AN ABSTRACT GRAPH WALK ALGORITHM

L. VARGA

(Received 12 July, 1979)

1. Introduction

In the last decade lots of work have been done in the field of the machine independent definitions of programming languages. One of the most remarkable results is the Vienna Definition Language (VDL). The first description of the Vienna Definition Language was published in [1]. In [2] VDL was used as a definition tool of the semantics of PL/1. [3] gives an informal explanation of the basic ideas of VDL. Lee [4] proposed some minor improvements in order to make the language suitable for the description of algorithms other than abstract interpreters. [5] gives a systematic introduction into VDL.

Though the VDL was originally constructed for the definition of the abstract syntax and semantics of programming languages, it has proven to be an excellent design language for top-down program development. It permits concentrating rather on the logical solutions of problems, than the form and constraints within that the solution must be stated. The language helps the programmers to think in terms of a hierarchy of routines and expresses structured logic.

Hoare's deductive system, using axioms, inference rules and assertions, can be used to prove correctness of structured programs. (The axioms and the rules of consequence appear in the Appendix.)

Therefore, the Vienna Definition Language with Hoare's deductive system can be used for verified design of programs. In this paper, this is applied for defining a general graph walk algorithm, where the walk strategy and the operations over the nodes are not specified. An abstract algorithm is given in this way, from that concrete graph walks can be deduced.

As example, the abstract algorithm is used for the specification of a linkage editor model.

2. The VDL-graph

Let
is-node-set = ({⟨*s*: is-node⟩|is-select(*s*)}),
is-node = (⟨s-value: is-pred⟩, ⟨s-desc:is-select-list⟩),

