

ON SYNTAX-DIRECTED, SEMANTICS-SUPPORTED PROGRAM SYNTHESIS

Wolfgang Bibel
Institut für Angewandte Informatik und Formale Beschreibungsverfahren
Universität Karlsruhe
on leave from Institut für Informatik
Technische Universität München

A number of strategies for the synthesis of algorithms from a given input-output specification of a problem are presented which are centered around a few basic principles. It has been verified for more than ten different algorithms that their uniform application in all cases results in a successful deductive synthesis. They include a spanning-tree algorithm, a graph-circuits algorithm, a finding-the-ith-smallest-element algorithm, and a linear (string) pattern-matching algorithm. Two of them are presented here.

INTRODUCTION

This paper is based on the view that (human or mechanical) programming is a deductive process which starts with a more or less detailed specification of the given problem in some representation language and after some search guided by several strategies eventually ends up with a deduction of the program which consists in the application of rules taken from logic, the underlying theory and from a knowledge base for programming skill. It also adopts the view that predicate logic (in a wider sense) so far still is the most suitable representation language for studying this process, because of its conciseness, naturalness, flexibility, extendibility - both descriptive problem specification and machine programs can be represented in logic - and in particular because of the well-understood deductive mechanisms for it.

Specifically, the topic of this paper is the search for such a deduction from a logical description of a problem to some equivalent *functional* form (be it in logic or in an Algol-like language). It is no secret that the search-tree for such a deduction is tremendous even for rather simple problems. Yet, it is a common observation that people are able to move in a rather direct way towards a solution even for problems that require knowledge not experienced before. There must be general guidelines or strategies, then.

Unfortunately, (no, fortunately!) our human thinking is not organized such that these strategies could be easily bootstrapped from our be-

haviour. Therefore the only way to detect general strategies seems to be by a detailed comparative study of different deductions of different problems in order to detect common mechanisms. This has been done by the author for the usual maximum problem and for Hoare's FIND problem resulting in a few principles in the form of strategies [6,9]. The surprising experience *after* this study was that these strategies proved successful for a number of different other problems. They include such interesting algorithms like a linear (string) pattern-matching algorithm [16], a spanning tree algorithm [1], and a graph-circuits algorithm [20].

The strategies center around four basic ideas how to deal with a predicate logic formula describing a given problem: an $\exists$ -quantifier together with a non-functional specification requires some kind of guessing (see GUESS); two (or more) consecutive $\exists$ -quantifiers require the evaluation of the dependency of the respective variables inherent in the specification (see GET-RNV,-SOC,DEPEND); for both requirements the cardinality of certain problem inherent domains and other semantic knowledge provide helpful strategic information (see DOMAIN and CHVAR); the rewriting process can be strongly goal-oriented with several intermediate goals (see GET-G). This uniform and concentrated strategic concept seems to distinguish this approach from other work done in this field like [11-15,19, etc.]. Specifically, no other approach offers both, a uniform scheme based on a few basic principles, and a guidance of the synthesis to such an extent that there is some reason for the

claim that the automatic synthesis of programs like those in this paper seems to become feasible. Of course, the second claim could only be fully verified by a running system (which at present does not exist). But the text has been formulated with such a system in mind and several details have been included in order to support that claim.

In the following the basic strategies are introduced and explained by application to the usual maximum problem. Exactly the same sequence of strategies mechanically produces the deduction of a linear pattern matching algorithm which for reasons of space is presented in a very packed form (see [7] for more details).

BASIC STRATEGIES

Logically, a programming problem has the form $\forall i \exists o (IC(i) \rightarrow OC(i, o))$, i, o, IC, OC abbreviating input, output, input-condition, output-condition, resp. For example, the usual maximum problem in this format reads

1.1. $\forall S \exists m (S \neq \emptyset \rightarrow m \in S \wedge S \leq m)$, shortly $\forall S \exists m \text{ MAX}(S) = m$,
where $S \leq m: \leftrightarrow \forall x (x \in S \rightarrow x \leq m)$.

Neither $m \in S$ nor $S \leq m$ (nor both) provide a way how to determine m for given S , not even partially. Therefore, in view of an algorithmic solution some trial-and-error method cannot be avoided. In this simple example we only can try for a hopefully correct m . Apparently it is wise to restrict such attempts somehow, specifically by requiring a subset of the conditions listed in the output-condition. The subset has to be proper since otherwise no reduction of the problem would be possible. Formally, this idea in the case of our example reads

1.2. $\forall S \forall m' \exists m (m \in S \rightarrow (S \neq \emptyset \rightarrow m \in S \wedge S \leq m \wedge (m' \neq m \vee m' = m)))$, or
1.3. $\forall S \forall m' \exists m (S \leq m' \rightarrow (S \neq \emptyset \rightarrow m \in S \wedge S \leq m \wedge (m' \neq m \vee m' = m)))$

The underlying general strategy which is of a purely syntactic nature is the following (with $\dot{\vee}$ denoting exclusive disjunction).

