminal vertex. Hence the cardinality of V(G) \ B decreases per execution of the guarded list.

As the balanced vertices are recorded in associons (b,?,?) , and each vertex is at least balanced with itself, we have

(∀x: [b,x,x] ≡ "vertex x in B ") and
(∀x: ([v,x] ∧ ¬[b,x,x]) ≡ "vertex x in V(G) \ B ") .

Our program will thus have the following structure.

```
do z: [v,z] ∧ ¬[b,z,z] → "create (b,x,y) for all terminal
                           vertices x and y in the graph
                           G \ B "
od
```

The terminal vertices in the graph G \ B constitute a subset of V(G) \ B , viz. those vertices for which all successors in G are in B . We record these vertices in associons (c,?) , i.e. we establish the truth of

(∀x: [c,x] ≡ ([v,x] ∧ ¬[b,x,x] ∧ (∀y: [s,x,y] ⇒ [b,y,y]))) . (9)

This equivalence can also be written as

(∀x: [c,x] ≡ ([v,x] ∧ ¬[b,x,x] ∧ ¬(∃y: [s,x,y] ∧ ¬[b,y,y]))) .

We record the quantified condition in associons (h,?) , i.e. we establish the truth of

(∀x: [h,x] ≡ (∃y: [s,x,y] ∧ ¬[b,y,y])) . (10)

Assuming ¬[h,?] to hold initially, the truth of (10) can be established by the closure statement

x,y: [s,x,y] ∧ ¬[b,y,y] :⇒ (h,x) ,

after which (9) can be made true by the closure statement (assuming ¬[c,?] to hold initially)

x: [v,x] ∧ ¬[b,x,x] ∧ ¬[h,x] :⇒ (c,x) .

Program:

```
do z: [v,z] ∧ ¬[b,z,z] →
   loc (h,?),(c,?):
       x,y: [s,x,y] ∧ ¬[b,y,y] :⇒ (h,x);
       x: [v,x] ∧ ¬[b,x,x] ∧ ¬[h,x] :⇒ (c,x);
       x,y: [c,x] ∧ [c,y] :⇒ (b,x,y)
   col
od
```

The integer expression (Nz: [v,z] ∧ ¬[b,z,z]) is a variant function of the repetition.

As the repetitive construct contains only one guard, and as its initialized constant z does not occur in the guarded list, the above program is deterministic. In Chapter 11 we shall show that it is also possible to write a nonrepetitive version.

In (9) and (10) the target associons (b,?,?) negatively match the right-hand side. Given this choice of (9) and (10), we had to declare the associons (c,?) and (h,?) local to the guarded list. Logically, the introduction of local associons is not necessary. The associons (c,?) were introduced to avoid "negative matching" in the statement catering for the creation of the associons (b,?,?) . Maintaining as an invariant

(∀x: [bb,x] ≡ [b,x,x]) ,

their introduction can be avoided.

The relation "balanced" is an equivalence relation. Per execution of the guarded list one equivalence class is recorded. If we extend the associons (h,?) with an extra member, representing a characteristic element of the "current" equivalence class, then the (resulting) associons (h,?,?) can be declared global to the repetitive construct. In order to be able to determine this characteristic element, we maintain as an invariant

(∀x: [bt,x] ≡ ([bb,x] ∨ ([v,x] ∧ (∀y: [s,x,y] ⇒ [bb,y])))) .

(In words: [bt,x] holds if and only if vertex x is in B or vertex x is a terminal vertex of G \ B .)

```
loc (bb,?),(bt,?),(h,?,?):
    x: [v,x] ∧ ¬[s,x,?] :⇒ (bt,x);
    do z: [bt,z] ∧ ¬[bb,z] →
       x,y: [bt,x] ∧ ¬[bb,x] ∧ [bt,y] ∧ ¬[bb,y] :⇒ (b,x,y);
       x: [bt,x] :⇒ (bb,x);
       x,y: [s,x,y] ∧ ¬[bb,y] :⇒ (h,x,z);
       x: [v,x] ∧ ¬[h,x,z] :⇒ (bt,x)
    od
col
```

For strategical reasons this second program is inferior to the first. The extra information that is kept --the (h,x,y) with $y \in B$ -- is left unused.

CHAPTER 9

FORMAL DEFINITION OF THE REPETITIVE CONSTRUCT

The syntax of the repetitive construct has been defined in the preceding chapter. Its semantics are defined as follows.

Let S denote the repetitive construct

$$\underline{\text{do}}\ E_0 \rightarrow S_0\ []\ldots[]\ E_{n-1} \rightarrow S_{n-1}\ \underline{\text{od}}\ .$$

Let BB denote the predicate $(\exists i\colon 0 \leq i < n\colon (N\ E_i) > 0)$, and let B_i denote the equation E_i, of which the list of unknowns has been dropped. (B_i expresses the relation which the initialized constants satisfy.)

Then, for any predicate R, by definition

$$wp(S,R) \equiv (\exists k\colon k \geq 0\colon H_k(R))\ ,$$

in which the predicates $H_k(R)$ are defined by

$$H_0(R) \equiv R \wedge \neg BB\ ,$$
$$H_{k+1}(R) \equiv H_0(R) \vee (BB \wedge (\forall i\colon 0 \leq i < n\colon B_i \Rightarrow wp(S_i,H_k(R))))\ .$$

The predicate $H_k(R)$ expresses the condition that the construct terminates after at most k executions of a guarded list, in a state satisfying R. The definition of $wp(S,R)$ expresses that there must exist a nonnegative integer k, such that the construct terminates after at most k executions of a guarded list, in a state satisfying R.

The following theorem expresses that any closure statement can be written as a repetitive construct, in which each guarded list consist of one closure statement with [] as its left-hand side and one target associon (format) as its right-hand side. It shows that an implementation of the closure statement may create the missing associons of the closure one by one, looking (in an arbitrary order) for solutions of the left-hand side for which a target associon is absent.

THEOREM 9.1. Let S denote the closure statement

$$x\colon E(x) :\Rightarrow A_0(x),\ldots,A_{\ell-1}(x)\ ,$$

Let S_i $(0 \leq i < \ell)$ denote the closure statement

$$[] :\Rightarrow A_i(x) \ ,$$

and let S' denote the repetitive construct

$$\begin{array}{l} \underline{\text{do}}\ x\colon E(x) \wedge \neg\tilde{A}_0(x) \rightarrow S_0 \\ [] \ \ldots \\ [] \ x\colon E(x) \wedge \neg\tilde{A}_{\ell-1}(x) \rightarrow S_{\ell-1} \\ \underline{\text{od}} \ . \end{array}$$

Then, for all R , $wp(S,R) \equiv wp(S',R)$.

PROOF. Let NC denote

$$(N\ A\colon wp(S,\tilde{A}) \wedge \neg\tilde{A}) \ ,$$

i.e. the number of associons A satisfying $wp(S,\tilde{A}) \wedge \neg\tilde{A}$. According to Theorem 6.1 (p. 35) NC is finite for all states. Obviously, we also have $NC \geq 0$.

We first prove the following properties.

1) $NC = 0$ if and only if $\neg BB$,

2) $E(x)$ implies for all R

$$wp(S_i,wp(S,R)) \equiv wp(S,R) \ ,$$

3) $E(x) \wedge \neg\tilde{A}_i(x)$ implies for all k $(k \geq 0)$

$$wp(S_i,NC = k) \equiv (NC = k+1) \ .$$

Property 2) expresses that the (permitted) execution of the statement S_i leaves the condition $wp(S,R)$ invariant. Property 3) expresses that the (permitted) execution of S_i decreases the function NC by one.

Proof of 1):

$$\begin{array}{l} \neg BB \equiv \neg(\exists i\colon 0 \leq i < \ell\colon (N x\colon E(x) \wedge \neg\tilde{A}_i(x)) > 0) \\ \quad \equiv (\forall i\colon 0 \leq i < \ell\colon (N x\colon E(x) \wedge \neg\tilde{A}_i(x)) = 0) \\ \quad \equiv (\forall i\colon 0 \leq i < \ell\colon (\forall x\colon E(x) \Rightarrow \tilde{A}_i(x))) \equiv \hat{S} \ . \end{array}$$

We prove $NC = 0$ if and only if $\hat{S}$.

We first assume $NC = 0$ and derive $\hat{S}$. Let i be such that $0 \leq i < \ell$, and let x be such that $E(x)$ is satisfied. We prove $\tilde{A}_i(x)$. From $E(x)$ we conclude, by applying Property 6.4 (p. 36),

$$wp(S,E(x)) \ . \tag{1}$$

From $wp(S,\hat{S})$ --Property 6.5 (p. 37)-- and the definition of $\hat{S}$ we conclude

$$wp(S,E(x) \Rightarrow \tilde{A}_i(x)) \ . \tag{2}$$

From (1) and (2) follows (apply Property 1.2 (p. 3))

$$wp(S,\tilde{A}_i(x)) \ . \tag{3}$$

As $NC = 0$ we have for all associons A

$$wp(S,\tilde{A}) \Rightarrow \tilde{A} \ .$$

In particular do we have

$$wp(S,\tilde{A}_i(x)) \Rightarrow \tilde{A}_i(x) \ . \tag{4}$$

From (3) and (4) we conclude $\tilde{A}_i(x)$.

The fact that $\hat{S}$ implies $NC = 0$ is a direct consequence of Property 6.6 (p. 37).

Proof of 2):

We assume $E(x)$. Then (3) holds. Let A denote an arbitrary associon. As S_i is a noncascading closure statement, we may apply Property 6.8 (p. 38), yielding

$$wp(S_i,\tilde{A}) \equiv (\tilde{A} \vee A = A_i(x)) \ . \tag{5}$$

From Property 6.2 (p. 36) we know

$$\tilde{A} \Rightarrow wp(S,\tilde{A}) \ . \tag{6}$$

From (3) and (6) we conclude

$$(\tilde{A} \vee A = A_i(x)) \Rightarrow wp(S,\tilde{A}) \ . \tag{7}$$

From (5) and (7) follows

$$wp(S_i,\tilde{A}) \Rightarrow wp(S,\tilde{A}) \ .$$

From Property 6.9 (p. 38) we then know

$$wp(S_i,wp(S,R)) \equiv wp(S,R) \ .$$

Proof of 3):

We assume $E(x) \wedge \neg\tilde{A}_i(x) \wedge k \geq 0$. Then, on account of $E(x)$, (3) holds. By definition,

$$wp(S_i,NC = k) \equiv wp(S_i,(N\ A:\ wp(S,\tilde{A}) \wedge \neg\tilde{A}) = k) \ ,$$

or

$$wp(S_i,NC = k) \equiv ((N\ A:\ wp(S_i,wp(S,\tilde{A}) \wedge \neg\tilde{A})) = k) \ ,$$

or (apply Properties 1.2 (p. 3) and 6.1 (p. 35))

$$wp(S_i,NC = k) \equiv ((N\ A:\ wp(S_i,wp(S,\tilde{A})) \wedge \neg wp(S_i,\tilde{A})) = k) \ .$$

As $E(x)$ holds, we may apply Property 2) yielding

$$wp(S_i,NC = k) \equiv ((N\ A:\ wp(S,\tilde{A}) \wedge \neg wp(S_i,\tilde{A})) = k) \ . \qquad (8)$$

From (5) and (8) we conclude

$$wp(S_i,NC = k) \equiv ((N\ A:\ wp(S,\tilde{A}) \wedge \neg\tilde{A} \wedge A \neq A_i(x)) = k) \ ,$$

or

$$wp(S_i,NC = k) \equiv ((N\ A:\ wp(S,\tilde{A}) \wedge \neg\tilde{A}) - (N\ A:\ wp(S,\tilde{A}) \wedge \neg\tilde{A} \wedge A = A_i(x)) = k) \ . \qquad (9)$$

From (3) and $\neg\tilde{A}_i(x)$ follows

$$(N\ A:\ wp(S,\tilde{A}) \wedge \neg\tilde{A} \wedge A = A_i(x)) = 1 \ . \qquad (10)$$

From (9), (10), and the definition of NC we conclude

$$wp(S_i,NC = k) \equiv (NC = k + 1) \ .$$

(End of proof of Properties 1), 2), and 3).)

Next we shall prove for arbitrary R and k $(k \geq 0)$

4) $H_k(R) \equiv (wp(S,R) \wedge NC \leq k)$.