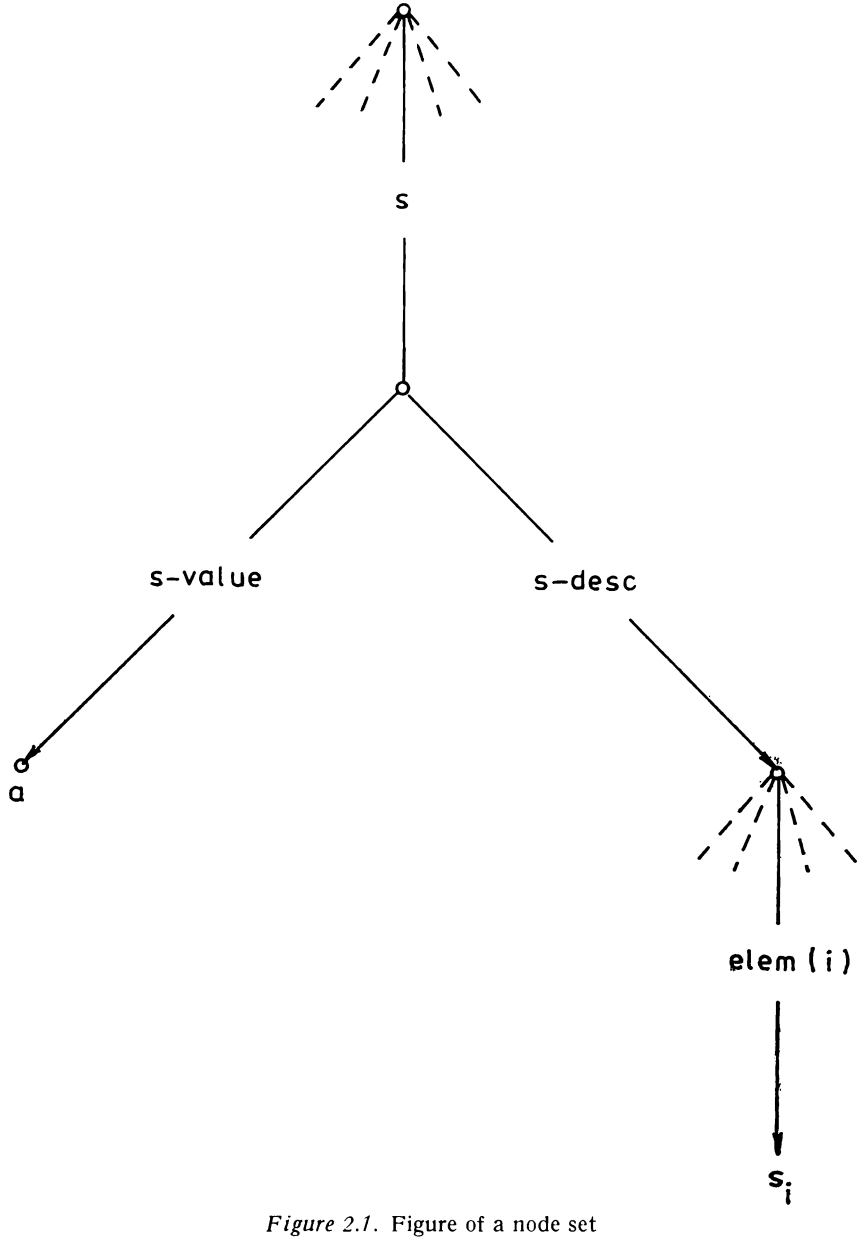

Figure 2.1. Figure of a node set

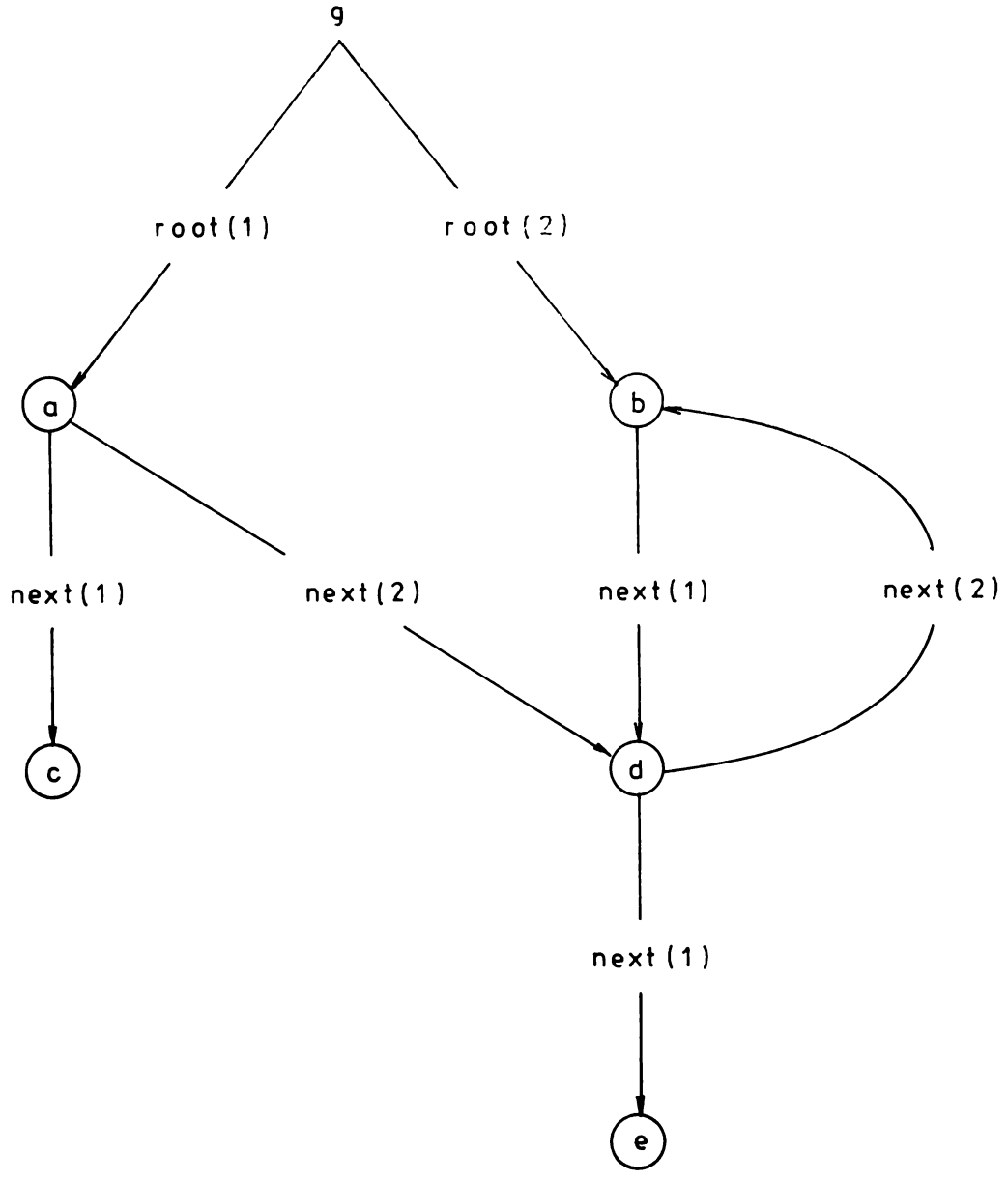

Figure 2.2. VDL-graph

where "is-select" and "is-pred" represent arbitrary predicates. Such an object is shown by Figure 2.1., where

$$a \in \{x \mid \text{is-pred}(x)\}$$

and

$$s,\ s_i \in \{s' \mid \text{is-select}(s')\}.$$

Let

is-node-set $(f) = T$.

Definition 2.1. Let $t \in f$ if

$$(\exists\, s,\ \text{is-select}\ (s)\ (\ s(f) = t)\ .$$

Definition 2.2. Let $t \in f$, $n \in f$. The node *n refers to t* if and only if

$$(\exists\, i,\ 1 \leq s \leq \text{length}\ s\text{-desc}(n))\ (\text{elem}(i)\ (s\text{-desc}(n))\ (f) = t).$$

Notationally we shall use the form

$$n \rightarrow t.$$

Definition 2.3. The node t_k *is reachable from* node t_1, or there exists *a reference path* from t_1 to t_k if and only if

$$t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_k, \qquad (t_i \in f,\ i = 1, 2, \ldots, k).$$

We shall use the following notation for the reference path

$$t_1 \rightarrow * t_k.$$

Definition 2.4. The set of VDL-graph is

$$\{g \mid \text{is-pred-graph}(g)\},$$

where

$$\text{is-pred-graph} = \text{is-node-set}$$

and there exists a non-empty subset $M(g)$ of the nodes of g distinguished with the property that any node $n \in g$ and $n \notin M(g)$ can be reached from at least one element of $M(g)$.

Definition 2.5. If $m \in M(g)$ then m is called a directly reachable node.

It follows that each node of a VDL-graph can be reached from at least one directly reachable node.

Definition 2.6. Let root (i) be the function, for which

$$\text{root}\ (i) = s_i, \qquad i = 1, 2, \ldots, m$$

if

$$M(g) = \{s_1\ (g), s_2(g), \ldots, s_m(g)\}$$

and

$$\text{value}\ (n) = s\text{-value}\ (n),$$

$$\text{next}\ (i)\ (n) = \text{elem}\ (i)\ (s\text{-desc}\ (n))\ (g),\ 1 \leq i \leq \text{length}\ (\text{s-desc}(n)\ (g)).$$

These functions can be used as selectors, for example

$$\text{value} \cdot \text{next}\ (2) \cdot \text{next}(1) \cdot \text{root}(3)(g) =$$

$$= \text{value}\Big(\text{next}(2)\big(\text{next}(1)(\text{root}(3)(g))\big)\Big).$$

Definition 2.7. A terminal node is a node n such that

$$\Big(\forall i,\ 1 \leq i \leq \text{length}\ (\text{s-desc}\ (n))\Big)\ (\text{next}\ (i)\ (n) = \Omega),$$

where Ω is the null object.