GUESS: Transform a problem of the form
 $\forall i \exists y (IC(i) \rightarrow OC(i, y))$ to
 $\forall i \forall y' \exists y (ds \rightarrow (IC(i) \rightarrow OC(i, y) \wedge (y \neq y' \vee y = y')))$
where domain-specification (ds) is a conjunction of a proper subset of the conjuncts in $OC(i, y')$

Without any additional support the synthesis would now have to consider in our example two (in general as many as there are proper subsets of clauses in $OC(i, y')$) different cases, (1.2) and (1.3) possibly leading to different algorithms. In order to reduce the search space, a semantical argument in favour of one of those cases would be desirable. In our example such an argument can easily be seen, favouring (1.2)

rather than (1.3) for the case of usual assumptions on the domain of S and its representation. Namely, if we keep in mind that m' has to be chosen to satisfy the domain-specification then apparently this choice can be performed immediately for $m' \in S$ while it would require a number of steps for $S \leq m'$. Moreover, if S is finite then in the first case the guess also is more likely to be correct than in the second one since $|S| < |\{m' \mid S \leq m'\}| = \infty$. These considerations can be generalized in the following strategy which requires semantic knowledge for its application.

For a problem as described in GUESS consider the set P of subsets of those conjuncts in $OC(i, y)$ which specify y . By conjuncting the elements of each of these subsets P becomes a set $\hat{P}$ of formulas. For each $C \in \hat{P}$ let $q_C(i) := |\{y \mid C \text{ holds}\}|$, and $s_C(i)$ be the number of steps required to determine an arbitrary y such that C holds. Then for a given weight-factor k and for $C, D \in \hat{P}$ let $C <_k D$ iff $q_C \cdot s_C < k \cdot q_D \cdot s_D \wedge q_D \cdot s_D \leq k \cdot q_C \cdot s_C$. With this priority-relation which captures the considerations illustrated above in a general form, the strategy reads as follows.

DOMAIN: For domain-specification in GUESS choose a minimal $C \in \hat{P}$ w.r.t. $<_k$ (which has not been considered so far).

Apparently, all the information necessary to apply DOMAIN has to be provided to the system from outside (interactively or via a knowledge base). Of course, application of DOMAIN to (1.1) yields $m' \in S$ as domain-specification, thus excluding (1.3) in a first attempt to synthesize MAX.

In the following we will see that (1.2) implicitly already contains an algorithmic solution for MAX which only has to be unfolded by re-writing the formula in a rather constrained goal-oriented way. The general strategic scheme and three of its instances are the following ones.

GET-G (goal-oriented equivalence transformation with goal G): Rewrite a given formula according to domain and goal dependent equivalence transformation rules until goal G is achieved.

GET-DNF: the goal is that the resulting formula is in disjunctive normal form (alternative cases).

GET-REC: the goal is to find some kind of recursion; if necessary, generalize the problem

GET-EP: the goal is that predicates become evaluable, at least recursively.

In a program we need alternative cases. Therefore GET-DNF will be applied to the conclusion

in (1.2) which yields

$$1.4. m' \in S \rightarrow (S \neq \emptyset \rightarrow m \in S \wedge S \leq m \wedge m \neq m') \\ \vee (S \neq \emptyset \rightarrow m \in S \wedge S \leq m \wedge m = m')$$

(often we drop the quantifiers and some parentheses if they are understood, where $\wedge$ binds more than $\vee$, both more than $\rightarrow$, etc.)

Next we look for some kind of recursion for the first alternative (in the second one the result is already determined). The formula talks on two objects m, m' and on a set S . Imagine a recursion generator which from a knowledge base produces the different possible recursive forms of problem reduction for these kinds of objects. It is just reduction of cardinality in the present case of a set (but even in general there are only a few known possibilities). A rule generator (knowledge base and/or theorem prover) tries to merge the newly derived information ($m \neq m'$ in 1.4) by rewriting the conjunction of these clauses on the basis of the input-condition and the underlying theory (a search limited by a function in the number of given clauses even in the exhaustive mode). Here this might start with the application of the rule $m' \neq S \rightarrow (m \in S \wedge m \neq m' \leftrightarrow m \in S - m' \wedge m \neq m')$. Whenever in the result there is a match with one of the alternative recursive forms ($m \in S - m'$ in this case), this suggests a substitution possibly leading to a recursive problem reduction. It is explored for the whole problem under the guidance of a theorem prover. Because of $m' \in S \rightarrow (S \leq m \wedge m \neq m' \leftrightarrow S - m' \leq m \wedge m \neq m' \leftrightarrow S - m' \leq m \wedge m' < m)$, this process yields