We then have

$$wp(S,R) \Rightarrow H_{NC}(R) \qquad (11)$$

and

$$H_k(R) \Rightarrow wp(S,R) \ . \qquad (12)$$

The combination of (11) and (12) allows us to conclude

$$wp(S,R) \equiv (\exists k: k \geq 0: H_k(R))$$
$$\equiv wp(S',R) \ .$$

Proof of 4):

We prove 4) by mathematical induction. By definition,

$$H_0(R) \equiv (R \wedge \neg BB) \ .$$

$\hat{S}$ and $\neg BB$ are equivalent. According to Property 6.6 (p. 37) $\hat{S}$ implies $wp(S,R) \equiv R$. From Property 1) and $NC \geq 0$ we conclude $\neg BB \equiv (NC \leq 0)$. We, consequently, have

$$H_0(R) \equiv (wp(S,R) \wedge NC \leq 0) \ . \tag{13}$$

We assume as induction hypothesis

$$H_k(R) \equiv (wp(S,R) \wedge NC \leq k) \tag{14}$$

and derive

$$H_{k+1}(R) \equiv (wp(S,R) \wedge NC \leq k + 1) \ .$$

By definition,

$$H_{k+1}(R) \equiv H_0(R) \vee (BB \wedge (\forall i: 0 \leq i < \ell: B_i \Rightarrow wp(S_i,H_k(R)))) . \tag{15}$$

By substituing (14) in the universally quantified term, this term becomes

$$(\forall i: 0 \leq i < \ell: B_i \Rightarrow wp(S_i,wp(S,R) \wedge NC \leq k)) \ ,$$

or (apply Property 1.2 (p. 3) and the definition of B_i)

$$(\forall i: 0 \leq i < \ell: (E(x) \wedge \neg \tilde{A}_i(x)) \Rightarrow (wp(S_i,wp(S,R)) \wedge wp(S_i,NC \leq k))) \ .$$

By applying Properties 2) and 3) this becomes

$$(\forall i: 0 \leq i < \ell: B_i \Rightarrow (wp(S,R) \wedge NC \leq k + 1)) \ ,$$

or

$$BB \Rightarrow (wp(S,R) \wedge NC \leq k + 1) \ . \tag{16}$$

Resubstituting (16) in (15) we obtain

$$H_{k+1}(R) \equiv (H_0(R) \vee (BB \wedge wp(S,R) \wedge NC \leq k + 1)) \ ,$$

or (use (13))

$$H_{k+1}(R) \equiv ((wp(S,R) \wedge NC = 0) \vee (BB \wedge wp(S,R) \wedge NC \leq k + 1))$$

$$\equiv ((wp(S,R) \wedge NC \leq k + 1) \wedge (BB \vee NC = 0)) ,$$

or (use Property 1))

$$H_{k+1}(R) \equiv ((wp(S,R) \wedge NC \leq k + 1) \wedge (BB \vee \neg BB))$$

$$\equiv ((wp(S,R) \wedge NC \leq k + 1) .$$

(End of proof.)

In the proof of the above theorem we have shown that the execution of a guarded list S_i decreases the nonnegative integer function NC under invariance of wp(S,R) . The final state satisfies $\neg BB$, or $\hat{S}$. If the initial state satisfies wp(S,R) , then the final state satisfies $wp(S,R) \wedge \hat{S}$, and hence (Property 6.6 (p. 37)) R . This is the general pattern for the construction of repetitive programs: we look for both an invariant relation P , such that $P \wedge \neg BB$ implies the relation to be established, and a (nonnegative) integer *variant function* t of the state, whose value decreases per execution of a guarded list.

In order to characterize all initial states such that a mechanism will decrease the variant function, we employ the notion *wdec*, which is defined as follows. (Although this definition differs from the one given in [6], it defines the same concept.)

> Let S denote some mechanism, and t an integer function of the state, then by definition
>
> $$wdec(S,t) \equiv wp(S, t < t_0)^{t_0}_{t} .$$

(For notation P^{x}_{y} , see Example 1.1 (p. 4).) The wdec(S,t) characterizes those states for which the execution of S establishes $t < t_0$, with t_0 being the initial value of t .

The following theorem is due to Hoare [10], who called it the "Rule of Iteration".

THEOREM 9.2. *Invariance theorem for repetitive constructs.*
With S , B_i , and BB as defined in the beginning of this chapter, there holds that, if for all i $(0 \leq i < n)$

$$(P \wedge B_i) \Rightarrow (wp(S_i,P) \wedge wdec(S_i,t) \wedge t \geq 0)$$

then

$$P \Rightarrow wp(S, P \wedge \neg BB) \ .$$

The above formulation has been taken from [6], which also contains a proof of the theorem. The theorem expresses that if the (permitted) execution of any guarded list leaves the truth of P invariant and decreases the nonnegative integer function t, then the execution of the repetitive construct, in a state satisfying P, will lead to termination in a state satisfying $P \wedge \neg BB$.

EXAMPLE 9.1. In this example we give a formal proof of the correctness of the program, presented in Chapter 8, for the recording of an arbitrary clique:

```
loc (h,?):
    do z: [v,z] ∧ ¬[h,z] →
       [] :⇒ (c,z),(h,z);
       x: [v,x] ∧ ¬[a,x,z] :⇒ (h,x)
    od
col
```

Let $A(u,w)$ denote

$$(\forall y: y \neq u: [c,y] \Rightarrow [a,w,y]) \ .$$

Then R, $P1$, and $P2$ can be written as follows. (Compare R, $P1$ and $P2'$ on pages 56 and 60.)

R $\quad (\forall x: [c,x] \equiv ([v,x] \wedge A(x,x)))$

P1 $\quad (\forall x: [c,x] \Rightarrow ([v,x] \wedge A(x,x)))$

P2 $\quad (\forall x: [h,x] \equiv ([c,x] \vee ([v,x] \wedge \neg A(x,x))))$

Let S denote the repetitive construct of the program. We prove $(\neg[c,?] \wedge \neg[h,?]) \Rightarrow wp(S,R)$. With the following notational conventions

S1 $\quad [] :\Rightarrow (c,z),(h,z)$,

S2 $\quad x: [v,x] \wedge \neg[a,x,z] :\Rightarrow (h,x)$,

B $\quad [v,z] \wedge \neg[h,z]$,

Pi $\quad (\forall x: [h,x] \equiv ([c,x] \vee ([v,x] \wedge \neg A(z,x))))$,

we first prove

$$(P1 \wedge P2 \wedge B) \Rightarrow wp(S1,P1 \wedge Pi \wedge [c,z]) \ , \tag{17}$$

and

$$(P1 \wedge Pi \wedge [c,z]) \Rightarrow wp(S2,P1 \wedge P2) \ . \tag{18}$$

Then (apply Property 1.4 (p. 3) and the definition of the semicolon)

$$(P1 \wedge P2 \wedge B) \Rightarrow wp("S1;S2",P1 \wedge P2) \ .$$

(Pi is used to characterize the state at the semicolon.) We first formulate some relations between $A(z,x)$ and $A(x,x)$. For all x

$$(\neg A(z,x) \wedge \neg[c,x]) \Rightarrow \neg A(x,x) \ , \tag{19}$$

$$(\neg A(x,x) \wedge \neg[c,z]) \Rightarrow \neg A(z,x) \ , \tag{20}$$

$$(A(z,x) \wedge [c,z] \wedge [a,x,z]) \Rightarrow A(x,x) \ . \tag{21}$$

For all x $(x \neq z)$

$$([c,z] \wedge \neg[a,x,z]) \Rightarrow \neg A(x,x) \ . \tag{22}$$

Proof of (17):

By applying Property 6.8 (p. 38) we obtain for all x

$$wp(S1,A(z,x)) \equiv A(z,x) \ , \tag{23}$$

and for all x $(x \neq z)$

$$wp(S1,A(x,x)) \equiv A(x,x) \wedge [a,x,z] \ . \tag{24}$$

We assume $P1 \wedge P2 \wedge B$ and derive $wp(S1,P1)$, $wp(S1,Pi)$, and $wp(S1,[c,z])$. From $P2 \wedge B$ we conclude

$$[v,z] \wedge \neg[c,z] \wedge A(z,z) \ . \tag{25}$$

We write $P1$ as follows.

$$(\forall x: x \neq z: [c,x] \Rightarrow ([v,x] \wedge A(x,x))) \\ \wedge [c,z] \Rightarrow ([v,z] \wedge A(z,z))$$

By applying Property 6.8 (p. 38), (23), and (24) we obtain

$$wp(S1,P1) \equiv (\forall x: x \neq z: [c,x] \Rightarrow ([v,x] \wedge A(x,x) \wedge [a,x,z])) \\ \wedge [v,z] \wedge A(z,z) \ ,$$

which, on account of (25), is implied by $P1$. We write Pi as follows.

$$(\forall x: x \neq z: [h,x] \equiv ([c,x] \vee ([v,x] \wedge \neg A(z,x)))) \\ \wedge [h,z] \equiv ([c,z] \vee ([v,z] \wedge \neg A(z,z)))$$

By applying Property 6.8 (p. 38) and (23) we obtain

$$wp(S1,Pi) \equiv (\forall x\colon x \neq z\colon [h,x] \equiv ([c,x] \vee ([v,x] \wedge \neg A(z,x)))) \ ,$$

which, on account of (19) and (20), is implied by $P2 \wedge \neg[c,z]$. We, furthermore, obtain $wp(S1,[c,z]) \equiv [] $.

Proof of (18):

We assume $P1 \wedge Pi \wedge [c,z]$ and derive $wp(S2,Pi)$ and $wp(S2,P2)$. From $Pi \wedge [c,z]$ we conclude $[h,z]$. From Property 6.3 (p. 36) we know $wp(S2,P1) \equiv P1$, which leaves only $wp(S2,P2)$ to be proved.

By applying Property 6.8 (p. 38) we obtain for $wp(S2,P2)$

$$(\forall x\colon ([h,x] \vee ([v,x] \wedge \neg[a,x,z])) \equiv ([c,x] \vee ([v,x] \wedge \neg A(x,x)))) \ . \qquad (26)$$

For $x = z$ (26) is implied by $[h,z] \wedge [c,z]$. Suppose $x \neq z$. From (19) and Pi we then conclude

$$[h,x] \Rightarrow ([c,x] \vee ([v,x] \wedge \neg A(x,x))) \qquad (27)$$

From (22) and $[c,z]$ we conclude

$$([v,x] \wedge \neg[a,x,z]) \Rightarrow ([v,x] \wedge \neg A(x,x)) \ . \qquad (28)$$

From Pi we conclude

$$[c,x] \Rightarrow [h,x] \ . \qquad (29)$$

From (21) and $[c,z]$ we conclude

$$([v,x] \wedge \neg A(x,x)) \Rightarrow ([v,x] \wedge (\neg A(z,x) \vee \neg[a,x,z])) \ ,$$

and, hence, (use Pi)

$$([v,x] \wedge \neg A(x,x)) \Rightarrow ([h,x] \vee ([v,x] \wedge \neg[a,x,z])) \ . \qquad (30)$$

From (27), (28), (29), and (30) follows (26).

As the variant function t we choose $(N x\colon [v,x] \wedge \neg[c,x])$. Then, obviously, $t \geq 0$. Using

$$t = (N x\colon x \neq z\colon [v,x] \wedge \neg[c,x]) + (N\ [v,z] \wedge \neg[c,z]) \ ,$$

we derive

$$\begin{aligned} wdec("S1;S2",t) &\equiv wp("S1;S2",t < t_0)_t^{t_0} \\ &\equiv ((N x\colon x \neq z\colon [v,x] \wedge \neg[c,x]) < t_0)_t^{t_0} \\ &\equiv (N x\colon x \neq z\colon [v,x] \wedge \neg[c,x]) < t \end{aligned}$$

$$\equiv [v,z] \wedge \neg[c,z] \ ,$$

which is implied by (25).

We conclude that $P1 \wedge P2 \wedge B$ implies

$$wp("S1;S2",P1 \wedge P2) \wedge wdec("S1;S2",t) \wedge t \geq 0 \ .$$

From Theorem 9.2 (p. 72) we then know

$$(P1 \wedge P2) \Rightarrow wp(S,P1 \wedge P2 \wedge \neg BB) \ . \qquad (31)$$

$\neg BB$ $\quad (\forall x\colon [v,x] \Rightarrow [h,x])$

We assume $\neg[c,?] \wedge \neg[h,?]$ and derive $wp(S,R)$. From $\neg[c,?] \wedge \neg[h,?]$ we conclude $P1 \wedge P2$, and, on account of (31), $wp(S,P1 \wedge P2 \wedge \neg BB)$. As

$$(P2 \wedge \neg BB) \Rightarrow (\forall x\colon ([v,x] \wedge A(x,x)) \Rightarrow [c,x]) \ ,$$

we then have (use $P1$ and R) $wp(S,R)$.
(End of example.)