Example 2.1. Using the functions defined above, the structure of a VDL-graph can be visualised by a graph. The VDL-graph is denoted by the following relations

$$g = (\langle \text{root}\ (1) : n_1\rangle, \langle \text{root}(2) : n_2\rangle, \langle s_3 : n_3\rangle, \langle s_4 : n_4\rangle, \langle s_5 : n_5\rangle),$$

$$n_1 = (\langle s\text{-value} : \text{a}\rangle, \langle \text{s-desc} : \langle s_3, s_4\rangle\rangle),$$

$$n_2 = (\langle \text{s-value} : \text{b}\rangle, \langle \text{s-desc} : \langle s_4\rangle\rangle),$$

$$n_3 = (\langle \text{s-value} : \text{c}\rangle, \langle \text{s-desc} : \langle\rangle\rangle),$$

$$n_4 = (\langle \text{s-value} : \text{d}\rangle, \langle \text{s-desc} : s_5, s_2\rangle\rangle),$$

$$n_5 = (\langle \text{s-value} : \text{e}\rangle, \langle \text{s-desc} : \langle\rangle\rangle),$$

where $M(g) = \{n_1, n_2\}$ is shown on Figure 2.1 The nodes on the Figure 2.1 are circles and the values yielded by the nodes of the VDL-graph are in the circles. They can be selected by the function "value". For example

$$\text{value}\ (\text{root}(1)\ (g)) = \text{a}.$$

The relationships between the nodes are represented by arrows and the arrows are named by function next (i).

The figure of a VDL-graph visualizes its structure in this way, but the formula of a VDL-graph does not reflect it directly. However it is not difficult to construct a formula that also satisfies this requirement. But this problem is not dealt here.

In Definition 2.5. the selection operations are defined on the VDL-graph. Construction operations can also be defined on it, but it is intended to deal with statical VDL-graph only, hence construction operations are not defined.

3. The graph walk algorithm and its verification

The graph walk is a fundamental operation. Most of the selection and construction operations can be established on it.

A graph walk can be carried out according to different strategies. In a graph walk algorithm each node of the VDL-graph is processed one after the other. The walk strategy determines the order of the nodes to be processed. In the following, we give a general graph walk algorithm, where the walk strategy and the operations over the nodes are not specified. In this way an abstraction of the graph walk algorithms is given from which concrete graph walks can be deduced by the specification of the walk strategy and the operation over the nodes.

In this paper the VDL with the Hoare's deductive system is used for the verified design of a graph walk algorithm. At verification the following notation is used:

$$\begin{array}{c}\{R\}\\ P\\ \{Q\}\end{array}$$

where R and Q are assertions about the state of the abstract machine and P is a VDL-program. This may be interpreted as follows: If Q is true before execution of a VDL-program P, then R is true after execution of P provided P terminates.

The correctness proof of VDL-programs can be carried out in similar way to correctness proofs of parallel programs [7]. The extension of Hoare's method to VDL-programs can be found in [8].

Let the set of the states of the abstract machine be:

$$(\xi \mid \text{is-state}(\xi))$$

where

$$\begin{aligned}\text{is-state} = (&\langle\text{s-input:is-pred-graph}\rangle,\\ &\langle\text{s-table:is-table}\rangle,\\ &\langle\text{s-control : is-control}\rangle)\end{aligned}$$

$$\text{is-table} = \text{is-value-set}$$

and is-vâlue $= \{Y, N\}$.

Let the initial state ξ_0 of the machine be characterized by

$$\begin{aligned}\xi_0 = \mu_0(&\langle\text{s-input: } g\rangle, \langle\text{s-table: } t_0\rangle,\\ &\langle\text{s-control : walk } (g; \text{next-selector, proc})\rangle)\end{aligned}$$

where

$$\text{is-pred-graph } (g) = T$$

$$t_0 = (\{\langle s : N\rangle\, s(g) \in M(g)\})$$

and let next-selector be a function and proc a macro. They are formal parameters of the algorithm. The function next-selector specifies the walk strategy and macro proc determines the operation over the nodes.

Definition 3.1. The function next-selector is a function over the set

$$\{t \mid \text{is-table } (t)\}$$

and the range of the function is

$$\text{is-sel}\hat{\text{e}}\text{ctor} \cup \{\Omega\},$$

so that if

$$(\exists s \in \text{is-sel}\hat{\text{e}}\text{ctor})\ (s(t) = N)$$

then next-selector(t) is a selector s so that

$$s(t) = N$$

else

$$\text{next-selector}(t) = \Omega .$$

Informally, the function next-selector(t) furnishes one of the selectors of the table t as

$$s(t) = N$$

if such an s exists and yields the object Ω otherwise. If the function next-selector is applied to a table t several times the result is the same.

Definition 3.2. Let proc be a function over the set

$$\{n \mid \text{is-node}(n)\}$$

with range $\{T, F\}$ so that

$$\text{proc}(n) = T$$

if and only if the macro proc (s-value(n)) has been executed:

$$\{\text{proc}(n)\}$$
$$\text{proc}\ (\text{s-value}(n))$$
$$\{\text{is-node}(n)\}.$$

Theorem 3.1. Let ξ_0 be the object given above, then the following algorithm executes the instruction

$$\text{proc}\ (\text{value}(n))$$

exactly once for each $n \in g$:
Level 1:

$$\text{walk}\ (g;\ \text{next-selector},\ \text{proc}) =$$
$$\text{next-selector}\ (s\text{-table}(\xi)) = \Omega \rightarrow \text{null}$$

$T \rightarrow$walk (g; next-selector, proc);

process-node (n, s; proc);

n : next-node (s, g);

s : next-name (;next-selector)

next-name (;next-selector) =

PASS: next-selector (s-table(ξ))

next-node (s, g) =

PASS: $s(g)$

Level 2:

process-node (n, s; proc) =

proc (s-value (n)),

process-desc (s-desc(n), s)

Level 3:

process-desc (w, s) =

length (w) = 0 $\rightarrow$ process-name (s)

$T \rightarrow$ process-desc (tail(w), s);

set (head(w))

process-name (s) =

s-table: μ(s-table (ξ); $\langle s{:}Y\rangle$)

Level 4:

set (s) =

s(s-table (ξ)) = $\Omega \rightarrow$ link (s)

T $\rightarrow$ null

link (s) =

s-table: μ (s-table (ξ); $\langle s{:}N\rangle$).

