SYNTACTIC INDUCTIVE SYNTHESIS USING EXAMPLES OF PROGRAMS WITH
CONJUNCTIVE CONDITIONS IN LOOPS

E. B. Kinber

A formal model of inductive syntactic synthesis is proposed for programs with con-
junctions of \( x \leq y \) predicates in loop conditions. A synthesis algorithm using an-
notated examples is developed for such programs. The algorithm is shown to be
polynomial in the length of the input examples and its correctness is proved.

A number of recent studies have focused on program synthesis using examples of programs
which in one way or another are connected with fragments of arithmetic progressions \([1-4]\).
Synthesis methods are usually purely syntactic: an input example is treated as a symbolic
string in which various syntactic analogies are sought. This in particular makes the syn-
thesizer essentially independent of the syntax chosen by the user for the description of the
algorithm. At the same time, purely syntactic tools in general are insufficient for the syn-
thesis of loops and conditional statements in the form in which they appear in real pro-
grams — the synthesis usually produces a "proxy" of these statements (multipoint expressions
in \([1]\), generalized regular expressions in \([4]\)), which can be converted into a corresponding
ordinary statement under a certain interpretation of the program.

However, in many algorithms (e.g., all the algorithms with FOR loops, a number of algo-
rithms with WHILE loops, such as merge algorithms for ordered arrays) the logical conditions
in the loop operators relate only to variables that take natural values and change according
to an arithmetic progression. These algorithms in principle admit a purely syntactic syn-
thesis of loops in the ordinary WHILE form, although in general this involves considerable,
at least enumerative, difficulties. In this paper, we propose a syntactic algorithm of syn-
thesis by examples for such algorithms which is polynomial (in the length of the input ex-
amples) in the case when the logic conditions in the loops are predicates of the form \( x \leq y \),
or conjunctions of these predicates. The conditional statements are synthesized in some gen-
eral form without separating the logical conditions from actions (instead of the usual IF...
THEN...ELSE and CASE we use the "neutral" union \( U \)). The following input examples are used in
order to elucidate the behavior of the algorithms.

Example 1. Example of computation for an algorithm of addition of two numbers.

\[
\begin{align*}
\text{INPUT}(x, y) \\
x &:= 4, y := 3 \\
ad\text{d} &\text{d} 1 t\text{r}o 3; \text{subtract} 1 f\text{r}om 4; \\
ad\text{d} &\text{d} 1 t\text{r}o 4; \text{subtract} 1 f\text{r}om 3; \\
ad\text{d} &\text{d} 1 t\text{r}o 5; \text{subtract} 1 f\text{r}om 2;
\end{align*}
\]
add 1 to 6; subtract 1 from 1;
OUTPUT (7).

Example 2. Example of computation for an algorithm merging two ordered arrays. Given are the arrays \( A = A(1), A(2), \ldots, A(x_1) \) and \( B = B(1), B(2), \ldots, B(x_2) \). It is required to merge them into an ordered array \( C = C(1), C(2), \ldots, C(x_1 + x_2) \).

\[
\begin{align*}
\text{INPUT} & \quad (x_1, x_2) : x_1 = 3, x_2 = 5 \\
A(1) \leq B(1)? & \; \text{yes}; \; \text{then } C(1) = A(1) \\
A(2) \leq B(1)? & \; \text{yes}; \; \text{then } C(2) = A(2) \\
A(3) \leq B(1)? & \; \text{no}; \; \text{then } C(3) = B(1) \\
A(3) \leq B(2)? & \; \text{no}; \; \text{then } C(4) = B(2) \\
A(3) \leq B(3)? & \; \text{yes}; \; \text{then } C(5) = A(3) \\
C(6) & = B(3) \\
C(7) & = B(4) \\
C(8) & = B(5) \\
\text{OUTPUT} & \; (C).
\end{align*}
\]

For this synthesizer, the examples are formal -- the semantics of the expressions "?", "add 1", etc., is ignored during synthesis. Under certain interpretation of expressions of this kind, formal examples represent actual computations of the corresponding programs. Input examples clearly should satisfy certain conditions which make it possible to identify the values of the same variables in the example. Some sufficient conditions of this kind are proposed.

1. CLASS OF PROGRAMS

First let us define a special programming language which is convenient for syntactic synthesis of this class of programs.

Let \( N \) be the set of all natural numbers (including 0), \( A \) a finite alphabet, \( X \) the set of variable symbols (possibly indexed), \( X \subseteq A \cup N \). Let the set \( OP \) contain the following operation symbols: := (assignment), U (union), WHILE(...)DO(...) (loop), - (subtract 1), + (add 1), and also parentheses (, ) and comma. We assume that \( OP \cap (X \cup A) = \emptyset \), and also use \( Pr \) -- the set of predicates \( x \leq y, x \geq y \) for any \( x, y \in X \) and all possible simple conjunctions of these predicates with pairwise different variables [e.g., \( (x \leq x_1) \land (y \leq y_1) \)].

The expressions \( x, x^+, x^- \) for an arbitrary \( x \) are called \( x \)-atoms (sometimes simply atoms). The atoms \( x^+ \) and \( x^- \) are essentially a convenient form of writing the statements \( x := x + 1 \) and \( x := x - 1 \).

For an arbitrary \( x \in X \), the \( x \)-projection of the word \( \alpha \) in the alphabet \( A \cup X \cup N \cup OP \) is the sequence of \( x \)-atoms in \( \alpha \) that occur outside the expressions \( x := \mu \) and \( \mu := x \).

Program Body (p.b.).

1. If \( \mu \) is an atom, then \( \mu \) is a p.b.
2. If \( a \in A \), then \( a \) is a p.b.
3. If \( P \) and \( R \) are p.b.s, then \( PR \) is p.b.
4. If \( P_1, P_2, \ldots, P_k \) are p.b.s and \( S \in Pr \), then WHILE (S)DO (P_1 U P_2 U \ldots U P_k) is a p.b.
5. If \( P \) is a p.b., \( x_1, x_2, \ldots, x_k \in X, \mu_1, \mu_2, \ldots, \mu_k \in X \cup \{0, 1\}, \) then \( (x_1 := \mu_1, x_2 := \mu_2, \ldots, x_k := \mu_k) P \) is a p.b.
6. There are no other p.b.s.

For an arbitrary word \( \alpha \), let \( X(\alpha) \) be the set of variables in \( \alpha \).

The fragments WHILE(...) DO(...) in a p.b. are called loops. For some loop \( R = \text{WHILE}(S) \text{DO}(T_1 U T_2 U \ldots U T_h) \), let each \( T_i \) be representable in the form

\[
A_0^i B_1^i \ldots B_{m_1}^i A_{m_1}^i (1.1)
\]

where all \( B_j^i \) are loops and \( A_j^i \) contain no loops.

The variable \( x \) is called the main variable in the loop \( R \) if at least for one \( i \):

a) the \( x \)-projection of \( A_0^i \ldots A_{m_1}^i \) is nonempty;

b) \( A_0^i \ldots A_{m_1}^i \) have no occurrences of the statements \( (x := \mu) \);