CHAPTER 10

SOME EXAMPLES

EXAMPLE 10.1. *Selection of distinct subsets.*

A finite set V and a finite "family" W of (not necessarily mutually distinct) subsets of V are given (Distinctly named subsets of V may be otherwise equal.). We are requested to write a program that makes a selection out of the subsets in W , such that for each subset S in W there is exactly one selected subset that contains the same elements as S . The initial state satisfies

$$(\forall x\colon [v,x] \equiv \text{" } x \text{ is the name of an element of } V \text{ "}) \wedge$$
$$(\forall x\colon [w,x] \equiv \text{" } x \text{ is the name of an element of } W \text{ "}) \wedge$$
$$(\forall x,y\colon [m,x,y] \equiv ([w,x] \wedge [v,y] \wedge \text{"element } y \text{ in subset } x \text{"}))\ .$$

Let, furthermore, be given that $\neg[c,?]$ holds. We are requested to write a program that establishes the truth of

$$R \qquad (\forall x\colon [c,x] \equiv ([w,x] \wedge (\forall y\colon y \neq x\colon [c,y] \Rightarrow \text{"subsets } x \text{ and } y \text{ differ"})))\ .$$

We record in associons (d,?,?) names of mutually distinct subsets in W , or --to be more precise-- we establish the truth of

$$(\forall x,y\colon [d,x,y] \equiv ((\exists z\colon [m,x,z] \wedge [w,y] \wedge \neg[m,y,z]) \vee (\exists z\colon [m,y,z] \wedge [w,x] \wedge \neg[m,x,z])))\ . \qquad (1)$$

The relation R can then be written as

$$(\forall x\colon [c,x] \equiv ([w,x] \wedge (\forall y\colon y \neq x\colon [c,y] \Rightarrow [d,x,y])))\ .$$

But that is exactly the relation that was to be established in the example of the arbitrary clique (Chapter 8). As we also have for all x $\neg[d,x,x]$, we can employ the program of Remark 8.3 (p. 60):

```
loc (h,?),(d,?,?):
    "establish (1)";
    do z: [w,z] ∧ ¬[h,z] →
       [] :⇒ (c,z);
       x: [w,x] ∧ ¬[d,x,z] :⇒ (h,x)
    od
col
```

We still have to implement "establish (1)". The closure statement

S x,y,z: [m,x,z] ∧ [w,y] ∧ ¬[m,y,z] :⇒ (d,x,y),(d,y,x)

establishes the truth of

$\hat{S}$ (∀x,y,z: ([m,x,z] ∧ [w,y] ∧ ¬[m,y,z]) ⇒ ([d,x,y] ∧ [d,y,x])) .

$\hat{S}$ can be written as

(∀x,y: (∃z: [m,x,z] ∧ [w,y] ∧ ¬[m,y,z]) ⇒ ([d,x,y] ∧ [d,y,x])) ,

which is equivalent to

(∀x,y: ((∃z: [m,x,z] ∧ [w,y] ∧ ¬[m,y,z]) ∨
(∃z: [m,y,z] ∧ [w,x] ∧ ¬[m,x,z])) ⇒ [d,x,y]) . (2)

By applying Theorem 6.3 (p. 39) we obtain that S maintains the invariance of

(∀x,y: [d,x,y] ⇒ ((∃z: [m,x,z] ∧ [w,y] ∧ ¬[m,y,z]) ∨
(∃z: [m,y,z] ∧ [w,x] ∧ ¬[m,x,z]))) . (3)

On account of ¬[d,?,?] (3) is satisfied to start with. S , therefore, establishes a state satisfying (2) and (3), and thus (1).

EXAMPLE 10.2. *Recording equivalence classes.*

Let a finite set V and an equivalence relation E on the elements of V be given as in Example 8.2 (p. 62). Let, furthermore, be given that ¬[c,?] ∧ ¬[m,?,?] holds. We are requested to write a program that records the equivalence classes of E . It should do so by selecting one characteristic element per equivalence class (as in Example 8.2 (p. 62)), and recording for each element of V the characteristic element of its equivalence class. More precisely: it should, by creating associons (c,?) and (m,?,?) , establish the truth of R1 ∧ R2 :

R1 (∀x: [c,x] ≡ ([v,x] ∧ (∀y: y ≠ x: [c,y] ⇒ ¬[e,x,y]))) ,

R2 (∀x,y: [m,x,y] ≡ ([c,x] ∧ [e,x,y])) .

Program (cf. Example 8.2 (p. 62)):

```
do z: [v,z] ∧ ¬[m,?,z] →
     [] :⇒ (c,z);
     x: [e,x,z] :⇒ (m,z,x)
od
```

REMARK 10.1. If we take for V the set of vertices of a finite directed graph, and for E the equivalence relation "mutually reachable", then we obtain a program for the recording of the strong components of a directed graph. For a finite directed graph that is given as in Example 7.4 (p. 47), we then obtain the following program.

```
loc (e,?,?):
    loc (w,?,?):
        x,y: [s,x,y] :⇒ (w,x,y);
        x,y,z: [w,x,y] ∧ [w,y,z] :⇒ (w,x,z);
        x,y: [w,x,y] ∧ [w,y,x] :⇒ (e,x,y);
        x: [v,x] :⇒ (e,x,x)
    col;
    do z: [v,z] ∧ ¬[m,?,z] →
       [] :⇒ (c,z);
       x: [e,x,z] :⇒ (m,z,x)
    od
col
```

(End of remark.)

EXAMPLE 10.3. *Arbitrary spanning tree.*

A finite (directed) graph without arcs is given by

(∀x: [v,x] ≡ " x is the name of a vertex") .

We are requested to write a program that generates arcs between the vertices, such that the resulting directed graph constitutes a spanning out-tree. We establish the truth of

(∀x,y: [q,x,y] ≡ "the out-tree has an arc from vertex x to vertex y ") ,

while initially ¬[q,?,?] holds.

We first select an arbitrary vertex as root. Then we extend the out-tree until it spans the whole graph. In order to be able to characterize the vertices in the tree, we maintain

(∀x: [h,x] ≡ "vertex x in the tree") ,

or

(∀x: [h,x] ≡ (" x is the name of the root" ∨ [q,?,x])) .

We accomplish the selection of the root by means of a repetitive construct, whose guarded list is executed as long as $\neg[h,?]$ holds, i.e. exactly once. As guard we choose

$$x: [v,x] \wedge \neg[h,?] \ .$$

Upon termination we then know that

$$(\forall x: [v,x] \Rightarrow [h,?]) \ ,$$

or

$$(\exists x: [v,x]) \Rightarrow [h,?] \ .$$

$$\underline{\text{do}}\ x: [v,x] \wedge \neg[h,?] \rightarrow [] :\Rightarrow (h,x)\ \underline{\text{od}}$$

REMARK 10.2. The above repetitive construct resembles the pseudo-repetition that occurs in some programs in [6]. There the alternative construct

$$\underline{\text{if}}\ B \rightarrow S\ []\ \neg B \rightarrow \text{skip}\ \ \underline{\text{fi}}$$

is in the case that $B \Rightarrow wp(S,\neg B)$ replaced by the pseudo-loop

$$\underline{\text{do}}\ B \rightarrow S\ \underline{\text{od}} \ .$$

(End of remark.)

An arc from vertex x to vertex y may be laid if x is in the tree, and y is not.

Program:

$$\begin{array}{l}\underline{\text{loc}}\ (h,?): \\ \quad \underline{\text{do}}\ x: [v,x] \wedge \neg[h,?] \rightarrow [] :\Rightarrow (h,x)\ \underline{\text{od}}; \\ \quad \underline{\text{do}}\ x,y: [h,x] \wedge [v,y] \wedge \neg[h,y] \rightarrow [] :\Rightarrow (q,x,y),(h,y)\ \underline{\text{od}} \\ \underline{\text{col}}\end{array}$$

The integer expression $(Nx: [v,x] \wedge \neg[h,x])$ is a variant function for the second repetitive construct. Only permissible final states are possible, and each permissible final state is possible.

REMARK 10.3. The generation of an arbitrary sequence, in which [q,x,y] denotes y to be the successor of x in the sequence, is equivalent to the generation of an arbitrary spanning tree in which each vertex has at most one outgoing arc. This implies that by adding the factor "$\neg[q,x,?]$" to the guard of the second repetitive construct, we obtain a program for the generation of an arbitrary sequence.
(End of remark.)

EXAMPLE 10.4. *Arbitrary order-preserving sequence.*

The vertices of a finite acyclic directed graph can be ordered into a sequence, in such a way that for any arc from a vertex u to a vertex v , u has a smaller ordinal number in the sequence than v .

A finite acyclic directed graph G is given as in Example 7.4 (p. 47). Let, furthermore, be given that ¬[q,?,?] holds. We are requested to write a program that establishes the truth of

R (∀x,y: [q,x,y] ≡ "vertex y is the successor of vertex x in the sequence") .

We construct the sequence in the order of increasing ordinal number. A vertex may be chosen as the next element of the sequence if (it is not already in the sequence, and) all its predecessors (in G) are in the sequence. As in Example 10.3 (p. 79) we record the names of the vertices in the sequence in associons (h,?) , i.e. we maintain the invariant

P1 (∀x: [h,x] ≡ (" x is the name of the first element" ∨ [q,?,x])).

We record in associons (c,?) the names of the vertices for which all predecessors are in the sequence, i.e. we also maintain the invariant

P2 (∀x: [c,x] ≡ ([v,x] ∧ (∀y: [s,y,x] ⇒ [h,y]))) .

The equivalence P2 can also be written as

(∀x: [c,x] ≡ ([v,x] ∧ ¬(∃y: [s,y,x] ∧ ¬[h,y]))) .

In order to be able to (re)establish the truth of P2 in the guarded list, we record the quantified condition in associons (d,?) , i.e. we establish the truth of

(∀x: [d,x] ≡ (∃y: [s,y,x] ∧ ¬[h,y])) . (4)

If (4) holds, P2 can be written as

(∀x: [c,x] ≡ ([v,x] ∧ ¬[d,x])) .

As in (4) the presence condition of the global associons (h,?) is negated, the associons (d,?) have to be declared local to the guarded list.

Program (comments between braces "{" and "}"):

```
loc (c,?),(h,?):
    x: [v,x] ∧ ¬[s,?,x] :⇒ (c,x) {P2};
    do x: [c,x] ∧ ¬[h,?] → [] :⇒ (h,x) od {P1};
    do z1,z2: [h,z1] ∧ ¬[q,z1,?] ∧ [c,z2] ∧ ¬[h,z2] →
       loc (d,?):
           [] :⇒ (q,z1,z2),(h,z2) {P1};
           x,y: [s,y,x] ∧ ¬[h,y] :⇒ (d,x) {(4)};
           x: [v,x] ∧ ¬[d,x] :⇒ (c,x) {P2}
       col
    od
col
```

The expression $(N x: [v,x] \wedge \neg[h,x])$ is a variant function for the second repetitive construct. Only permissible final states are possible, and each permissible final state is possible.

REMARK 10.4. For a graph G that does not have any arcs, we continually have

$$\neg[d,?] \wedge (\forall x: [c,x] \equiv [v,x]) \ .$$

The above program then reduces to

```
loc (h,?):
    do x: [v,x] ∧ ¬[h,?] → [] :⇒ (h,x) od;
    do z1,z2: [h,z1] ∧ ¬[q,z1,?] ∧ [v,z2] ∧ ¬[h,z2] →
              [] :⇒ (q,z1,z2),(h,z2)
    od
col .
```

This is exactly the program of Remark 10.3 (p. 80) for the generation of an arbitrary sequence.
(End of remark.)

EXAMPLE 10.5. *Most distant leaves of an out-tree.*

A finite out-tree T is given by

$(\forall x: [v,x] \equiv$ " x is the name of a vertex of T ") $\wedge$
$(\forall x,y: [s,x,y] \equiv$ " T has an arc from vertex x to vertex y ").