Proof. The partial correctness of the program will be proved according to the following specification:

Let the initial state ξ_0 be specified by the predicate

$$\varphi(\xi_0):(\forall s, \alpha(s, \xi_0)\neq\Omega)(\alpha(s, \xi_0) = N$$
$$\wedge s(g)\in M(g))\wedge(\forall s, s(g)\in M(g))(\alpha(s, \xi_0) = N)\wedge\text{is-pred-graph}(g)$$

where

$$\alpha(s, \xi) = s(s\text{-table }(\xi));$$

and let the input-output predicate be the following:

$$\psi(\xi_0, \zeta):(\forall s, s(g)\neq\Omega)\,(\text{proc }(s(g))\wedge\text{is-pred graph }(g)\wedge g = s\text{-input }(\xi_0)),$$

so that the instruction proc (s-value(n)) is executed exactly once for each $n\in g$.

The proof is done as follows. The program is divided into levels. At the first step the correctness is proved for the first level making use of lemmas and theorems not provable on that level. The correctness of the latters is assumed. At the next step the correctness of the assumptions is shown in a similar way. Continueing this process, the rest of the assumptions is verified by proving lemmas.

Level 1.

To show the partial correctness of the algorithm, first of all we have to specify the assertion, that must be invariant at Level 1:

$$Q_1(\xi):R_1(\xi)\wedge R_2(\xi)\wedge R_3(\xi)\wedge R_4(\xi)$$

where

$$R_1(\xi):(\forall s, \alpha(s, \xi)\neq\Omega)(s(g)\neq\Omega\wedge\alpha(s, \xi)) = Y\vee\alpha(s, \xi) = N);$$

$$R_2(\xi):(\forall s, \alpha(s, \xi) = Y)(\text{proc }(s(g)));$$

$$R_3(\xi):(\forall s, \text{proc}(s(g)))(\alpha(s, \xi) = Y);$$

$$R_4(\xi):(\forall s', s'(g)\neq\Omega\wedge\alpha(s', \xi) = \Omega)$$
$$((\exists s)(\alpha(s, \xi) = N\wedge s(g)\rightarrow * s'(g))).$$

Lemma 3.1.1.

$$\varphi(\xi_0)\supset Q_1(\xi_0).$$

Proof. It is obvious that

$$\varphi(\xi_0)\supset R_1(\xi_0)\wedge R_2(\xi_0)\wedge R_3(\xi_0).$$

But

$$(\forall s, s(g)\in M(g))(\alpha(s, \xi_0) = N)\supset$$
$$(\forall s', s'(g)\neq\Omega\wedge\alpha(s', \xi_0) = \Omega)(s'\,(g)\notin M(g))$$

and hence it is also true, that

$$(\forall s',\ s'(g) \neq \Omega \wedge \alpha(s',\ \xi_0) = \Omega)((\exists s)(s(g) \in M(g) \wedge s(g) \rightarrow * s'(g))).$$

Lemma 3.1.2.

$$\{Q_1(\xi) \wedge \neg P(\xi) \wedge \alpha(s,\ \xi) = N\}$$
$$s\text{: next-name (;next-selector)},$$
$$\{Q_1(\xi) \wedge \neg P(\xi)\}$$

where

$$P(\xi)\text{: next-selextor } (s\text{-table}(\xi)) = \Omega$$

Proof. From Definition 3.1 we have

$$Q_1(\xi) \wedge \neg P(\xi) \supset \alpha(\text{next-selector } (s\text{-table } (\xi)),\ \xi) = N$$

and therefore the proof can be given by the rule of assignment.

Lemma 3.1.3.

$$\{Q_1(\xi) \wedge \neg P(\xi) \wedge \alpha(s,\ \xi) = N \wedge n = s(g)\}$$
$$n\text{: next-node } (s, g)$$
$$\{Q_1(\xi) \wedge \neg P(\xi) \wedge \alpha(s, \xi) = N\}.$$

Proof. It follows from the definition of macro n: next-node (s, g) immediately.

Now, suppose we have already the proof.

Assumption 3.1.1.

$$\{Q_1(\xi)\}$$
$$\text{process-node } (n, s; \text{proc})$$
$$\{Q_1(\xi) \wedge \neg P(\xi) \wedge \alpha(s, \xi) = N \wedge n = s(g)\}.$$

Then from the rule of composition it is that

$$\{Q_1(\xi)\}$$
$$\text{process-node } (n, s; \text{proc});$$
$$n\text{: next-node } (s, g);$$
$$s\text{: next-name (;next-selector)}$$
$$\{Q_1(\xi) \wedge \neg P(\xi)\},$$

hence the rule of iteration shows the partial correctness of the algorithm at Level 1.

Level 2.

The assertion

$$Q_2(\xi, s): R_1(\xi) \wedge R_{21}(\xi, s) \wedge R_{31}(\xi, s) \wedge R_4(\xi)$$

Where

$$R_{21}(\xi, s): \big(\forall s', s' \neq s \wedge \alpha(s', \xi) = Y \big) \big(\text{proc}(s'(g)) \big)$$

$$R_{31}(\xi, s): \big(\forall s', s' \neq s \wedge \text{proc}(s'(g)) \big) \big(\alpha(s', \xi) = Y \big)$$

must be invariant at Level 2.

