Derivation of a String-Matching Algorithm using a Nondeterministic, State-Oriented Programming Language

David A. Rosenblueth Carlos Velarde V. Enrique Silva

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
Universidad Nacional Autónoma de México
Apdo. 20-726, 01000 México D.F.
{drosenbl, velarde}@servidor.unam.mx

Keywords: State-oriented programs, formal languages, nondeterminism, program correctness, derivation of algorithms.

1 Introduction

Partial evaluation of nondeterministic, state-oriented programs. Flowcharts look like finite automata. This similarity suggests a common generalization resulting in a nondeterministic, state-oriented (imperative) programming language. Such a formalism determines a regular algebra, in which each symbol is interpreted as a binary relation over a given set of "states," and string concatenation is interpreted as relational composition. It is widely known [2, 3, 6, 13, 15] that under this interpretation, such a regular-relational algebra obeys the axioms of the algebra of regular languages. Such axioms can then be viewed as a set of program-transformation rules with a formal-language basis. To this set we add an "unfold" operation, similar to Jones' "driving" [11]. Through the string-matching problem we argue (1) that nondeterministic, state-oriented programs are important for partial evaluation, and (2) that it is possible to carry over formal-language results into partial evaluation.

Address correspondence to David A. Rosenblueth.

Previous work. Variants of the Knuth–Morris–Pratt algorithm [14] for string matching have been formally derived in different ways, using *deterministic*, state-oriented programming [11], functional programming [1, 5, 9, 12], and logic programming [8, 18]. By basing our derivation on *nondeterministic*, state-oriented programming, we try to retake the work of Kaplan [13], Mazurkiewicz [15], Blikle [2, 3], and Engelfriet [6], connecting state-oriented programs with formal languages. Our bridge to formal languages, however, is done in the style of Prolog's "difference lists" [16].

Overview of the paper. We start by defining the semantics of our state-oriented programming language, where a program is called an "active graph." Next, we relate such a semantics with formal languages. Subsequently, we incorporate a projection operation, playing the role of the "unfold" operation of other formalisms. We finish by arriving at a variant of the Knuth–Morris–Pratt string-matching algorithm from a naive, nondeterministic description of the problem.

We think that starting from such a naive specification is preferable to starting from a more contrived program, as done in other derivations. Of the approaches mentioned above, only [18], based on logic programming, uses a naive, nondeterministic specification. The main idea of [18] is an extension of the *fold* operation, folding several clauses back into one clause. This operation amounts to a breadth-first traversal of the derivation tree. We achieve essentially the same effect by using the subset construction of a deterministic, finite automaton.

2 Active graphs

2.1 Relations defined by active graphs

Active graphs. Informally, an active graph is a directed graph with its arcs labeled by binary relations over a given set of "states," and having two distinguished nodes sometimes called "input" and "output" nodes, respectively. Essentially the same concept as that of our active graphs has appeared in various places: in [3] as "iterative algorithms," in [4] as "flowgraphs," in [6] as "nondeterministic Ianov schemes," and in [19] as "transition diagrams."

We define an *active graph* as composed of:

1. a finite automaton $\langle \Sigma, \mathcal{N}, \mathbf{s}, \mathbf{h}, \mathcal{A} \rangle$, where Σ is an alphabet, $\mathcal{N}$ is a finite set of *nodes* in which we distinguish a *start node* $\mathbf{s}$ and a *halt node* $\mathbf{h}$, and $\mathcal{A} \subseteq \mathcal{N} \times \Sigma \times \mathcal{N}$ is a set of *arcs*; and

2. an interpretation of the elements of Σ as binary relations on a set D, i.e. a function $\mathcal{I} : \Sigma \to 2^{D \times D}$.