We define an integer function P on the vertices of T , with P(x) being the length of the path from the root to vertex x . In terms of the representation of T :

$([v,x] \wedge \neg[s,?,x]) \Rightarrow P(x) = 0$,

$[s,y,x] \Rightarrow P(x) = P(y) + 1$.

We have to design a program that records the vertices with maximal path length from the root, i.e. a program that, if initially $\neg[d,?]$ holds, establishes the truth of

R $\quad (\forall x\colon [d,x] \equiv ([v,x] \wedge (\forall y\colon [v,y] \Rightarrow P(x) \geq P(y))))$.

We have to record all vertices x of T for which $(\forall y\colon [v,y] \Rightarrow P(x) \geq P(y))$. We shall record these vertices for a subtree U , and we shall extend U until it equals T . The vertices of U are recorded in associons $(u,?)$. We maintain the invariance of

P1 $\quad (\forall x\colon [u,x] \Rightarrow [v,x])$.

We have to characterize the most distant leaves of U . We do, however, not record the most distant leaves themselves, but instead of that, we record --in associons $(i,?)$-- those vertices of U that are <u>not</u> the most distant ones, as this last property is a monotonic one during the growth of U , i.e. we also maintain as an invariant

P2 $\quad (\forall x\colon [i,x] \equiv ([u,x] \wedge (\exists y\colon [u,y] \wedge P(x) < P(y))))$.

U initially contains the root only. We proceed to extend U until

$(\forall x\colon [v,x] \Rightarrow [u,x])$.

As a variant function we choose $(N x\colon [v,x] \wedge \neg[i,x])$.

Program:

```
loc (u,?),(i,?):
    x: [v,x] ∧ ¬[s,?,x] :⇒ (u,x);
    do z: [v,z] ∧ ¬[u,z] → x: [u,x] :⇒ (i,x);
                           x,y: [i,x] ∧ [s,x,y] :⇒ (u,y)
    od;
    x: [v,x] ∧ ¬[i,x] :⇒ (d,x)
col
```

Upon termination of the repetitive construct we know, from P1 $\wedge \neg B$, that

$(\forall x\colon [u,x] \equiv [v,x])$,

which combined with P2 , yields

$$(\forall x\colon [i,x] \equiv ([v,x] \wedge (\exists y\colon [v,y] \wedge P(x) < P(y)))) \ .$$

At the end of the program we then have for all x

$$\begin{aligned} [d,x] &\equiv ([v,x] \wedge \neg[i,x]) \\ &\equiv ([v,x] \wedge (\neg[v,x] \vee (\forall y\colon [v,y] \Rightarrow P(x) \geq P(y)))) \\ &\equiv ([v,x] \wedge (\forall y\colon [v,y] \Rightarrow P(x) \geq P(y))) \ , \end{aligned}$$

which exhibits the truth of R to be established.

EXAMPLE 10.6. *Equidistant locus of two vertices in an undirected graph.*

The *equidistant locus* of two vertices u and v in a finite connected undirected graph is the set of all vertices for which the length of the shortest path to u equals the length of the shortest path to v. Let m and n be the names of the vertices M and N, respectively, of a finite connected undirected graph G, which is given as in Example 7.7 (p. 50). We are requested to design a program that records in associons $(1,?)$ the equidistant locus of M and N. For that reason we define two integer functions P_M and P_N on the vertices of G, with $P_M(x)$ and $P_N(x)$ being the length of the shortest path from x to, respectively, M and N. It is given that initially $\neg[1,?]$ holds. The program should establish the truth of

$$R \qquad (\forall x\colon [1,x] \equiv ([v,x] \wedge P_M(x) = P_N(x))) \ .$$

We generate two sets of vertices, called U_M and U_N, around M and N, respectively. Let $maxp$ be a ghost variable with initial value zero and let its value be incremented by one per execution of the guarded list. Upon every execution of the guarded list U_M and U_N are the sets of vertices x with $P_M(x) \leq maxp$ and $P_N(x) \leq maxp$, respectively. The vertices x in U_M with $P_M(x) < maxp$ are recorded in the set I_M. U_N likewise contains an "interior" I_N. For any path length we can then distill those vertices x for which $P_M(x) = P_N(x)$, as they are in $U_M \cap U_N$, but not in $I_M \cup I_N$. We can extend U_M and U_N until all vertices of G are in $U_M \cap U_N$.

Recording the sets U_M, U_N, I_M, and I_N in associons $(um,?)$, $(un,?)$, $(im,?)$, and $(in,?)$, respectively, we maintain the following invariants:

$$P1 \qquad (\forall x\colon ([um,x] \vee [un,x]) \Rightarrow [v,x]) \ ,$$

P2 $(\forall x: [im,x] \equiv ([um,x] \wedge (\exists y: ([um,y] \wedge P_M(y) > P_M(x)) \vee ([un,y] \wedge P_N(y) > P_M(x)))))$,

P3 $(\forall x: [in,x] \equiv ([un,x] \wedge (\exists y: ([un,y] \wedge P_N(y) > P_N(x)) \vee ([um,y] \wedge P_M(y) > P_N(x)))))$,

P4 $(\forall x: [l,x] \equiv ([um,x] \wedge [un,x] \wedge P_M(x) = P_N(x)))$.

From P2 and P3 we may conclude

$$(\forall x: ([um,x] \wedge \neg[im,x] \wedge [un,x] \wedge \neg[in,x]) \Rightarrow P_M(x) = P_N(x)) .$$

As for all x

$$(([um,x] \wedge \neg[un,x]) \vee ([un,x] \wedge \neg[um,x])) \Rightarrow P_M(x) \neq P_N(x) ,$$

we may write P4 as

P4' $(\forall x: [l,x] \equiv (([um,x] \vee [un,x]) \wedge P_M(x) = P_N(x)))$.

As a variant function we choose $(Nx: [v,x] \wedge \neg([im,x] \vee [in,x]))$.

Program:

```
loc (um,?),(un,?),(im,?),(in,?):
    [] :⇒ (um,m),(un,n); [un,m] :⇒ (l,m);
    do z: [v,z] ∧ ¬([um,z] ∨ [un,z]) →
       x: [um,x] :⇒ (im,x); x: [un,x] :⇒ (in,x);
       x,y: [im,x] ∧ [a,x,y] :⇒ (um,y);
       x,y: [in,x] ∧ [a,x,y] :⇒ (un,y);
       x: [um,x] ∧ ¬[im,x] ∧ [un,x] ∧ ¬[in,x] :⇒ (l,x)
    od
col
```

Upon termination of the repetitive construct we know, from P1 $\wedge \neg B$,

$$(\forall x: [v,x] \equiv ([um,x] \vee [un,x])) ,$$

which, combined with P4' , exhibits R to hold.

CHAPTER 11

DYNAMICALLY CREATED NAMES

When programming for associons we establish new relations between names, on account of already existing relations between these names. We may be faced with the situation that the collection of names in the initially existing relations --the names in the associons initially present-- is insufficient to represent the new relations. If the number of necessary new names is statically known --i.e. is independent of the initial state--, then they can be introduced as constants in the program text. If, however, this number is not known until the execution of the program, then new names have to be created dynamically. An instance of this we encounter in the following problem.

Let V be a set of elements. We are requested to write a program that generates all subsets of V . The set V is given by

$$(\forall x\colon [v,x] \equiv \text{" } x \text{ is the name of an element of } V \text{ "}) \ .$$

We know, furthermore, of the initial state that $\neg[s,?] \wedge \neg[m,?,?]$ holds. For any set X , S(X) will denote the set of all subsets of X . We have to establish the truth of

R $\quad (\forall x\colon [s,x] \equiv \text{" } x \text{ is the name of an element of } S(V) \text{ "}) \wedge$
$\quad\quad (\forall x,y\colon [m,x,y] \equiv ([s,x] \wedge [v,y] \wedge \text{"element } y \text{ in subset } x \text{ "}))\ .$

We obtain the invariant relation by replacing in R the constant set V by a variable set W , which is invariantly contained in V . W is initialized empty -- S(W) then contains the empty set only-- and is extended until it equals V . The relation

$$S(W \cup \{u\}) = (S(W) \cup \{x \cup \{u\}\colon x \in S(W)\})$$

suggests to extend W with one element of $V \setminus W$ at a time. Recording the names of the elements of W in associons (w,?) , the program will have the following structure.

```
loc (w,?):
    [] :⇒ (s,empty);
    do u: [v,u] ∧ ¬[w,u] →
       "for all  x ∈ S(W)  generate the set  x ∪ {u} ";
       [] :⇒ (w,u)
    od
col
```

The refinement of the quoted statement requires the generation of as many new names as the cardinality of S(W) . For any set in S(W) --whose names are recorded in associons (s,?)-- a new set has to be generated. As a consequence we need a new name for each solution of the equation x: [s,x] . For this reason we introduce the possibility to create a new name for each solution of an equation by changing the syntax of "target associon format" as follows.

```
<target associon format> ::= (<nue>{,<nue>})
<nue> ::= <name> | <unknown> | !
```

The execution of such a closure statement should effect a final state in which a new name has been created for each solution of the left-hand side. In order to guarantee that this is always possible, we pose the restriction that no target associon format containing an exclamation point "!" may match the equation in the left-hand side.

The effect of the statement can best be described as if it were executed in two steps. If all target associon formats contain an exclamation point, then the first step is a "skip", otherwise it is the closure statement that has the original equation as its left-hand side, and the target associon formats that do not contain the exclamation point as its right-hand side. In the second step the associons with the new names are created, possible cascading being taken care of in the first step. Each element of the solution set of the equation is extended with a unique new name --"new" meaning "distinct from any name occurring in the present associons or as a constant in the program text"--, and for these "extended solutions" the corresponding target associons (with each exclamation point replaced by the new name) are created. If x: E(x) denotes the equation in the left-hand side, then the number of created new names equals $(N x\colon E(x))|U$, in which U denotes the state after the first step (or, thanks to the "nonmatching", after the second step).

EXAMPLE 11.1. The effect of the closure statement

```
x: [a,x] :⇒ (b,!),(c,!,!)
```

is that (*N*x: [a,x]) distinct new names are introduced. For each new name n the associons (b,n) and (c,n,n) are created. The closure statement

```
x: [a,x] :⇒ (a,x),(b,!),(c,!,!)
```

has the same effect.
(End of example.)

In the example of the subsets we want to create a new name for every solution of the equation x: [s,x] . We, therefore, create a one-to-one function between the solutions of x: [s,x] and the new names. This function is temporarily recorded in associons (f,?,?) . The closure statement

```
x: [s,x] :⇒ (f,x,!)
```

establishes the truth of

```
(∀x,y: [f,x,y] ≡ ([s,x] ∧ " y  is the new name corresponding to
                                x ") .
```

Program:

```
loc (w,?):
    [] :⇒ (s,empty);
    do u: [v,u] ∧ ¬[w,u] →
       loc (f,?,?):
           x: [s,x] :⇒ (f,x,!);
           x,y,z: [f,x,y] ∧ [m,x,z] :⇒ (m,y,z);
           y: [f,?,y] :⇒ (m,y,u),(s,y)
       col;
       [] :⇒ (w,u)
    od
col
```

Now that we can create new names, we could think about creating integers. The following program should --according to Peano-- generate the natural numbers.

```
[] :⇒ (int,zero);
do x: [int,x] ∧ ¬[suc,x,?] → [] :⇒ (int,!),(suc,x,!) od
```

The above program, of course, does not terminate. But this is due to the fact that the set of natural numbers is infinite. We can, however, instead of trying to generate all natural numbers, generate some suitable subset.

Suppose we want to solve Example 10.5 (Most distant leaves of an outtree. (p. 82)) by computing for each vertex x the length P(x) of the path from the root to x . The subset of the natural numbers ranging from zero through (Nx: [v,x]) will then be sufficient to represent these function values.