From Definition 3.2 we have

$$\{Q_2(\xi, s) \wedge \text{proc}(n) \wedge n = s(g) \wedge n \neq \Omega\}$$

$$\underline{\text{proc}}\ (s\text{-value}\ (n))$$

$$\{Q_2(\xi, s) \wedge n = s(g) \wedge n \neq \Omega\}. \tag{i}$$

Assumption 3.1.2.

$$\{Q_2(\xi, s) \wedge n = s(g) \wedge \alpha(s, \xi) = Y\}$$

$$\underline{\text{process-desc}}\ (s\text{-desc}(n), s)$$

$$\{Q_2(\xi, s) \wedge n = s(g) \wedge \alpha(s, \xi) = N\} \tag{ii}$$

and the proofs (i), (ii) are interference-free.

If Assumption 3.1.2 is correct, then using the rule of parallelism we have

$$\{Q_2(\xi, s) \wedge n = s(g) \wedge \text{proc}\ (n) \wedge \alpha(s, \xi) = Y\}$$

$$\underline{\text{proc}}\ (\text{s-value}\ (\text{n})),$$

$$\underline{\text{process-desc}}\ (s\text{-desc}(n), s)$$

$$\{Q_2(\xi, s) \wedge n = s(g) \wedge \alpha(s, \xi) = N\}.$$

But

$$Q_2(\xi, s) \wedge n = s(g) \wedge \text{proc}(n) \wedge \alpha(s, \xi) = Y \supset Q_1(\xi)$$

and

$$Q_1(\xi) \wedge \alpha(s, \xi) = N \supset Q_2(\xi\ s)$$

that shows the correctness of Assumption 3.1.1.

Level 3.

In order to show the correctness of Assumption 3.1.2 let

$$Q_3(\xi, s): R_1(\xi) \wedge R_{21}(\xi, s) \wedge R_{31}(\xi, s) \wedge R_{41}(\xi, s)$$

where

$$R_{41}(\xi, s) : (\forall s', s'(g) \neq \Omega \wedge \alpha(s', \xi) = \Omega \wedge s(g) -/\!\!\rightarrow * s'(g))$$
$$\big((\exists \bar{s})(\alpha(\bar{s}, \xi) = N\bar{s} \wedge (g) \rightarrow * s'(g))\big).$$

Let us define the following assertions

$$R_{42}(\xi, s\ s^*) : (\forall s', s'(g) \neq \Omega \wedge \alpha(s', \xi) = \Omega \wedge s(g) \rightarrow s^*(g) \rightarrow * s'(g))$$
$$\big((\exists \bar{s}, \bar{s} \neq s)(\alpha(\bar{s}, \xi) = N \wedge \bar{s}(g) \rightarrow * s'(g))\big),$$
$$R_{43}(\xi, s) : (\forall s', s'(g) \neq \Omega \wedge \alpha(s', \xi) = \Omega \wedge s(g) \rightarrow * s'(g))$$
$$\big((\exists \bar{s}, \bar{s} \neq s)(\alpha(\bar{s}, \xi) = N \wedge \bar{s}(g) \rightarrow * s'(g))\big).$$

Lemma 3.1.4.

$$\{Q_3(\xi, s) \wedge R_{43}(\xi, s) \wedge \alpha(s, \xi) = Y\}$$
$$\underline{\text{process-name}}\ (s)$$
$$\{Q_3(\xi, s) \wedge R_{43}(\xi, s) \wedge \alpha(s, \xi) = N\} \qquad \text{(iii)}$$

and the proofs (i) (iii) are interference-free.

Proof. This is concluded from the definition of the macros proc and process-name immediately.

Assumption 3.1.3.

$$\{Q_3(\xi, s) \wedge \alpha(s, \xi) = N \wedge R_{42}(\xi, s, s^*)\}$$
$$\underline{\text{set}}\ (s^*)$$
$$\{Q_3(\xi, s) \wedge \alpha(s, \xi) = N \wedge s(g) \rightarrow s^*(g)\} \qquad \text{(iv)}$$

and the proofs (i), (iv) are interference-free.

It is obvious now, that

$$Q_3(\xi, s) \wedge \alpha(s, \xi) = N \wedge (\forall s^*, s(g) \rightarrow s^*(g))(R_{42}(\xi, s, s^*)) \supset$$
$$Q_3(\xi, s) \wedge \alpha(s, \xi) = N \wedge R_{43}(\alpha, s).$$

Hence using the rule of iteration it is easy to deduce the following proof:

$$\{Q_3(\xi, s) \wedge R_{43}(\xi, s) \wedge \alpha(s, \xi) = Y\}$$
$$\underline{\text{process-desc}}\ (w, s)$$
$$\{Q_3(\xi, s) \wedge \alpha(s, \xi) = N \wedge w = s\text{-desc}\,(s(g))\}.$$

But

$$Q_3(\xi, s) \wedge R_{43}(\xi, s) \wedge \alpha(s, \xi) = Y \supset Q_2(\xi, s) \wedge \alpha(\xi, s) = Y$$

that shows the correctness of Assumption 3.1.2.

Level 4.

Lemma 3.1.5.

$$\{Q_3(\xi, s) \wedge \alpha(s, \xi) = N \wedge R_{42}(\xi, s, s^*)\}$$
$$\underline{\text{link}}\ (s^*) \qquad \text{(V)}$$
$$\{Q_3(\xi, s) \wedge \alpha(s, \xi) = N \wedge s(g) \rightarrow * s^*(g) \wedge \alpha(s^*, \xi) = \Omega\}$$

and the proofs (i), (v) are interference-free.

Proof. This is a simple assignment statement. From the rule of assignment the correctness of Lemma 3.1.5 follows.

The macro set (s^*) is a conditional statement. Using the axiom of the null statement and the conditional rule, the correctness of Assumption 3.1.3 can be shown. The correctness of the assumptions used in the proof of the partial correctness of the walk algorithm is verified until now.

