A Final Comment Regarding
“An Alternative Control Structure and its Formal Definition”*

David Lorge Parnas William W. Wadge


Meaning is explicitly assigned only to programs in which a guard is only true in states within the competence set of the program that it limits. In other words, for the program $g \rightarrow p$, $g$ must not be true in any state where $p$ might not terminate. Although this seems reasonable at first glance, it requires that guards test the truth of conditions that can be proven true in the context in which the limited program appears. For example, if we had a guard for a square-root program, it would have to check that an argument is not negative even if it appeared at a point in the program where the argument would always be positive. Programs without such unnecessary tests are meaningless according to the formal semantics given in the paper. Several such “meaningless” programs appear in the paper.

The restriction was introduced because of a fundamental limitation on implementations of lists as defined in the paper. Implementations cannot be expected to ascertain the competence set of a program. They must proceed as if the guard identifies that set. Abortion is possible if the programmer does not use guards properly. However, some simple changes allow us to define the semantics completely and without requiring the superfluous tests.

Rather than require that the guard $g$, of $g \rightarrow p$, only be true in states that are in the competence set of $p$, we can define the competence set of $g \rightarrow p$ to be the intersection of the set of states in which $g$ is true and the competence set of $p$. In the notation of [1],

$$C_{g \rightarrow p} = \{x : g(x)\} \land C_L^p$$

It is also useful to define the set $F$ for $g \rightarrow p$ to be the set of states $s$ such that $s$ is not in the competence set of $p$, and yet $g$ is true.

$$F_{g \rightarrow p} = \{x : g(x)\} \land \neg C_L^p$$

The competence set of the limited program list $g_1 \rightarrow p_1 \lor g_2 \rightarrow p_2 \lor \cdots \lor g_n \rightarrow p_n$ includes a state if and only if it is in the competence set of a component and not in the set $F$ of any component.

$$C_{g_1 \rightarrow p_1 \lor g_2 \rightarrow p_2 \lor \cdots \lor g_n \rightarrow p_n} = \{x : g_1(x) \lor g_2(x) \lor \cdots \lor g_n(x)\} \land \neg (F_1 \cup F_2 \cup \cdots \cup F_n),$$

where $F_i$ is $F_{g_i \rightarrow p_i}$.

The relation of the limited program list is as defined in the original paper. Jury lists are handled in the same way.

Because of these changes, it is necessary to modify the definition of $L_0$ and $L'$ in the definition of the it–ti construct.

Let $L^*$ be the LD-relation of the limited program list comprising the components that are marked with a ↓. Let $L^{**}$ be the LD-relation of the limited program list comprising the components that are marked with an ↑. We define $L_0$ and $L'$ as follows:

1. $C_{L_0} = C_{L^*} \land \neg C_{L^{**}} \land \neg F^{**}$, where $F^{**}$ is the union of the $F$’s for the limited programs marked with an uparrow;

*Published in CACM 27(5):499,522, May 1984, Technical Correspondence.
2. $R_{L_0} = R_{L'}$;

3. $C_{L'} = C_{L^*} \land \neg F^*$, where $F^*$ is the union of the $F$'s for the limited programs marked with a downarrow; and

4. $R_{L'} = R_{L^*}$.

$L_0$ and $L'$ are used to define $L_l$ and $L_{it\text{-}ti}$ as in the original paper.

The only disadvantage of this change is that one can no longer compute the LD-relation of a limited program list or an it-ti knowing only the LD-relation of the limited program it comprises. It is necessary to know the guards or the $F$'s as well. This is not a serious hindrance either in theory or in any practical effort to develop easily understood programs. We can simply regard a limited program list or it-ti as being constructed of programs and guards rather than from limited programs. The semantics of a program remains independent of the context in which it appears. The revised semantics is complete, that is, there are no meaningless programs. It corresponds to the machine behavior for any machine with a finite number of states. It supports a programming methodology in which each constructed program can be replaced by its LD-relation and treated as a “black box”. We consider that an essential property of a programming notation and its definition.

References