We record the fact that P(x) = y by creating the associon (p,x,y) . As the most distant leaves are vertices x0 for which there exists no vertex x1 with P(x1) = P(x0) + 1 , the following program will do the job.

```
loc (int,?),(suc,?,?):
    "create the natural numbers from zero through (Nx: [v,x]) ";
    loc (p,?,?):
        x: [v,x] ∧ ¬[s,?,x] :⇒ (p,x,zero);
        x0,x1,y0,y1: [s,x0,x1] ∧ [p,x0,y0] ∧ [suc,y0,y1] :⇒ (p,x1,y1);
        x0,y0,y1: [p,x0,y0] ∧ [suc,y0,y1] ∧ ¬[p,?,y1] :⇒ (d,x0)
    col
col
```

The operation "create the natural numbers from zero through (Nx: [v,x]) " can be refined into

```
loc (h,?):
    [] :⇒ (int,zero);
    do x,y: [int,x] ∧ ¬[suc,x,?] ∧ [v,y] ∧ ¬[h,y] →
       [] :⇒ (int,!),(suc,x,!),(h,y)
    od
col .
```

REMARK 11.1. Apart from the generation of the natural numbers, the program for the recording of the most distant leaves does not involve explicit repetition. Would our programs be executed in an environment in which a sufficient amount of integers were pre-created, we would be able to do away with explicit repetition in many solutions. Nonrepetitive programs give the implementation a larger freedom to choose its own sequencing. In the above

program for the recording of the most distant leaves, e.g., it is only at the two semicolons that the implementation has a "synchronization task", whereas in the repetitive version (Example 10.5 (p. 82)) the implementation is forced to break up the generation of the associons (i,?) and (u,?) into a number of successive steps that is proportional to the depth of the tree (which in our new version could be recognized as the "depth of cascading").
(End of remark.)

We shall now solve in the above fashion two other problems that were previously solved using repetition.

EXAMPLE 11.2. *Balanced vertices.* This problem was earlier solved in Chapter 8. We shall now record for any vertex x the --in Chapter 8 defined-- function value L(x) , by establishing the truth of

$$(\forall x,y\colon [l,x,y] \Rightarrow ([v,x] \wedge [int,y] \wedge L(x) \geq y)) \wedge$$
$$(\forall x,y\colon [nl,x,y] \equiv ([l,x,y] \wedge L(x) > y)) \ .$$

(In words: associons (l,?,?) record lower bounds for the function L , associons (nl,?,?) characterize the recorded lower bounds that are not attained.)
Then

$$(\forall x,y\colon ([l,x,y] \wedge \neg[nl,x,y]) \Rightarrow L(x) = y) \ .$$

In order to be able to generate the associons (nl,?,?) , we must first create the relation "smaller than" on the set of introduced integers. We shall for that reason establish the truth of

$$(\forall x,y\colon [sm,x,y] \equiv ([int,x] \wedge [int,y] \wedge x < y)) \ ,$$

which can be accomplished by "create the smaller-than relation":

$$x,y\colon [suc,x,y] :\Rightarrow (sm,x,y);$$
$$x,y,z\colon [sm,x,y] \wedge [sm,y,z] :\Rightarrow (sm,x,z)$$

Program:

```
loc (int,?),(suc,?,?),(sm,?,?):
    "create the natural numbers from  zero  through  (Nx: [v,x]) ";
    "create the smaller-than relation";
    loc (l,?,?),(nl,?,?):
        x: [v,x] ∧ ¬[s,x,?] :⇒ (l,x,zero);
        x0,x1,y0,y1: [l,x0,y0] ∧ [s,x1,x0] ∧ [suc,y0,y1] :⇒ (l,x1,y1);
        x,y0,y1: [l,x,y0] ∧ [l,x,y1] ∧ [sm,y0,y1] :⇒ (nl,x,y0);
        x0,x1,y: [l,x0,y] ∧ ¬[nl,x0,y]
               ∧ [l,x1,y] ∧ ¬[nl,x1,y] :⇒ (b,x0,x1)
    col
col
```

REMARK 11.2. The three statements catering for the creation of the associons (l,?,?) and (nl,?,?) can be combined into the following two.

```
x: [v,x] :⇒ (l,x,zero);
x0,x1,y0,y1,y2: [l,x0,y0] ∧ [s,x1,x0] ∧ [l,x1,y1] ∧ [suc,y0,y2]
              ∧ [sm,y1,y2] :⇒ (l,x1,y2),(nl,x1,y1)
```

(End of remark.)

(End of example.)

EXAMPLE 11.3. *Equidistant locus of two vertices in an undirected graph.* This is the same problem as Example 10.6 (p. 84). We shall now record for any vertex x the function values $P_M(x)$ and $P_N(x)$, by establishing the truth of (cf. Example 11.2 (p. 90))

$$(\forall x,y\colon [pm,x,y] \Rightarrow ([v,x] \wedge [int,y] \wedge P_M(x) \leq y)) \wedge$$
$$(\forall x,y\colon [pn,x,y] \Rightarrow ([v,x] \wedge [int,y] \wedge P_N(x) \leq y)) \wedge$$
$$(\forall x,y\colon [npm,x,y] \equiv ([pm,x,y] \wedge P_M(x) < y)) \wedge$$
$$(\forall x,y\colon [npn,x,y] \equiv ([pn,x,y] \wedge P_N(x) < y)) .$$

Then

$$(\forall x,y\colon ([pm,x,y] \wedge \neg[npm,x,y]) \Rightarrow P_M(x) = y) \wedge$$
$$(\forall x,y\colon ([pn,x,y] \wedge \neg[npn,x,y]) \Rightarrow P_N(x) = y) .$$

In order to be able to perform a correct initialization, we must know the maximum of the created natural numbers. We create max as the name of the next natural number by means of the program

```
        x: [int,x] ∧ ¬[sm,x,?] :⇒ (suc,x,max);
        x: [int,x] :⇒ (sm,x,max);
        [] :⇒ (int,max) .
```

Program:

```
loc (int,?),(suc,?,?),(sm,?,?):
    "create the natural numbers from zero through (Nx: [v,x]) ";
    "create the smaller-than relation";
    "create max as the name of the next natural number";
    loc (pm,?,?),(pn,?,?),(npm,?,?),(npn,?,?):
        x: [v,x] :⇒ (pm,x,max),(pn,x,max);
        [] :⇒ (pm,m,zero),(pn,n,zero),(npm,m,max),(npn,n,max);
        x0,x1,y0,y1,y2: [a,x0,x1] ∧ [pm,x0,y0] ∧ [pm,x1,y1]
           ∧ [suc,y0,y2] ∧ [sm,y2,y1] :⇒ (pm,x1,y2),(npm,x1,y1);
        x0,x1,y0,y1,y2: [a,x0,x1] ∧ [pn,x0,y0] ∧ [pn,x1,y1]
           ∧ [suc,y0,y2] ∧ [sm,y2,y1] :⇒ (pn,x1,y2),(npn,x1,y1);
        x,y: [pm,x,y] ∧ ¬[npm,x,y] ∧ [pn,x,y] ∧ ¬[npn,x,y] :⇒ (l,x)
    col
col
```

REMARK 11.3. Often one is not only interested in the length of a shortest path, but also in the identity of a (the) shortest path(s). Suppose, e.g., that we want to record for each vertex x in the equidistant locus of M and N all shortest paths from M to that vertex x . We wish to establish the truth of

```
(∀x,y,z: [s,x,y,z] ≡ ([l,x] ∧ [a,y,z] ∧ "there exists a
          shortest path from M to x containing y → z ")) .   (1)
```

We know that each "target vertex" x in the equidistant locus is in all shortest paths from M to that vertex x . If vertex x0 is in a shortest path to x , and x0 is adjacent to a vertex x1 with $P_M(x1) = P_M(x0) - 1$, then x1 is also in a shortest path to x , and x1 → x0 is part of a shortest path from M to x . We record the names of the vertices in the shortest paths in associons (h,?,?) by establishing the truth of

```
(∀x,y: [h,x,y] ≡ ([l,x] ∧ [v,y] ∧ "vertex y in a shortest
                                   path from M to x ")) .
```

The truth of (1) is established by inserting at the end of the program (before the first "col") the following text.

```
; loc (h,?,?):
      x: [1,x] :⇒ (h,x,x);
      x,x0,x1,y0,y1: [h,x,x0] ∧ [pm,x0,y0] ∧ ¬[npm,x0,y0]
         ∧ [suc,y1,y0] ∧ [a,x0,x1] ∧ [pm,x1,y1] :⇒ (s,x,x1,x0),(h,x,x1)
  col
```

(End of remark.)

(End of example.)

CHAPTER 12

RECORDING THE CLIQUES OF AN UNDIRECTED GRAPH

In Chapter 8 a program was designed that records an arbitrary clique of an undirected graph. An undirected graph may contain many cliques. The recording of all of them is a more cumbersome affair (an NP-complete problem, in the complexity jargon [1]). This is one reason why we have chosen this problem. The other reason is that it is a problem that with traditional programming tools would probably be solved in a backtracking fashion (cf. [3]).

A finite undirected graph G (possibly containing loops) is given as in Example 7.7 (p. 50). Let, furthermore, be given that $\neg$[c,?] $\wedge$ $\neg$[m,?,?] holds, and let it be requested to write a program that establishes the truth of

R (∀x: [c,x] ≡ " x is the name of a clique") ∧
(∀x,y: [m,x,y] ≡ ([c,x] ∧ [v,y] ∧ " y in clique x")) .

As a clique is a maximal complete subgraph, we decide to record all complete subgraphs of G . Given the complete subgraphs, the cliques can easily be determined.

Let V denote the set of vertices of G . For any subset X of V , CS(X) will denote the set of all complete subgraphs of G with vertices from X . We record the complete subgraphs of G by establishing the truth of

R1 (∀x: [cs,x] ≡ " x is the name of an element of CS(V) ") ∧
(∀x,y: [mc,x,y] ≡ ([cs,x] ∧ [v,y] ∧ " y in x")) .

This problem resembles the problem of the generation of all subsets, for which we designed a program in Chapter 11. We shall try to derive an analogous program for the generation of all complete subgraphs. As in the preceding chapter, we obtain the invariant relation by replacing the constant set V by a variable set W :

P1 (∀x: [w,x] ≡ " x is the name of an element of W ") ∧
(∀x: [w,x] ⇒ [v,x]) ∧
(∀x: [cs,x] ≡ " x is the name of an element of CS(W) ") ∧
(∀x,y: [mc,x,y] ≡ ([cs,x] ∧ [v,y] ∧ " y in x")) .

Likewise, the program will be of the same structure as the one on page 87. There is, of course, one difference: we do not record all subgraphs, but only the complete ones. More precisely: per execution of the guarded list we generate the extension with vertex u only for those $x \in CS(W)$, that satisfy

$$(\forall z: [mc,x,z] \Rightarrow [a,u,z]) .$$

This quantified condition can be written as

$$\neg(\exists z: [mc,x,z] \wedge \neg[a,u,z]) .$$

We record, as usual, the existentially quantified condition in associons $(h,?,?)$, i.e. we maintain as an extra invariant

P2 $$(\forall x,u: [h,x,u] \equiv ([v,u] \wedge (\exists z: [mc,x,z] \wedge \neg[a,u,z]))) .$$

(In words: $[h,x,u]$ holds if and only if the complete subgraph x contains at least one vertex to which u is not adjacent.) Per execution of the guarded list we shall generate the extension with vertex u for each $x \in CS(W)$ satisfying $\neg[h,x,u]$.