But the termination of the algorithm is still left. The set of the nodes of the graph is finite. Let N be the number of the nodes and let $\nu(\xi)$ be the number of selectors for which

$$\alpha(s, \xi) = Y.$$

Then

$$U_1(\xi) : N - \nu(\xi)$$

is an integer function and

$$R_1(\xi) \supset U_1(\xi) \geq 0.$$

Assumption 3.1.4.

$$\{R_1(\xi) \wedge U_1(\xi) < C\}$$
$$\underline{\text{process-node}}\ (n, s; \underline{\text{proc}})$$
$$\{R_1(\xi) \wedge U_1(\xi) = C\}$$

and the macro process-node terminates.

It is obvious that the macros next-name and next-node do not alter the values of the function $U_1(\xi)$, hence the rule of termination shows that the walk algorithm also terminates.

To prove the correctness of Assumption 3.1.4 let

$$U_2(w) : \text{length}\ (w)$$

where $w = s\text{-desc}\ (n)$ and is-node $(n) = T$. Now

$$U_2(w) \geq 0$$

and $U_2\ (\text{tail}\ (w)) < U_2(w)$ if length $(w) \geq 1$. Hence it is seen that the macro process-desc (w, s) terminates. But the macros proc and process-desc have

no common variables, therefore their parallel execution does not create an interference problem, that shows the correctness of Assumption 3.1.4.

In the following two special cases of the graph walk algorithm will be given by specifying the predicate is-pred-graph and the macro proc. These algorithms have their practical importance in system programming.

4. An abstract linkage editor

Consider a programming system, where the segments refer to each other only by the segment name. Then the graph walk algorithm can be applied for defining the linkage editor of this system as follows:

Let

$$\text{is-state} = (\langle s\text{-input:is-}r/b\text{-program}\rangle, \langle s\text{-table:is-value-set}\rangle, \langle s\text{-control:is-control}\rangle),$$

where

$$\text{is-}r/b\text{-program} = \text{is-segment-code-graph}$$

where

$$\text{is-select} = \text{is-segment-name}.$$

In detail:

$$\text{is-}r/b\text{-program} = (\{\langle s\text{-is-node}\rangle \mid \text{is-segment-name } (s)\})$$
$$\text{is-node} = (\langle s\text{-value:is-segment-code}\rangle, \langle s\text{-desc:is-segment-name-list}\rangle).$$

Definition 4.1. Let

$$\underline{\text{editor}}\ (t)$$

be the macro instruction that processes a segment-code as needed for linking. The actual operation is not relevant here.

Then the abstract linkage editor can be characterized by the machine, where the initial state ξ_0 is the following

$$\xi_0 = \mu_0(\langle s\text{-input:}p\rangle, \langle s\text{-table: }t_0\rangle, \langle s\text{-control: } \underline{\text{walk}}\ (p;\ \text{next-selector},\ \underline{\text{editor}})\rangle)$$

and specifically is-r/b-program$(p) = T$.

5. Appendix
Axioms and rules of consequence

We introduce the following notation

$$\{R\}$$
$$P$$
$$\{Q\}$$

where R and Q are assertions about variables and P is a VDL program. This may be interpreted as follows: if Q is true before execution of a VDL program P, then R is true after execution of P, provided that P terminates.

Axiom of assignment 5.1

$$\{P(a;\ \xi)\}$$

$$a : \underline{\text{ut}}\ (x_1, \ldots, x_k)$$

$$\{P(e_0;\ \mu(\xi;\ \langle s\text{-}c_1 : e_1'\rangle, \ldots, \langle s\text{-}c_n : e_n'\rangle))\}$$

where

$$\underline{\text{ut}}\ (x_1, \ldots, x_k) =$$

$$\text{PASS} : e_0(x_1, \ldots, x_k,\ \xi)$$

$$s - c_1 : e_1(x_1, \ldots, x_k;\ \xi)$$

$$\vdots$$

$$s - c_n : e_n(x_1, \ldots, x_k;\ \xi)$$

and

$$e_i' = e_i(x_1, \ldots, x_k;\ \xi) \quad i = 0, 1, \ldots, n.$$

Axiom of null statement 5.2

$$\{P\}$$

$$\underline{\text{null}}$$

$$\{P\}.$$

Defining inference rules in addition to axioms, Hoare [6] describes a deductive system for proving properties of sequential programs. This rules may be expressed formally:

$$\frac{X}{Y}$$

which means, if X is true, then Y is also true.

In the case of VDL-programs the following inference rules are accepted as axioms.

Rule of consequence 5.1

$$\frac{\begin{array}{l}\{q_1\}\\ \text{ut and } p \supset p_1 \text{ and } q_1 \supset q\\ \{p_1\}\end{array}}{\begin{array}{l}\{q\}\\ \text{ut}\\ \{p\}\end{array}}.$$

Rule of assignment 5.2

On the basis of the axiom of assignment, by means of the consequence rule we have

$$\frac{P(x_1, \ldots, x_k;\ \xi) \supset Q(e_0';\ \mu(\xi;\ \langle s - c_1 : e_1' \rangle, \ldots, \langle s - c_n : e_1' \rangle))}{\begin{array}{c}\{Q(a;\ \xi)\} \\ a : \underline{\mathrm{ut}}\ (x_1, \ldots, x_k) \\ \{P(x_1, \ldots, x_k;\ \xi)\}\end{array}}$$

where

$$\underline{\mathrm{ut}}\ (x_1, \ldots, x_k) =$$

$$\begin{array}{l}\mathrm{PASS} : e_0(x_1, \ldots, x_k;\ \xi) \\ s - c_1 : e_1(x_1, \ldots, x_k;\ \xi) \\ \quad\vdots \\ s - c_n : e_n(x_1, \ldots, x_k;\ \xi)\end{array}$$