We denote an active graph G as a 7-tuple $\langle D, \Sigma, \mathcal{N}, \mathcal{I}, \mathcal{A}, \mathbf{s}, \mathbf{h}\rangle$. D will be called the *domain* of G. The elements of D will be called *states.* The elements of Σ will be called *command symbols*, and the binary relations associated with them by $\mathcal{I}$ will be called *commands.* Relying on the context, we will sometimes write simply p instead of $\mathcal{I}(p)$.

Computations in active graphs. An active graph can be viewed as the definition of a binary relation in terms of the commands of the graph, combining these relations by means of the composition and union operations, in a way determined by the structure of the graph.

Let us consider an active graph $G = \langle D, \Sigma, \mathcal{N}, \mathcal{I}, \mathcal{A}, \mathbf{s}, \mathbf{h}\rangle$. A *computation* is a nonempty and possibly infinite sequence of (node, state)-pairs

$$\ldots, (n_i, \sigma_i), \ldots \qquad i = \ldots, -2, -1, 0, 1, 2, \ldots$$

such that σ_i and σ_{i+1} are related by a command c_i labeling an arc from n_i to n_{i+1} in G, for all i in the sequence. When the sequence is finite, $(n_j, \sigma_j), \ldots, (n_k, \sigma_k)$, we say that it is a *computation from* n_j *to* n_k. The relation whose elements are the pairs (σ_j, σ_k) for which there exists such a computation is called the *computed relation from* n_j *to* n_k, and when these nodes are respectively the start and halt nodes, we will call it the *relation computed* by the active graph and denote it by $\mathcal{R}(G)$.

Example. Consider the string-matching problem (determining whether or not a particular substring, called the pattern, occurs in another string, called the text) over the alphabet $\Gamma = \{a, b, c\}$. Figure 1 depicts an active graph. In this case, $G = \langle D, \Sigma, \mathcal{N}, \mathcal{I}, \mathcal{A}, \mathbf{s}, \mathbf{h}\rangle$, where:

$D = \Gamma^* \times \Gamma^*$; the first coordinate is associated with the pattern and the second coordinate with the text;

$\Sigma = \{t, pt, \mathit{fin}\}$,

$\mathcal{N} = \{\mathbf{s}, \mathbf{m}, \mathbf{h}\}$,

$\mathcal{A} = \{(\mathbf{s}, t, \mathbf{s}), (\mathbf{s}, pt, \mathbf{m}), (\mathbf{m}, pt, \mathbf{m}), (\mathbf{m}, \mathit{fin}, \mathbf{h})\}$, and

$\mathcal{I}$ assigns relations to command symbols as follows:

$$\begin{array}{rcl} t & \mapsto & \{(\langle\alpha, x\beta\rangle, \langle\alpha, \beta\rangle) : x \in \Gamma \ \&\ \alpha, \beta \in \Gamma^*\} \\ pt & \mapsto & \{(\langle x\alpha, x\beta\rangle, \langle\alpha, \beta\rangle) : x \in \Gamma \ \&\ \alpha, \beta \in \Gamma^*\} \\ \mathit{fin} & \mapsto & \{(\langle\Lambda, \beta\rangle, \langle\Lambda, \beta\rangle) : \beta \in \Gamma^*\} \end{array}$$

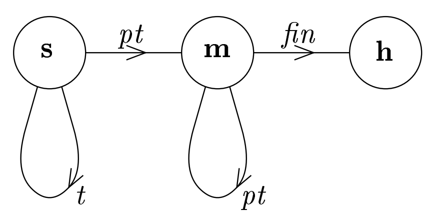

Figure 1: *An active graph for the string-matching problem.*

Hence, execution of the command (with command symbol) t advances one symbol on the text. Execution of the command pt advances one common symbol on both the pattern and the text. Execution of the command *fin* does not change the state, but can only be performed if the current pattern is empty. Note that the active graph in Figure 1 is nondeterministic: from node $\mathbf{s}$, if the leftmost symbol of the current pattern and text are the same, we can either (1) execute the command t and remain in node $\mathbf{s}$, or (2) execute the command pt and go to node $\mathbf{m}$.