$$1.5. m' \in S \rightarrow (S - m' \neq \emptyset \vee S = \{m'\} \rightarrow m \in S - m' \wedge S - m' \leq m \wedge m' < m) \vee \text{MAX}(S) = m' \\ 1.6. m' \in S \rightarrow (S - m' \neq \emptyset \vee m \in S - m' \wedge S - m' \leq m \wedge m' < m) \wedge S \neq \{m'\} \vee \text{MAX}(S) = m' \\ 1.7. m' \in S \rightarrow \text{MAX}(S - m') = m \wedge m' < m \wedge S \neq \{m'\} \vee \text{MAX}(S) = m'$$

completing the efforts of GET - REC in this case.

There is still the unpleasant clause $m' < m$ which cannot be executed, not even recursively. This causes GET - EP to rewrite (1.7) which gives the equivalent formula

$$1.8. m' \in S \rightarrow \text{MAX}(S - m') = m \wedge m' < \text{MAX}(S - m') \wedge S \neq \{m'\} \vee \text{MAX}(S) = m'$$

All these 8 formulas are logically equivalent. The last one can easily (and automatically) be compiled into efficient algorithmic code (see [7] for some more details). Note that the deduction also serves as a correctness proof for the resulting algorithm.

A CHALLENGING PROBLEM

For the general case a few additional strategies are required. But even with these basic ones quite a few non-trivial algorithms can be synthesized by mechanical application of the basic and standard sequence GUESS, DOMAIN, GET-DNF,

-REC, -EP. Linear string-pattern-matching is among these algorithms, for the development of which several outstanding researchers in computer science have been awarded with honour.

We denote by $\mathbb{N}_m^n$ the natural numbers k with $m \leq k \leq n$.

A string is a function $c_S: \mathbb{N}_1^{|S|} \rightarrow \langle \text{characters} \rangle$. Further assume fct as a built-in test for functionality, dom as the function which denotes the domain of a function. Then the definition of the pattern-matching problem can be stated as:

$$2.1. \forall c_S \forall c_P \exists k (\text{fct } c_S \wedge \text{dom } c_S = \mathbb{N}_1^n \wedge \text{fct } c_P \wedge \text{dom } c_P = \mathbb{N}_1^m + \\ k \in \mathbb{N}_0^{n-m} \wedge \forall j (j \in \mathbb{N}_1^m \rightarrow c_P(j) = c_S(j+k)))$$

shortly $\forall c_S \forall c_P \exists k (\text{PM}(c_S, c_P) = k)$

GUESS, DOMAIN, GET-DNF, -REC yield

2.2.

$$\forall c_S \forall c_P \forall k' \exists k (k' \in \mathbb{N}_0^{n-m} \rightarrow (\text{IC} \vee \text{OC} \wedge k = k') \vee (\text{IC} \wedge k \in \mathbb{N}_0^{n-m} \setminus k' \wedge \\ \forall j (j \in \mathbb{N}_1^m \rightarrow c_P(j) = c_S(k+j)) \wedge \exists \bar{j} (\bar{j} \in \mathbb{N}_1^m \wedge c_P(\bar{j}) \neq c_S(k' + \bar{j})))$$

GET-EP recognizes a subproblem determined by $\exists j$ which has to be solved prior to the solution of the whole problem. This subproblem can be treated in isolation since there is no clause containing both output-variables k and $\bar{j}$; a solution - the well-known quadratic algorithm - is easily and mechanically obtained with the standard technique of this paper (see [8]).

The linear solution can be obtained only if this treatment in isolation is abandoned or, more precisely, if the range of the $\exists \bar{j}$ -quantifier, which determines the subproblem, does include also the remaining clauses in the second alternative in (2.2) (causing a search in general which again is limited by a function in the number of clauses). In this case our subproblem, denoted by DIFF, has the following output-condition (abbreviated by $A(\mathbb{N}_1^m)$)

$$2.3. k \in \mathbb{N}_0^{n-m} \setminus k' \wedge c_P(j) = c_S(k+j) \wedge \bar{j} \in \mathbb{N}_1^m \wedge c_P(\bar{j}) \neq c_S(k' + \bar{j})$$

GUESS adds a $(\bar{j} \neq \bar{j}' \vee \bar{j} = \bar{j}') \rightarrow$ -clause where DOMAIN chooses $\bar{j}'$ from $\mathbb{N}_1^m$. The only clauses with which $\bar{j} \neq \bar{j}'$ in the first alternative can be merged are those containing $\bar{j}$ with the following obvious result.

$$2.4. A(\mathbb{N}_1^m \setminus \bar{j}') \wedge c_P(\bar{j}') = c_S(k' + \bar{j}') \vee A(\mathbb{N}_1^m) \wedge \bar{j} = \bar{j}'$$