and

$$e_i' = e_i(x_1, \ldots, x_k;\ \xi) \quad i = 0, 1, \ldots, n.$$

Conditional rule 5.3

$$\frac{\begin{array}{ccccc}\{q_1\} & & \{q_2\} & & \{q_n\} \\ \underline{\mathrm{ut}}_1 & \text{and} & \underline{\mathrm{ut}}_2 & \text{and} & \underline{\mathrm{ut}}_n \\ \{p \wedge p_1\} & & (p \wedge \neg p_1 \wedge p_2) & & \{p \wedge \neg(p_1 \wedge p_2 \wedge \ldots \wedge p_{n-1})\}\end{array}}{\begin{array}{c}\{q_1 \vee q_2 \vee \ldots \vee q_n\} \\ \underline{\mathrm{ut}} \\ \{p\}\end{array}}$$

where

$$\underline{\mathrm{ut}} =$$

$$\begin{array}{l}p_1 \rightarrow \underline{\mathrm{ut}}_1 \\ p_2 \rightarrow \underline{\mathrm{ut}}_2 \\ \\ \\ T \rightarrow \underline{\mathrm{ut}}_n .\end{array}$$

Rule of iteration 5.4

If

$$\underline{\text{ut}}\,(x_1, \ldots, x_n) =$$

$$p_1(x_1, \ldots, x_n;\ \xi) \rightarrow \underline{\text{ut}}\,(z_1, \ldots, z_n);$$

$$\underline{\text{ut}}_1\,(a_1, \ldots, a_k)$$

$$T \rightarrow \underline{\text{null}}$$

where

$$a_i = a_i(x_1, \ldots, x_n;\ \xi) \qquad i = 1,2, \ldots, k$$

$$z_j = z_j(x_1, \ldots, x_n;\ \xi) \qquad j = 1, 2, \ldots, n$$

and

$$p(x_1, \ldots, x_n;\ \xi) \wedge p_1(x_1, \ldots, x_n;\ \xi) \supset q(x_1, \ldots, x_n, a_1, \ldots, a_k;\ \xi),$$

and

$$\{r(x_1, \ldots, x_n:) \wedge R(a_1, \ldots, a_k)\}$$

$$\underline{\text{ut}}_1(a_1, \ldots, a_k)$$

$$\{q(x_1, \ldots, x_n,\ a_1, \ldots, a_k)\}$$

furthermore

$$r(x_1, \ldots, x_n;\ \xi) \supset p(z_1, \ldots, z_n;\ \xi)$$

then

$$\{p(w_1, \ldots, w_n;\ \xi) \wedge \neg p_1(w_1, \ldots, w_n;\ \xi) \wedge Q\}$$

$$\underline{\text{ut}}\,(x_1, \ldots, x_n;\ \xi)$$

$$\{p(x_1, \ldots, x_n;\ \xi)\}$$

where

$$Q : R(a_1, \ldots, a_k) \wedge R(b_1, \ldots, b_k) \wedge \ldots \wedge R(c_1, \ldots, c_k)$$

and the transformation of the argument of $\underline{\text{ut}}$ is

$$(x_1, \ldots, x_n) \rightarrow (z_1, \ldots, z_n) \rightarrow \ldots \rightarrow (v_1, \ldots, v_n) \rightarrow (w_1, \ldots, w_n)$$

and the arguments of $\underline{\text{ut}}_2$ during the iteration are

$$(a_1, \ldots, a_k), (b_1, \ldots, b_k), \ldots, (c_1, \ldots, c_k).$$

For example, if

$$\underline{\text{ut}}\,(t) =$$
$$\text{length}\,(t) \neq 0 \rightarrow \underline{\text{ut}}\,(\text{tail}\,(t));$$
$$\underline{\text{ut}}_1\,(\text{head}\,(t))$$
$$T \rightarrow \underline{\text{null}}$$

and

$$p(t) : \text{is-pred-list}(t)$$
$$t = \langle a, b, \ldots, c \rangle$$

then

$$\{p(w) \wedge \text{length}\,(w) = 0 \wedge R(a) \wedge R(b) \wedge \ldots \wedge R(c)\}$$
$$\underline{\text{ut}}\,(t)$$
$$\{p(t)\}$$

Rule of composition 5.5

$$\frac{\begin{array}{cc} \{q\} & \{r\} \\ \underline{\text{ut}}_2 \text{ and } & \underline{\text{ut}}_1 \\ \{p\} & \{q\} \end{array}}{\begin{array}{c} \{r\} \\ \text{ut} \\ \{p\} \end{array}}$$

where

$$\underline{\text{ut}} =$$
$$\underline{\text{ut}}_1;$$
$$\underline{\text{ut}}_2$$

5.1 Definition. Given a control tree t, with the statement

$$\{q\}$$
$$\underline{\text{t}}$$
$$\{p\}$$

and the value returning instruction ut with the precondition pre ut. If the execution of ut after t does not alter the validity of q, that is

$$\{q\}$$
$$\underline{\text{ut}}$$
$$\{\text{pre(ut)} \wedge q\}$$

and the execution of $\underline{ut}$ before any $\underline{u}$ within $\underline{t}$ does not alter the validity of pre ($\underline{u}$), that is

$$\{\text{pre}(\underline{u})\}$$
$$\underline{ut}$$
$$\{\text{pre}\,(\underline{ut}) \wedge \text{pre}\,(\underline{u})\}$$

then we say that $\underline{ut}$ does not interfere with

$$\{q\}$$
$$\underline{t}$$
$$\{p\}$$

Definition 5.2. Given the statement

$$\begin{array}{lll} \{q_1\} & \{q_2\} & \{q_n\} \\ \underline{ut}_1, & \underline{ut}_2, \ldots, & \underline{ut}_n \\ \{p_1\} & \{p_2\} & \{p_n\} \end{array} \qquad \text{(i)}$$

Let $\underline{u}_i$ be a value returning instruction within $\underline{ut}_i$. If for all i, $i = 1, 2, \ldots, n$ $\underline{u}_i$ does not interfere with

$$\{q_.\}$$
$$\underline{ut}_j \quad j = 1, 2, \ldots, n; j \neq i$$
$$\{p_j\}.$$

then we say that the statements (i) are interference free.