After the recording of the complete subgraphs of G, we shall determine which complete subgraphs are cliques. By having the scope of the associons $(h,?,?)$ range over the complete program, we can exploit the fact that P2 holds after the generation of $CS(V)$. A complete subgraph x of G is a clique of G if and only if for each vertex y in the complement of x there exists at least one vertex in x to which y is not adjacent; in formula:

$$(\forall x: [c,x] \equiv ([cs,x] \wedge (\forall y: ([v,y] \wedge \neg[mc,x,y]) \Rightarrow [h,x,y]))) ,$$

or

$$(\forall x: [c,x] \equiv ([cs,x] \wedge \neg(\exists y: [v,y] \wedge \neg[mc,x,y] \wedge \neg[h,x,y]))) . \qquad (1)$$

We determine the cliques of G by recording the quantified condition of (1) in associons $(nc,?)$, i.e. we establish the truth of

$$(\forall x: [nc,x] \equiv ([cs,x] \wedge (\exists y: [v,y] \wedge \neg[mc,x,y] \wedge \neg[h,x,y]))) .$$

Then (1) can be written as

$$(\forall x: [c,x] \equiv ([cs,x] \wedge \neg[nc,x])) .$$

The program will, consequently, have the following structure.

```
loc (cs,?),(mc,?,?),(h,?,?):
    loc (w,?):
        [] :⇒ (cs,empty);
        do u: [v,u] ∧ ¬[w,u] →
           "for all  x ∈ CS(W)  satisfying  ¬[h,x,u]
            generate the extension of  x  with  u ,
            reestablish  P2 ";
            [] :⇒ (w,u)
        od
    col;
    loc (nc,?):
        x,y: [cs,x] ∧ [v,y] ∧ ¬[mc,x,y] ∧ ¬[h,x,y] :⇒ (nc,x);
        x: [cs,x] ∧ ¬[nc,x] :⇒ (c,x)
    col;
    x,y: [c,x] ∧ [mc,x,y] :⇒ (m,x,y)
col
```

The refinement of the quoted statement is, except for the condition "¬[h,x,u]" and the reestablishing of P2 , the same as in Chapter 11 (cf. p. 88):

"for all ...":

```
loc (f,?,?):
    x: [cs,x] ∧ ¬[h,x,u] :⇒ (f,x,!);                    (2)
    x,y,z: [f,x,y] ∧ [mc,x,z] :⇒ (mc,y,z);
    y: [f,?,y] :⇒ (mc,y,u),(cs,y);
    x,y,z: [f,x,y] ∧ [h,x,z] :⇒ (h,y,z);
    y,z: [f,?,y] ∧ [v,z] ∧ ¬[a,z,u] :⇒ (h,y,z)
col
```

The last two statements of the above block reestablish P2 . A vertex z has a disconnection with the new y if and only if either it has a disconnection with the old x , or it is not adjacent to the added vertex u . (Note that the second and the fourth statement are "pseudo-cascades", as in the present associons (f,x,y) x and y are never equal to each other.)

As each vertex of W is in at least one complete subgraph of W , we have for all x

$[w,x] \equiv [mc,?,x]$.

We can, consequently, eliminate the associons $(w,?)$.

* *
*

We are interested in the recording of cliques. As each clique is a complete subgraph, we decided to record all complete subgraphs of G . The number of complete subgraphs, however, can exceed the number of cliques by far. The complete n-graph, for instance, has only one clique, but it has 2^n complete subgraphs. We shall now try to reduce the number of recorded complete subgraphs.

Statement (2) records the names of the new complete subgraphs. We shall accomplish the reduction by changing statement (2) so as to have it create fewer new names.

Per execution of the guarded list a new complete subgraph is generated for each $x \in CS(W)$ that can be extended with vertex u . The old x remains recorded as well. We investigate whether there are cases in which there is no need to preserve the old complete subgraph. Of course, we can never destroy the old complete subgraph. We can, however, transform it into the new one by simply adding to it the vertex u .

For each recorded complete subgraph x of W we call a vertex z a candidate for x if and only if $\neg[w,z] \wedge \neg[h,x,z]$. When we generate an extension of a recorded complete subgraph x of W , we do that because ultimately we wish to find all cliques of G that are supergraphs of that x and that contain, besides the vertices in x , only candidate vertices for x . If vertex u is a candidate for x , and u is adjacent to all other candidates for x , then any such clique will contain vertex u . In that case there is no need to continue the recording of the old complete subgraph x , as it does not (and will not) contain vertex u . Instead of creating a new complete subgraph next to the old one, we shall, in such a case, add vertex u to the old complete subgraph.

For any recorded complete subgraph x of W that has u as a candidate vertex, we record the candidates in associons $(e,x,?)$, i.e. we establish the truth of

(∀x,z: [e,x,z] ≡ ([cs,x] ∧ ¬[h,x,u] ∧
[v,z] ∧ ¬[w,z] ∧ ¬[h,x,z])) .

The old x has to be kept --and a new name has to be created-- if and only if

(∃z: z ≠ u: [e,x,z] ∧ ¬[a,z,u]) .

Otherwise, we record vertex u as a member of x ; the reestablishment of P2 (with respect to x) may in that case be omitted. We thus arrive at the following text, which should replace statement (2).

```
loc (e,?,?):
    x,z: [cs,x] ∧ ¬[h,x,u]
       ∧ [v,z] ∧ ¬[w,z] ∧ ¬[h,x,z] :⇒ (e,x,z);
    x,z: [e,x,z] ∧ ¬[a,z,u] ∧ z ≠ u :⇒ (f,x,!);
    x: [e,x,?] ∧ ¬[f,x,?] :⇒ (mc,x,u)
col
```

As each vertex of W is in at least one clique of W , we can eliminate the associons (w,?) in this version as well.

If G is the complete n-graph (because of its regularity always an easy example for counting purposes), then we have [a,z,u] for any two vertices z and u (z ≠ u). Then the above block never creates a new name, and the program records only one complete subgraph.

REMARK 12.1. This is another example of a program that traditionally would be solved by backtracking. As in Example 7.3 (Missionaries and Cannibals (p. 45)), we first generate a large enough portion of the "search tree", large enough to ensure that its complement does not contain any "interesting" elements, and then we trace which (or: whether any) interesting elements have been generated. It seems that, when programming with associons, backtracking is replaced by this "concurrent generation".
(End of remark.)

REMARK 12.2. We have solved an NP-complete problem with a program that contains only one repetitive construct. Its guarded list is executed exactly (Nx: [v,x]) times. All closure statements in the program are either noncascading or pseudo-cascading. If the complexity theory is also applicable to associon programs, this seems to imply that the execution of a noncascading closure statement can be an NP-complete problem.

(End of remark.)

CHAPTER 13

ON WHAT WE HAVE REJECTED

In the preceding chapters we have presented the ultimate product of our research. We have not tried to give a true design history. Most of the proposals that looked promising at first sight, but that had to be rejected at a later stage, have not been treated. In this chapter we shall deal with some of these proposals, and we shall indicate why we have decided to reject them. We cannot "prove" why we could not accept these proposals. All we shall do is try to communicate our motives for rejection.

It was E.W. Dijkstra who first started to think about programming with associons. His original ideas included, besides the creation of associons, the explicit destruction of associons. The statement

$$x\colon [v,x] :\Rightarrow \overline{(w,x)} \ ,$$

for instance, would, for any solution of the equation $x\colon [v,x] \wedge [w,x]$, destroy the associon (w,x) . Its execution would establish the truth of the implication

$$(\forall x\colon [v,x] \Rightarrow \neg[w,x]) \ .$$

The negative match of a target associon and the left-hand side was originally also allowed. The statement

S $\qquad x,y\colon [r,x,y] \wedge \neg[t,x] :\Rightarrow (t,y) \ ,$

for instance, would establish the truth of the implication

$$(\forall x,y\colon ([r,x,y] \wedge \neg[t,x]) \Rightarrow [t,y]) \ . \qquad (1)$$

The semantics of our basic statement were described in an operational fashion. We postulated that the effect of the execution of, for instance, the above statement S is equivalent to the effect of a repetitive implementation that, as long as the equation

$$x,y\colon [r,x,y] \wedge \neg[t,x] \wedge \neg[t,y]$$

has a nonempty solution set, selects an arbitrary nonempty subset of that solution set, and that creates for all these selected solutions the associons (t,y) . (Destruction was described analogously.)

If such a repetitive implementation would select subsets of size one only, then we would have a sequential execution; by only requiring the selected subsets to be nonempty, we leave it to the implementation to choose the degree of concurrency. Note that such an implementation does not necessarily create the minimal number of target associons that is required to establish (1). If S is, e.g., executed in the state

$$\{(),(r,a,b),(r,b,a)\} ,$$

then either (t,a) , or (t,b) , or both will be created. It is a nondeterministic construct.

We have tried to write a program for the selection of exactly one (arbitrary) element of a nonempty set V . The names of the elements of V are recorded in associons (v,?) , the name of the selected element should be recorded in an associon (s,?) . We tried the statement

$$x\colon [v,x] :\Rightarrow (s,x) ,$$

but that selects the whole V . We tried the statement

$$x\colon [v,x] \wedge \neg[s,?] :\Rightarrow (s,x) ,$$

but that selects an arbitrary nonempty subset of V . We tried to select all of them, followed by the destruction of all but one:

$$x\colon [v,x] :\Rightarrow (s,x);$$
$$x,y\colon [s,x] \wedge [s,y] \wedge x \neq y :\Rightarrow \overline{(s,y)} .$$

But then all of them could have been destroyed again.

As we wrote at that time: "a new tool seems indicated". We considered the introduction of a special kind of unknowns --that we during this discussion will denote by adding an exclamation point--, to indicate that the described repetitive implementation should select subsets of size one only, and we arrived at the program

$$x!\colon [v,x] \wedge \neg[s,?] :\Rightarrow (s,x) .$$

(The program

$$x!\colon [v,x] :\Rightarrow (s,x)$$

would still copy the whole V , be it one at a time.)

For "ordinary" unknowns the implementation may select an arbitrary nonempty subset of the solution set, for unknowns "x!" it selects only one so-

lution at a time. What if we have both types in one statement? We declared the order in which the unknowns are listed to be important, and we designed a very complicated rule --too complicated to be repeated here-- for the permitted selections of solutions. Consider, e.g. the execution of the program

$$x,y!: [a,x,y] :\Rightarrow (b,x,y) \qquad (2)$$

in a state satisfying $\neg[b,?,?]$. For each x such that $[a,x,?]$, it would select one y satisfying $[a,x,y]$, and for these (x,y)-pairs, it would create the associons (b,x,y) . The program

$$y!,x: [a,x,y] :\Rightarrow (b,x,y), \qquad (3)$$

on the other hand, would (if $[a,?,?]$) select one y satisfying $[a,?,y]$, and (with that y) it would, for all x satisfying $[a,x,y]$, create the associons (b,x,y) .

Even with such a scheme, repetition was still indispensable. We did, for instance, not succeed in writing a nonrepetitive program for the selection of one characteristic element per equivalence class. As it, furthermore, turned out to be virtually impossible to give a nonoperational description of its semantics, we dropped this complicated scheme, and we decided to have only one type of unknowns, indicating by ":→" (instead of ":⇒") that the implementation should select one solution at a time. The statement (3) could then be coded as

$$y: [a,?,y] \wedge \neg[1,?] :\rightarrow (1,y);$$
$$x,y: [a,x,y] \wedge [1,y] :\Rightarrow (b,x,y) ,$$

whereas the equivalent of (2) would require repetition. (The single arrow ":→" was dropped again when we started to think about repetition.)

The definition of our statement should be such that it allows for the derivation of practicable theorems that characterize all relevant aspects of the semantics. When we tried to achieve this, we did not succeed for statements with a negative match, neither did we succeed in a satisfactory manner for statements with associon destruction. We shall elucidate this.

The property that the execution of the statement

S $\qquad x,y: [r,x,y] \wedge \neg[t,x] :\Rightarrow (t,y)$

establishes the truth of the implication

$$(\forall x,y: ([r,x,y] \wedge \neg[t,x]) \Rightarrow [t,y]) \qquad (4)$$

does not exclude the possibility that too many associons (t,?) are created. One could, for instance, in order to establish the truth of (4), create for all "prevailing" names x the associons (t,x) (cf. the set V in the proof of Lemma 5.2 (p. 25)).

We wish to express that if prior to the execution of S no "wrong" associons (t,?) --"wrong" in the sense that "their unknowns do not constitute a solution of the left-hand side"-- were present, then they will not be present afterwards either; i.e. we wish to express the invariance of "no wrong associons present". For the statement S this would be the invariance of the implication

$$(\forall y\colon [t,y] \Rightarrow (\exists x\colon [r,x,y] \wedge \neg[t,x])) \ .$$

In the case of a negative match --as in S -- this invariance, however, is in general not guaranteed, as the created target associons can cause the solution set of the left-hand side to shrink. We decided to disallow the negative match of target associons and the left-hand side.

For a statement with associon destruction the prevention of shrinking solution sets gives rise to the disallowance of target associons that fit positive presence conditions of the left-hand side. With such a prohibition we obtain the invariance of "no wrong associons absent". The statement

$$x,y\colon [r,x,y] \wedge \neg[s,x] :\Rightarrow \overline{(s,y)} \ , \qquad (5)$$

for instance, would establish the truth of the implication

$$(\forall y\colon (\exists x\colon [r,x,y] \wedge \neg[s,x]) \Rightarrow \neg[s,y])$$

under invariance of "no wrong associons absent":

$$(\forall y\colon \neg[s,y] \Rightarrow (\exists x\colon [r,x,y] \wedge \neg[s,x])) \ . \qquad (6)$$

This invariance, however, is of little use. As the set of names is unlimited, implication (6) will be false for any finite set of associons. Having "automatic destruction" of local associons at block exit, there is little sense in introducting explicit associon destruction as well. If one wishes to record that certain associons are not "valid" anymore, one can always record this in additional associons, and write, instead of (5), e.g.,

$$x,y\colon [r,x,y] \wedge ([invalid,x] \vee \neg[s,x]) :\Rightarrow (invalid,y) \ .$$

We decided to reject explicit associon destruction.