2.2 Finite automata viewed as active graphs

The representation of lists by "difference lists" [16] suggests the possibility of representing a formal language by a binary relation. We define the *quotient representation* of a language L over an alphabet Σ as:

$$\mathcal{Q}_L = \{(\langle\alpha\beta\rangle, \langle\beta\rangle) \;:\; \alpha \in L \text{ and } \beta \in \Sigma^*\}$$

It is easy to prove that the quotient representation of the product of languages[1] is equal to the composition[2] of the quotient representation of the languages: $\mathcal{Q}_{L\cdot M} = \mathcal{Q}_L; \mathcal{Q}_M$. Similarly, the quotient representation of the union of languages is equal to the union of their quotient representations: $\mathcal{Q}_{L\cup M} = \mathcal{Q}_L \cup \mathcal{Q}_M$. Observe now that every finite automaton $A = \langle \Sigma, \mathcal{N}, \mathbf{s}, \mathbf{h}, \mathcal{A}\rangle$ with a single final node can be used to construct an active graph G_A by taking Σ^* as D and $\mathcal{Q}_{\{a\}}$ as $\mathcal{I}(a)$ for each $a \in \Sigma$. We then have that $\mathcal{R}(G_A) = \mathcal{Q}_{L(A)}$, where $L(A)$ is the language accepted by A. We will say that G_A is *the $\mathcal{Q}$ active graph of A*.

Example (continued). Our objective is to derive a variant of the Knuth–Morris–Pratt algorithm from the active graph in Figure 1 using a formal-language result. Note, however, that the states of such an active graph have two components, whereas the states of active graphs representing finite automata have only one component. We will therefore introduce an operation which removes one component of each of the two states related by a command of our string-matching active graph. In addition, we will see that *the resulting graph is the $\mathcal{Q}$ active graph of a finite automaton.*

2.3 Projection

Projection. Our "projection" function is similar to Jones' "driving" [11]. A difference is that we deal with nondeterministic computations. When an active graph is projected with respect to a given value of some coordinate of the initial state, an active graph results, which is specialized to computations starting with such a value. For example, the string-matching active graph in Figure 1 has states with one coordinate associated with the pattern and another coordinate associated with the text. The projection of such a graph with respect to a given initial pattern results in an active graph specialized to searching for such a pattern.

Intuitively, the projection operation on an active graph discards the coordinate of the states corresponding to the initial value given (i.e. projects, in the usual sense of this word, the commands on the rest of the coordinates). This elimination is accompanied by the addition of nodes which record the values of the eliminated coordinate occurring in computations of the original active graph. The new nodes thus code the information

[1] The product of languages L and M is defined as: $L \cdot M = \{\alpha\beta : \alpha \in L \;\&\; \beta \in M\}$.

[2] The composition of relations R and S is defined as: $R; S = \{(x, z) : \exists y.(x, y) \in R \;\&\; (y, z) \in S\}$.

lost by the projection of the commands. Hence, the projection function constructs a new active graph in such a way that the values of a coordinate of the states are transformed into nodes (i.e. into “states” in the sense of finite automata).

Before defining the projection function, we need some notational conventions. Let $R \subseteq (D_1 \times D_2 \times \cdots \times D_n)^2$. We define:

$$\begin{aligned}
R|_x &= \{(\langle u_1, \ldots, u_n\rangle, \langle v_1, \ldots, v_n\rangle) \in R : u_1 = x\} \\
\pi_{D_1}(R) &= \{(\langle u_1\rangle, \langle v_1\rangle) : (\langle u_1, u_2, \ldots, u_n\rangle, \langle v_1, v_2 \ldots, v_n\rangle) \in R\} \\
\pi_{D_2\times\cdots\times D_n}(R) &= \{(\langle u_2, \ldots, u_n\rangle, \langle v_2, \ldots, v_n\rangle) : (\langle u_1, u_2, \ldots, u_n\rangle, \langle v_1, v_2 \ldots, v_n\rangle) \in R\} \\
\mathbf{1}_{D_2\times\cdots\times D_n} &= \{(\langle u_2, \ldots, u_n\rangle, \langle u_2, \ldots, u_n\rangle) : u_i \in D_i,\ i = 2, \ldots, n\}
\end{aligned}$$