But this time the GET-REC can be more successful if it keeps merging the added conjunct with the clauses in the original part, specifically with $c_P(j) = c_S(j+k)$; since in the first alternative for $c_P(j) \neq c_P(\bar{j}')$ we have: $c_S(j+k) = c_P(j) \neq c_P(\bar{j}') = c_S(k' + \bar{j}') = c_P(j+k' + \bar{j}' - j)$, i.e. $k \neq k' + \bar{j}' - j$ in this case for any appropriate $k', \bar{j}', j$. Similarly, in the second alternative for $c_P(j) = c_P(\bar{j}')$ and $j \neq \bar{j}'$ we have: $c_S(j+k') \neq c_P(\bar{j}') = c_P(j) = c_S(k+j) = c_S(\bar{j}' + k + j - \bar{j}')$ i.e. $k \neq k' + \bar{j}' - j$ also in this case for any appropriate k', j', j . We obtain the following recur-

sive solution for DIFF in which $PK, P\bar{j}$ denote the sets of possible k - and j -values.

$$\begin{aligned} 2.5. D_1 &:= \{k' + \bar{j}' - j \in PK \mid k' \in PK \wedge \bar{j}' \in P\bar{j} \wedge j \in \mathbb{N}_1^m \wedge c_p(j) * c_p(\bar{j}')\} \wedge \\ D_2 &:= \{k' + \bar{j}' - j \in PK \mid k' \in PK \wedge \bar{j}' \in P\bar{j} \wedge j \in \mathbb{N}_1^m \wedge c_p(j) = c_p(\bar{j}') \wedge j * \bar{j}'\} \wedge \\ \bar{j}' \in P\bar{j} \rightarrow \text{DIFF}(P\bar{j}, PK) &= \text{DIFF}(P\bar{j} \setminus \bar{j}', PK \setminus D_1) \wedge c_p(\bar{j}') = c_s(k' + \bar{j}') \vee \\ \text{DIFF}(P\bar{j}, PK) &= PK \setminus D_2 \end{aligned}$$

which has an obvious algorithmic interpretation.

With that solution (2.2) now reads

$$2.6. k' \in PK \rightarrow (IC \rightarrow OCAk = k') \vee (IC \rightarrow OCAk \in \text{DIFF}(\mathbb{N}_1^m, PK) \setminus k')$$

Comparison with (2.1) gives

$$\begin{aligned} 2.7. k' \in PK \rightarrow PM(c_s, c_p, PK) &= k' \vee \\ PM(c_s, c_p, PK) &= PM(c_s, c_p, \text{DIFF}(\mathbb{N}_1^m, PK) \setminus k') \end{aligned}$$

In algorithmic (loop) form this reads:

```
2.8. proc pm( $c_s, c_p$ ) =  $k; n \leftarrow |\text{dom } c_s|; m \leftarrow |\text{dom } c_p|; PK \leftarrow \mathbb{N}_0^{n-m};$ 
  loop  $|PK|$ ; choose  $k'$  such that  $k' \in PK; PK \leftarrow \text{diff}(\mathbb{N}_1^m, PK);$ 
  if  $PK = \text{false}$  then return  $k'; PK \leftarrow PK \setminus k';$ 
  if  $PK = \emptyset$  then return false;
end;
```

FURTHER STRATEGIES AND ALGORITHMS

For more general cases the following additional strategies are necessary: In order to structure complex problems output-conditions have to be separated into (partly) independent parts (GET-SOC); if possible, output-variables have to be expressed by others in the presence of more than one $\exists$ -quantifier (GET-RNV), otherwise, a hierarchical order between the different $\exists$'s has to be determined (CHVAR).

These and the basic strategies applied mechanically (hand-simulation) in the standard sequence yielded a detailed synthesis - as that for the maximum and the pattern-matching algorithm - of the following algorithms: maximum algorithms in different semantic context; binary search in lists; searching for duplicates in lists; testing equality of two lists; partitioning a set w.r.t. a given element; finding the i -th smallest element in a set; determining circuits in a graph; determining a minimum-cost spanning tree in a graph. Others have been synthesized in detail such as determining the moves of a roboter, etc. This remarkable list suggests that there is something fundamental in our strategies. For further details and all references see

[7] Bibel, W., Syntax-directed, Semantics-supported Program Synthesis, Proceedings 4th Workshop on Automated Decution (W.H.Joyner, jr., ed., IBM, Yorktown Heights, N.Y.), Austin, Tex., 140-147 (1979).

Improved version available as Report no. 78, University of Karlsruhe (submitted to Artificial Intelligence).

This work has been partially supported by the Deutsche Forschungsgemeinschaft (DFG).

Typescript: G. Weiher