At this stage we hoped to have simplified our basic statement --then called "creation statement"-- enough, as to allow for a nonoperational definition of its semantics. This turned out to be rather difficult for cascading creation statements. We did, however, not want to disallow cascade, as we felt

$$x,y\colon [v,x] \wedge [r,x,y] :\Rightarrow (v,y)$$

to be a statement too beautiful to be rejected. We decided to introduce the mathematical concept "closure of a set of associons". We postulated that the creation statement generates the closure, and we called it "closure statement" since.

* *
*

It was at this stage of the research that we again introduced some ideas that we have not maintained.

We, for instance, wished to distinguish between unknowns that only occur in the left-hand side and the other unknowns of a closure statement, calling them "existential" and "free" variables, respectively. This distinction caused matters to become needlessly intricate, and after we realized that

$$(\forall x\colon P(x) \Rightarrow Q)$$

is equivalent to

$$(\exists x\colon P(x)) \Rightarrow Q\ ,$$

we dropped this distinction again.

We also have considered to have the closure statement abort if initially there is a "wrong" associon present. The statement

$$x\colon [v,x] :\Rightarrow (w,x)\ ,$$

for instance, should abort if initially

$$(\exists x\colon [w,x] \wedge \neg[v,x])$$

holds. Although it was felt encouraging to have a construct in the language that could give rise to an abort, we did not pursue this idea. In order to avoid abortion we had to introduce auxiliary associons to record intermediate results. The presence conditions of these auxiliary associons lengthened

the equations of subsequent closure statements.

Another idea that has been abandoned was the treatment of the concurrent closure statement as the basic statement. This complicated our presentation tremendously: we had subscripts --and sometimes even subsubscripts-- hanging around everywhere. Although the concurrent closure statement is more powerful than the nonconcurrent one, we could only find far-fetched cases in which the concurrent closure statement could not be written as a nonconcurrent one.

* *
*

The execution of a statement, be it a block or a single closure statement, cannot effect the destruction of associons that are global to that statement. The effect of a statement that does not create global associons, is, consequently, equivalent to that of "skip". When we first embarked upon the introduction of repetition, we observed that there is little sense in continuing to execute a repetitive construct if an execution of the "repeatable statement" has not created any global associons. We considered not to introduce explicit guards in our repetitive construct, but to postulate that the execution of a repetitive construct finishes as soon as a single execution of the repeatable statement has not created any associons global to it. Every repetitive construct that does not involve dynamic generation of new names, would then be a terminating construct.

For such a scheme the formulation of the invariance theorem turned out to be rather difficult. For we had to characterize the states in which no global associons are created.

Without the "single arrow" closure statement we were, furthermore, not able to write a program for the selection of one characteristic element per equivalence class. The single arrow closure statement had in the meantime become suspect. For its effect could only be described nonoperationally by means of the (*N*...)-concept, and that concept usually does not provide a simple interface.

We then decided to combine (from the single arrow closure statement) the idea of selection of one solution at a time, and (from traditional programming) the idea of explicit guards. We introduced equations as guards. A

guarded list is only eligible for execution if its guarding equation has a nonempty solution set. If a guarded list is chosen to be executed, then an arbitrary solution of its guarding equation is selected and may be "used" during the execution of the guarded list.

We decided to allow a set of guarded commands, although we never came across a problem for which the program contained a repetitive construct with more than one guard. It may be that repetitive constructs with only one guard suffice, but we introduced the general guarded command set, because without complicating matters it enabled us to formulate Theorem 9.1 (p. 67) about the arbitrariness of the order in which an implementation of the closure statement may trace (and create) the missing associons.

CHAPTER 14

EPILOGUE

We have proposed an instruction repertoire that should allow ultraconcurrent execution. New hardware techniques may make such implementations feasible (cf. [14]). One could try to reach such a goal by the introduction of large fancy data-types, upon which numerous powerful operations are defined. We have not done so. We have designed a limited and simple instruction repertoire, consisting of mathematically well-manageable concepts. We did not introduce any concept without discussing its desirability, nor did we introduce any concept without simultaneously providing a discipline for its use.

As a data structure we have chosen a set of relations. On this data structure we have defined only one operation, the closure statement. It creates, as a function of the existing relations, new relations. Its semantics are formally defined by means of its weakest pre-condition. Two sequencing primitives have been introduced: concatenation --the semicolon-- and repetition. In order to be able to judge when --and how-- to employ repetition, an appropriate design principle --the recording of the quantified condition-- has been formulated.

Designing a language for a nonexisting machine, we enjoyed the (lucky) circumstance of being forced to obey Wirth's following demand (cited from [15], underlining as in the original): "The first criterion that any future programming language must satisfy {...} is a complete definition without reference to compiler or computer. Such a definition will inherently be of a rather mathematical nature. To many hardcore programmers, this demand perhaps sounds academic and (nearly) impossible. I certainly have not claimed that it is easy! I only claim that it is a necessary condition for genuine progress."

We have fairly often encountered programs that could be used for many (at first sight different) problems. This may perhaps confirm the idea that there do not exist too many really different algorithms. It also seems to indicate that, when programming for associons, we arrive at some sort of "canonical" programs.

A need of recursion did not arise. We did introduce a "cascading creation" which is nothing else than the creation of a (generalized) closure. If U denotes the set of vertices of some out-tree, of which R represents the predecessor-successor relation, then determining the closure under R of the subset of U that contains only the root, is also known as "traversing a tree". Its classical solution is a recursive one. The recursive solution for tree-traversal is often regarded to be the prototype application for recursive subroutines. The generation of closures, however, is not restricted to trees, it may be applied in directed graphs in general. As such cascade may be considered to be a logical generalization of recursion. Recent publications (cf. [5]) seem to indicate that the generation of closures in directed graphs is a frequently recurring theme.

There is another branch of computing science in which the data structure consists of a set of relations, viz. that of "relational data bases" [4], and one may wonder how different our approaches are.

In articles on (relational) data bases the data structure usually models named objects of the "real world". (Examples then deal with "suppliers and parts", "employers and employees", etc.) As a consequence the name of a relation --which is not considered to be the name of a "real" object-- is treated differently from the arguments of a relation. Our associon

(largerthan,transitive) ,

representing that the larger-than relation is a transitive relation, would then either have to be coded as

largerthan(transitive) ,

or as

transitive(largerthan) ,

meaning, respectively, "transitivity is one of the properties of the larger-than relation", and "larger-than is one of the transitive relations". This choice is not an irrelevant one, as names of relations may usually not occur as arguments in relations. The symmetry in the associon representation allows us to write both statements like

x: [x,transitive] ∧ [x,reflexive] ∧ [x,symmetric] :⇒ (x,equivalence) ,

and statements like

x: [largerthan,x] :⇒ (smallerthan,x) .

A second major difference seems to be that in relational data bases only some standard operations for the creation of relations are defined. These operations may comprise (in the data base jargon) composition, (cyclic) join, projection, restriction, tie, etc. All of these are special cases of the closure statement. The resulting relation, however, is always a relation with a different name. A cascading creation can, consequently, not be expressed in such repertoires.

Another branch of computing science in which the need of operations on relations --including that of the generation of the transitive closure-- has been uttered, is the field of artificial intelligence. To quote a paper on programming languages for artificial intelligence [2]: "In particular, {...} one would like to be able to use content-retrievable ordered triples (or n-tuples) {...} as basic data types." The languages Sail and Planner do have these content-retrievable tuples. They are then called, respectively, associations and assertions. In neither of these languages, however, the creation of closures has been recognized as a primitive operation.

The above observations give a strong indication that the creation of closures is indeed a fundamental concept. The preceding chapters have shown that it is very well possible to construct a programming language around it.

BIBLIOGRAPHY

[1] Aho, A.V., Hopcroft, J.E. & Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] Bobrow, D.G. & Raphael, B., *New Programming Languages for Artificial Intelligence Research*, Computing Surveys 6, No. 3, September 1974, pp. 153 - 174.

[3] Bron, C. & Kerbosch, J., *Finding All Cliques of an Undirected Graph*, Communications of the ACM 16, No. 9, September 1973, pp. 575 - 577.

[4] Codd, E.F., *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM 13, No. 6, June 1970, pp. 377 - 387.

[5] Dijkstra, E.W., *Determinacy and Recursion Versus Nondeterminacy and the Transitive Closure*, Report EWD 456, October 1974.

[6] Dijkstra, E.W., *A Discipline of Programming*, Prentice-Hall, 1976.

[7] Dijkstra, E.W., Feijen, W.H.J. & Rem, M., *Associons: An Effort Towards Accomodating Potentially Ultra-high Concurrency*, Report DFR 0 / EWD 435, July 1974.

[8] Dijkstra, E.W., Feijen, W.H.J. & Rem, M., *Associons Continued*, Report DFR 1 / EWD 439, August 1974.

[9] Harary, F., *Graph Theory*, Addison-Wesley, 1969.

[10] Hoare, C.A.R., *An Axiomatic Basis for Computer Programming*, Communications of the ACM 12, No. 10, October 1969, pp. 576 - 580.

[11] Hoare, C.A.R., *Recursive Data Structures*, Report STAN-CS-73-400, Stanford University, October 1973.

[12] Kelley, J.L., *General Topology*, Van Nostrand, 1955.

[13] Naur, P. (ed.), *Report on the Algorithmic Language ALGOL 60*, Communications of the ACM 3, No. 5, May 1960, pp. 299 - 314.

[14] Ozkarahan, E.A., Schuster, S.A. & Smith, K.C., *RAP: An Associative Processor for Data Base Management*, Proceedings AFIPS NCC 44, 1975,

pp. 379 - 387.

[15] Wirth, N., *Programming Languages: What To Demand and How To Access Them*, Berichte des Instituts für Informatik 17, ETH Zürich, March 1976.

INDEX

$\tilde{A}$	17
abort	4
acyclic graph	48
adjacent vertices	50
assignment statement	4
associon	6
associon format	16,21
BB	67
B_i	67
balanced vertices	63
block	21
BNF	2
cascading closure statement	18
clique	56
closure statement	15,17,20,35
closure with respect to a binary relation	33
closure with respect to a closure statement C(S,U)	25
complete graph	55
concurrent closure statement	21
condition (on the state)	3, 7
connected	50
constant	18,59
cycle	48,50
deterministic construct	4
edge-disjoint paths	51
empty associon	7
equation	9,10
equidistant locus	84
exclamation point	87

F_i 30
factor 9
fit 12

guard 58
guarded command 58
guarded list 58

$H_k(R)$ 67

initialized constant 59
invariance theorem for closure statements 39
invariance theorem for repetitive constructs 72

kernel 49

left-hand side 17
length of a walk 47
local associons 21
loop 50

match 12
mechanism 3
minimal transition pair 53
multiple edges 50

(*N* ...) 11
name 5, 9
negative match 12,20
negative presence condition 9
nontrivial walk 47

out-tree 22

path 47,50
positive match 12
positive presence condition 9
predecessor 53

predicate P|U 7
predicate transformer 4
presence condition 7, 9
primary 9

question-mark 8

reachable 48
repetitive construct 58,67

$\hat{S}$ 23
safely connected vertices 51
semicolon 4
set difference \ 12
skip 4
solution set Z(E,U) 9
spanning subgraph 55
state 7
statement list 2, 4
state vector 7
strong component 61
strongly connected 61
subgraph 55
successor 53
supergraph 55
symmetric set difference ÷ 12

$T(x), \tilde{T}(x)$ 17
target associon 17
target associon format 17
target associon format set 17
term 9
terminal vertex 48
terminating construct 35
transitive closure 19
trap 48

unilateral component 61
unilaterally connected 61
universal relation 6
unknown 9

variant function 60

walk 47,50
wdec 72
weakest pre-condition wp(S,R) 2

Z(E,U) 9

OTHER TITLES IN THE SERIES MATHEMATICAL CENTRE TRACTS

A leaflet containing an order-form and abstracts of all publications mentioned below is available at the Mathematisch Centrum, Tweede Boerhaavestraat 49, Amsterdam-1005, The Netherlands. Orders should be sent to the same address.