That is: $R|_x$ is the subset of R whose elements have x as first coordinate of the left tuple; $\pi_{D_1}(R)$ is the relation formed from the values of the first coordinates of each tuple of each element of R, $\pi_{D_2\times\cdots\times D_n}(R)$ is the relation obtained by projecting (in the usual sense of this word) each tuple of each element of R on $D_2 \times \cdots \times D_n$; and $\mathbf{1}_{D_2\times\cdots\times D_n}$ is the identity relation on $D_2 \times \cdots \times D_n$.

For simplicity, we will only define active-graph projection with respect to a value of the first coordinate. Let $G = \langle D, \Sigma, \mathcal{N}, \mathcal{I}, \mathcal{A}, \mathbf{s}, \mathbf{h}\rangle$ be an active graph, such that $D = D_1 \times D_2 \times \cdots \times D_n$. We define a function Π from active graphs and D_1 to active graphs as follows. $\Pi(G, x_0) = \langle D', \Sigma', \mathcal{N}', \mathcal{I}', \mathcal{A}', \mathbf{s}', \mathbf{h}'\rangle$, such that:

1. $D' = D_2 \times \cdots \times D_n$,

2. $\Sigma' \subseteq \{[p]_x : p \in \Sigma \ \&\ x \in D_1\}$, where $[p]_x$ is the equivalence class of p with respect to the following equivalence relation:

   $$\{(p, q) : p, q \in \Sigma \ \&\ \pi_{D'}(\mathcal{I}(p)|_x) = \pi_{D'}(\mathcal{I}(q)|_x)\}$$

3. $\mathcal{N}' \subseteq (\mathcal{N} \times D_1) \cup \{\mathbf{h}'\}$,

4. $\mathcal{I}'([p]_x) = \pi_{D'}(\mathcal{I}(p)|_x)$,

5. $\mathcal{A}'$ is inductively defined as the smallest set such that:

   $$(\langle \mathbf{s}, x_0\rangle, [p]_{x_0}, \langle m, y\rangle) \in \mathcal{A}' \quad \text{if} \quad (\mathbf{s}, p, m) \in \mathcal{A} \ \&\ (x_0, y) \in \pi_{D_1}(\mathcal{I}(p)) \tag{1}$$

   $$(\langle m, y\rangle, [q]_y, \langle n, z\rangle) \in \mathcal{A}' \quad \text{if} \quad (m, q, n) \in \mathcal{A} \ \&\ (y, z) \in \pi_{D_1}(\mathcal{I}(q)) \ \&\ \exists l, x, p.\ (\langle l, x\rangle, [p]_x, \langle m, y\rangle) \in \mathcal{A}' \tag{2}$$

   $$(\langle \mathbf{h}, z\rangle, \mathbf{1}_{D'}, \mathbf{h}') \in \mathcal{A}' \quad \text{if} \quad \exists m, y, q.\ (\langle m, y\rangle, [q]_y, \langle \mathbf{h}, z\rangle) \in \mathcal{A}' \tag{3}$$

   Note that (1)–(3) also define Σ' and $\mathcal{N}'$.

6. $\mathbf{s}' = \langle \mathbf{s}, x_0\rangle$. $\square$

In the definition of $\mathcal{A}'$, (1) is the base case, (2) is the inductive case, and (3) is included to produce an active graph with only one halt node, since nondeterminism of (1) and (2) may generate several nodes in which computations must end.

