KOWALSKI79
Kowalski, R;
Algoithm = Logic + Control;
Communications of the ACM (July 1979) pp. 424-436.

"An Algorithm can be regarded as consisting of a logic component, which specifies the knowledge to be used in solving problems, and a control component, which determines the problem-solving stategies by means of which that knowledge is used. The logic component determines the meaning of the algorithm whereas the control component only affects its efficiency."

This paper is a self-contained article which describes the need and advantages of separating logic and control. Predicate logic is presented as the tool for logical analysis and various control strategies are examined.

The separation of the logic component (L) from the control component (C) in an algorithm (A) is desirable because it gives us a clearer and more obvious statement of the problem and its associated knowledge, and allows us to easily change algorithm efficiency by improving only the control. In both cases it becomes more obvious what the program does, how it does it and whether it is correct.

Symbolically if A=L+C and A'=L+C' then algorithms A and A' are equivalent since the logic component is unchanged, though the control components are different. This gives the basis for logic analysis.

Alternatively we could have a single algorithm A where A=L+C and A=L'+C'. This could represent a simple logic component L and a complex efficient control C, or a complex logic L' and a simple control C', both yielding the same result.

The actual logical analysis is done using the clausal form of the predicate logic. This reduces the logic component to clauses and assertions of the following form:

```
Father(Zeus, Areas) <-              (data, assertion)
Father(Ares,Harmonia) <-
Parent(x,y) <- Father(x,y)          (data relationship)
Grandparent(x,y) <- Parent(x,z),Parent(z,y)
```

"Horn Clauses" (predicate logic with at most one conclusion) are used because they more closely resemble conventional programming languages. Symbolically we have four kinds:

```
B <-                    (assertions)
B <- A1,A2,...,An       (procedure definitions)
<- C1,C2,...,Cn         (denials or conclusions)
<-                      (contradiction)
```

In general there are two methods of problem solution. Top-down solution involves backward reasoning from the theorem (conclusion) repeatedly reducing goals to subgoals until the original assertions are

reached. In bottom-up solutions we reason forward from the assertions, deriving new assertions until we reach the conclusion.

Given a conclusion <- C1,C2,...,Cn each Ci can be solved using either top-down or bottom-up logic (or some combination of both). Since the order of the Ci's is unimportant computation can often be performed in parallel. (Part of Kowalski's article introduces a graphical representation of this control process). This control component could be specified in a separate language allowing programmer experience to improve efficiency, or be executed automatically by the system as it is done by the PROLOG language. Alternative strategies of bottom-up vs. top-down logic are examined by Kowalski for a number of problems.

Kowalski maintains that in designing an algorithm the logic component should be specified first. Not only does this define the problem in an understandable manner but more sophistication in this area can reduce the complexity of the control component. As higher level programming languages evolve, and as the expertise of the average user decreases, the mor the system will need to assume responsibility for efficiency.

Separation of logic and control would create more easily adaptable, correct programs. While this separation is becoming more common in database systems, current programming languages still do not make this distinction.