MCT 1 T. VAN DER WALT, *Fixed and almost fixed points*, 1963. ISBN 90 6196 002 9.

MCT 2 A.R. BLOEMENA, *Sampling from a graph*, 1964. ISBN 90 6196 003 7.

MCT 3 G. DE LEVE, *Generalized Markovian decision processes, part I: Model and method*, 1964. ISBN 90 6196 004 5.

MCT 4 G. DE LEVE, *Generalized Markovian decision processes, part II: Probabilistic background*, 1964. ISBN 90 6196 006 1.

MCT 5 G. DE LEVE, H.C. TIJMS & P.J. WEEDA, *Generalized Markovian decision processes, Applications*, 1970. ISBN 90 6196 051 7.

MCT 6 M.A. MAURICE, *Compact ordered spaces*, 1964. ISBN 90 6196 006 1.

MCT 7 W.R. VAN ZWET, *Convex transformations of random variables*, 1964. ISBN 90 6196 007 X.

MCT 8 J.A. ZONNEVELD, *Automatic numerical integration*, 1964. ISBN 90 6196 008 8.

MCT 9 P.C. BAAYEN, *Universal morphisms*, 1964. ISBN 90 6196 009 6.

MCT 10 E.M. DE JAGER, *Applications of distributions in mathematical physics*, 1964. ISBN 90 6196 010 X.

MCT 11 A.B. PAALMAN-DE MIRANDA, *Topological semigroups*, 1964. ISBN 90 6196 011 8.

MCT 12 J.A.TH.M. VAN BERCKEL, H. BRANDT CORSTIUS, R.J. MOKKEN & A. VAN WIJNGAARDEN, *Formal properties of newspaper Dutch*, 1965. ISBN 90 6196 013 4.

MCT 13 H.A. LAUWERIER, *Asymptotic expansions*, 1966, out of print; replaced by MCT 54 and 67.

MCT 14 H.A. LAUWERIER, *Calculus of variations in mathematical physics*, 1966. ISBN 90 6196 020 7.

MCT 15 R. DOORNBOS, *Slippage tests*, 1966. ISBN 90 6196 021 5.

MCT 16 J.W. DE BAKKER, *Formal definition of programming languages with an application to the definition of ALGOL 60*, 1967. ISBN 90 6196 022 3.

MCT 17 R.P. VAN DE RIET, *Formula manipulation in ALGOL 60, part 1*, 1968. ISBN 90 6196 025 8.

MCT 18 R.P. VAN DE RIET, *Formula manipulation in ALGOL 60, part 2*, 1968. ISBN 90 6196 038 X.

MCT 19 J. VAN DER SLOT, *Some properties related to compactness*, 1968. ISBN 90 6196 026 6.

MCT 20 P.J. VAN DER HOUWEN, *Finite difference methods for solving partial differential equations*, 1968. ISBN 90 6196 027 4.

MCT 21 E. WATTEL, *The compactness operator in set theory and topology*, 1968. ISBN 90 6196 028 2.

MCT 22 T.J. DEKKER, *ALGOL 60 procedures in numerical algebra, part 1*, 1968. ISBN 90 6196 029 0.

MCT 23 T.J. DEKKER & W. HOFFMANN, *ALGOL 60 procedures in numerical algebra, part 2*, 1968. ISBN 90 6196 030 4.

MCT 24 J.W. DE BAKKER, *Recursive procedures*, 1971. ISBN 90 6196 060 6.

MCT 25 E.R. PAERL, *Representations of the Lorentz group and projective geometry*, 1969. ISBN 90 6196 039 8.

MCT 26 EUROPEAN MEETING 1968, *Selected statistical papers, part I*, 1968. ISBN 90 6196 031 2.

MCT 27 EUROPEAN MEETING 1968, *Selected statistical papers, part II*, 1969. ISBN 90 6196 040 1.

MCT 28 J. OOSTERHOFF, *Combination of one-sided statistical tests*, 1969. ISBN 90 6196 041 X.

MCT 29 J. VERHOEFF, *Error detecting decimal codes*, 1969. ISBN 90 6196 042 8.

MCT 30 H. BRANDT CORSTIUS, *Excercises in computational linguistics*, 1970. ISBN 90 6196 052 5.

MCT 31 W. MOLENAAR, *Approximations to the Poisson, binomial and hypergeometric distribution functions*, 1970. ISBN 90 6196 053 3.

MCT 32 L. DE HAAN, *On regular variation and its application to the weak convergence of sample extremes*, 1970. ISBN 90 6196 054 1.

MCT 33 F.W. STEUTEL, *Preservation of infinite divisibility under mixing and related topics*, 1970. ISBN 90 6196 061 4.

MCT 34 I. JUHÁSZ, A. VERBEEK & N.S. KROONENBERG, *Cardinal functions in topology*, 1971. ISBN 90 6196 062 2.

MCT 35 M.H. VAN EMDEN, *An analysis of complexity*, 1971. ISBN 90 6196 063 0.

MCT 36 J. GRASMAN, *On the birth of boundary layers*, 1971. ISBN 90 6196 064 9.

MCT 37 J.W. DE BAKKER, G.A. BLAAUW, A.J.W. DUIJVESTIJN, E.W. DIJKSTRA, P.J. VAN DER HOUWEN, G.A.M. KAMSTEEG-KEMPER, F.E.J. KRUSEMAN ARETZ, W.L. VAN DER POEL, J.P. SCHAAP-KRUSEMAN, M.V. WILKES & G. ZOUTENDIJK, *MC-25 Informatica Symposium*, 1971. ISBN 90 6196 065 7.

MCT 38 W.A. VERLOREN VAN THEMAAT, *Automatic analysis of Dutch compound words*, 1971. ISBN 90 6196 073 8.

MCT 39 H. BAVINCK, *Jacobi series and approximation*, 1972. ISBN 90 6196 074 6.

MCT 40 H.C. TIJMS, *Analysis of (s,S) inventory models*, 1972. ISBN 90 6196 075 4.

MCT 41 A. VERBEEK, *Superextensions of topological spaces*, 1972. ISBN 90 6196 076 2.

MCT 42 W. VERVAAT, *Success epochs in Bernoulli trials (with applications in number theory)*, 1972. ISBN 90 6196 077 0.

MCT 43 F.H. RUYMGAART, *Asymptotic theory of rank tests for independence*, 1973. ISBN 90 6196 081 9.

MCT 44 H. BART, *Meromorphic operator valued functions*, 1973. ISBN 90 6196 082 7.

MCT 45 A.A. BALKEMA, *Monotone transformations and limit laws*, 1973. ISBN 90 6196 083 5.

MCT 46 R.P. VAN DE RIET, *ABC ALGOL, A portable language for formula manipulation systems, part 1: The language*, 1973. ISBN 90 6196 084 3.

MCT 47 R.P. VAN DE RIET, *ABC ALGOL, A portable language for formula manipulation systems, part 2: The compiler*, 1973. ISBN 90 6196 085 1.

MCT 48 F.E.J. KRUSEMAN ARETZ, P.J.W. TEN HAGEN & H.L. OUDSHOORN, *An ALGOL 60 compiler in ALGOL 60, Text of the MC-compiler for the EL-X8*, 1973. ISBN 90 6196 086 X.

MCT 49 H. KOK, *Connected orderable spaces*, 1974. ISBN 90 6196 088 6.

MCT 50 A. VAN WIJNGAARDEN, B.J. MAILLOUX, J.E.L. PECK, C.H.A. KOSTER, M. SINTZOFF, C.H. LINDSEY, L.G.L.T. MEERTENS & R.G. FISKER (Eds), *Revised report on the algorithmic language ALGOL 68*, 1976. ISBN 90 6196 089 4.

MCT 51 A. HORDIJK, *Dynamic programming and Markov potential theory*, 1974. ISBN 90 6196 095 9.

MCT 52 P.C. BAAYEN (ed.), *Topological structures*, 1974. ISBN 90 6196 096 7.

MCT 53 M.J. FABER, *Metrizability in generalized ordered spaces*, 1974. ISBN 90 6196 097 5.

MCT 54 H.A. LAUWERIER, *Asymptotic analysis, part 1*, 1974. ISBN 90 6196 098 3.

MCT 55 M. HALL JR. & J.H. VAN LINT (Eds), *Combinatorics, part 1: Theory of designs, finite geometry and coding theory*, 1974. ISBN 90 6196 099 1.

MCT 56 M. HALL JR. & J.H. VAN LINT (Eds), *Combinatorics, part 2: graph theory, foundations, partitions and combinatorial geometry*, 1974. ISBN 90 6196 100 9.

MCT 57 M. HALL JR. & J.H. VAN LINT (Eds), *Combinatorics, part 3: Combinatorial group theory*, 1974. ISBN 90 6196 101 7.

MCT 58 W. ALBERS, *Asymptotic expansions and the deficiency concept in statistics*, 1975. ISBN 90 6196 102 5.

MCT 59 J.L. MIJNHEER, *Sample path properties of stable processes*, 1975. ISBN 90 6196 107 6.

MCT 60 F. GÖBEL, *Queueing models involving buffers*, 1975. ISBN 90 6196 108 4.

* MCT 61 P. VAN EMDE BOAS, *Abstract resource-bound classes, part 1*. ISBN 90 6196 109 2.

* MCT 62 P. VAN EMDE BOAS, *Abstract resource-bound classes, part 2*. ISBN 90 6196 110 6.

MCT 63 J.W. DE BAKKER (ed.), *Foundations of computer science*, 1975. ISBN 90 6196 111 4.

MCT 64 W.J. DE SCHIPPER, *Symmetrics closed categories*, 1975. ISBN 90 6196 112 2.

MCT 65 J. DE VRIES, *Topological transformation groups 1 A categorical approach*, 1975. ISBN 90 6196 113 0.

MCT 66 H.G.J. PIJLS, *Locally convex algebras in spectral theory and eigenfunction expansions*. ISBN 90 6196 114 9.

* MCT 67 H.A. LAUWERIER, *Asymptotic analysis, part 2.* ISBN 90 6196 119 x.

* MCT 68 P.P.N. DE GROEN, *Singularly perturbed differential operators of second order.* ISBN 90 6196 120 3.

* MCT 69 J.K. LENSTRA, *Sequencing by enumerative methods.* ISBN 90 6196 125 4.

MCT 70 W.P. DE ROEVER JR., *Recursive program schemes: semantics and proof theory*, 1976. ISBN 90 6196 127 0.

MCT 71 J.A.E.E. VAN NUNEN, *Contracting Markov decision processes*, 1976. ISBN 90 6196 129 7.

* MCT 72 J.K.M. JANSEN, *Simple periodic and nonperiodic Lamé functions and their applications in the theory of conical waveguides.* ISBN 90 6196 130 0.

* MCT 73 D.M.R. LEIVANT, *Absoluteness of intuitionistic logic.* ISBN 90 6196 122 x.

MCT 74 H.J.J. TE RIELE, *A theoretical and computational study of generalized aliquot sequences.* ISBN 90 6196 131 9.

* MCT 75 A.E. BROUWER, *Treelike spaces and related connected topological spaces.* ISBN 90 6196 132 7.

MCT 76 M. REM , *Associons and the closure statement.* ISBN 90 6196 135 1.

* MCT 77

* MCT 78 E. de Jonge, A.C.M. van Rooij, *Introduction to Riesz spaces*, 1977. ISBN 90 6196 133 5

* MCT 79 M.C.A. VAN ZUIJLEN, *Empirical distributions and rankstatistics*, 1977. ISBN 90 6196 145 9.

* MCT 80 P.W. HEMKER, *A numerical study of stiff two-point boundary problems*, 1977. ISBN 90 6196 146 7.

MCT 81 K.R. APT & J.W. DE BAKKER (eds), *Foundations of computer science II*, part I, 1976. ISBN 90 6196 140 8.

MCT 82 K.R. APT & J.W. DE BAKKER (eds), *Foundations of computer science II*, part II, 1976. ISBN 90 6196 141 6.

* MCT 83 L.S. VAN BENTEM JUTTING, *Checking Landau's "Grundlagen" in the automath system*, 1977. ISBN 90 6196 147 5.

MCT 84 H.L.L. Busard, *The translation of the elements of Euclid from the Arabic into Latin by Hermann of Carinthia books vii-xii*, 1977. ISBN 90 6196 148 3

An asterik before the number means "to appear".