Rule of parallelism 5.6

$$\frac{\begin{array}{l}\{q_1\} \\ \underline{ut}_1 \\ \{p_1\}\end{array} \text{ and } \begin{array}{l}\{q_n\} \\ \underline{ut}_n \\ \{p_n\}\end{array} \text{ and they are interference free}}{\begin{array}{c}\{q_1 \wedge q_2 \wedge \ldots \wedge q_n\} \\ \underline{ut} \\ \{p_1 \wedge p_2 \wedge \ldots \wedge p_n\}\end{array}}$$

where

$$\underline{ut} =$$
$$\underline{ut}_1,$$
$$\underline{ut}_2,$$
$$\vdots$$
$$\underline{ut}_n.$$

Termination

Definition 5.3. Let

$$\underline{\mathrm{ut}}\,(x_1, \ldots, x_n; \xi) =$$
$$p_1(x_1, \ldots, x_n; \xi) \rightarrow \underline{\mathrm{ut}}\,(z_1, \ldots, z_n); \qquad (ii)$$
$$\underline{\mathrm{ut}_1}\,(a_1, \ldots, a_k)$$
$$T \rightarrow \underline{\mathrm{null}}$$

and let $\underline{\mathrm{u}}$ be a value returning instruction. Let $u(x_1, \ldots, x_n; \xi)$ be an integer function which is used at the verification of the termination of $\underline{\mathrm{ut}}$. If for all $x_1, \ldots, x_n, \xi$ we have

$$\{u(x_1, \ldots, x_n; \xi') \leqq u(x_1, \ldots, x_n; \xi)\}$$
$$\underline{\mathrm{u}}$$
$$\{\mathrm{pre}\,(\underline{\mathrm{u}}) \wedge u(x_1, \ldots, x_n; \xi) \geqq 0\}$$

then we say that the execution of $\underline{\mathrm{u}}$ does not interfere with the termination of $\underline{\mathrm{ut}}$.

Rule of termination 5.7

Let be given the loop $\underline{\mathrm{ut}}\,(x_1, \ldots, x_n; \xi)$ of the form (ii) and let

$$\mathrm{pre}\,(\mathrm{ut}(x_1, \ldots, x_n)) : p(x_1, \ldots, x_n; \xi)$$

be its precondition. Let

$$u(x_1, \ldots, x_n; \xi)$$

be an integer function. If

1. $p(x_1, \ldots, x_n; \xi) \supset u(x_1, \ldots, x_n; \xi) \geqq 0$
2. $p(x_1, \ldots, x_n; \xi) \wedge p_1(x_1, \ldots, x_n; \xi) \supset u(x_1, \ldots, x_n; \xi) > 0$
3. $\{u(x_1, \ldots, x_n; \xi') \leqq u(x_1, \ldots, x_n; \xi)\}$
   $\underline{\mathrm{ut}_1}\,(a_1, \ldots, a_k)$
   $\{u(x_1, \ldots, x_n; \xi) \geqq 0\}$
4. $u(z_1, \ldots, z_n; \xi') < u(x_1, \ldots, x_n; \xi)$
5. there is no such a value returning instruction in parallel execution of $\underline{\mathrm{ut}}$ which interferes with the termination of $\underline{\mathrm{ut}}$ then the execution of $\underline{\mathrm{ut}}$ terminates.

For example, if

$$\underline{\mathrm{ut}}(t) =$$
$$\mathrm{length}\,(t) \neq 0 \rightarrow \underline{\mathrm{ut}}\,(\mathrm{tail}\,(t));$$
$$\underline{\mathrm{ut}_1}\,(\mathrm{head}\,(t))$$
$$T \rightarrow \underline{\mathrm{null}}$$

and

$$\text{pre}\,(\underline{\text{ut}}(t)) : \text{is-pred-list}(t),$$

$$u(t) = \text{length}\,(t)$$

then

1. is-pred-list $(t) \supset u(t) \cong 0$
2. is-pred-list $(t) \wedge \text{length}(t) \neq 0 \supset u(t) > 0$
3. $u(t)$ does not depend on ξ, therefore the execution of $\text{ut}_1(\text{head}\,(t))$ does not alter its value
4. $u(\text{tail}(t)) < u(t)$
5. $u(t)$ contains only a local variable, therefore the execution of any value returning instruction does not alter its value

hence the loop $\underline{\text{ut}}$ terminates.

6. Conclusion

We used the VDL-language both as a language and as a program development methodology to teach the students for designing programs at the Eötvös Loránd University in Budapest. As a teaching aid, the language helps the students to grasp the common features of different programs meeting the same specification and reduce the complexity of the proof of correctness.

It can also be used as a tool for defining abstract algorithms, as illustrated in the paper. Such algorithms or verified abstract programs can be implemented on a given hardware and software environment.

REFERENCES

[1] *Lucas P., Lauer P., Stiegleitner H.:* Method and notation for the formal definition of programming languages. IBM Lab. Vienna, TR 25.087, 1968.

[2] *Lucas, P. Walk K.:* On the formal description of PL/1. Annual Review in Automatic Programming **6.** 105 – 182 (1969).

[3] *Neuhold E. J.:* The formal description of programming languages. IBM Systems J., **10.** (1971).

[4] *Lee, J. A. N.:* Computer semantics. Van Nostrand Reinhold Co., New York, 1972.

[5] *Wegner P.:* The Vienna definition language. Comput. Surveys *4.* 5 – 63 (1972).

[6] *Hoare, C. A. R.:* An axiomatic basis for computer programming. Comm. ACM **12.** 576 – 580 (1969).

[7] *Owicki, S., Gries, D.:* Axiomatic proof techniques for parallel programs. Acta Informatica **6.** 319 – 340 (1976).

Eötvös Loránd's University
Department of Numerical Analysis and Computer Science
1088 Budapest, Múzeum krt. 6 – 8.