Example (continued). Let us compute $\Pi(G, ababc)$ for the example. Before that, however, it is convenient to observe that $t = t_a \cup t_b \cup t_c$, where:

$$\begin{array}{rcl} t_a & = & \{(\langle \alpha, a\beta\rangle, \langle \alpha, \beta\rangle) : \alpha, \beta \in \Gamma^*\} \\ t_b & = & \{(\langle \alpha, b\beta\rangle, \langle \alpha, \beta\rangle) : \alpha, \beta \in \Gamma^*\} \\ t_c & = & \{(\langle \alpha, c\beta\rangle, \langle \alpha, \beta\rangle) : \alpha, \beta \in \Gamma^*\} \end{array}$$

So that in G we can replace the arc $(\mathbf{s}, t, \mathbf{s})$ by: $(\mathbf{s}, t_a, \mathbf{s})$, $(\mathbf{s}, t_b, \mathbf{s})$, and $(\mathbf{s}, t_c, \mathbf{s})$. This operation is closely related to Jones' "bounded static variation" [12].

Consider $\mathcal{I}'([t_a]_{ababc})$. We first take the elements of $\mathcal{I}(t_a)$ having $ababc$ as first coordinate of the left tuple:

$$\mathcal{I}(t_a)|_{ababc} = \{(\langle ababc, a\beta\rangle, \langle ababc, \beta\rangle) : \beta \in \Gamma^*\}$$

and then we project each tuple of each element of this relation on D_2:

$$\mathcal{I}'(\pi_{D_2}(\mathcal{I}(t_a)|_{ababc})) = \{(\langle a\beta\rangle, \langle \beta\rangle) : \beta \in \Gamma^*\}$$

The crucial point in our derivation is the fact that the same command is obtained through pt. That is:

$$\mathcal{I}(pt)|_{ababc} = \{(\langle ababc, a\beta\rangle, \langle babc, \beta\rangle) : \beta \in \Gamma^*\}$$

so that $\mathcal{I}'(\pi_{D_2}(\mathcal{I}(t_a)|_{ababc})) = \mathcal{I}'(\pi_{D_2}(\mathcal{I}(pt)|_{ababc}))$. This will allow us to use a formal-language result, since the resulting active graph is the $\mathcal{Q}$ active graph of a nondeterministic finite automaton.

Applying (1)–(3), we obtain:

$$\begin{array}{rl} \mathcal{A}' = \{ & (\langle \mathbf{s}, ababc\rangle, [t_a]_{ababc}, \langle \mathbf{s}, ababc\rangle) \\ , & (\langle \mathbf{s}, ababc\rangle, [t_b]_{ababc}, \langle \mathbf{s}, ababc\rangle) \\ , & (\langle \mathbf{s}, ababc\rangle, [t_c]_{ababc}, \langle \mathbf{s}, ababc\rangle) \\ , & (\langle \mathbf{s}, ababc\rangle, [pt]_{ababc}, \langle \mathbf{m}, babc\rangle) \\ , & (\langle \mathbf{m}, babc\rangle, [pt]_{babc}, \langle \mathbf{m}, abc\rangle) \\ , & (\langle \mathbf{m}, abc\rangle, [pt]_{abc}, \langle \mathbf{m}, bc\rangle) \\ , & (\langle \mathbf{m}, bc\rangle, [pt]_{bc}, \langle \mathbf{m}, c\rangle) \\ , & (\langle \mathbf{m}, c\rangle, [pt]_{c}, \langle \mathbf{m}, \Lambda\rangle) \\ , & (\langle \mathbf{m}, \Lambda\rangle, [\mathit{fin}]_{\Lambda}, \langle \mathbf{h}, \Lambda\rangle) \\ , & (\langle \mathbf{h}, \Lambda\rangle, \mathbf{1}_{D_2}, \mathbf{h}') \\ \} & \end{array}$$

where:

$$[t_a]_{ababc} = [pt]_{ababc} = [pt]_{abc} = \{(\langle a\beta\rangle, \langle \beta\rangle) : \beta \in \Gamma^*\}$$

$$
\begin{aligned}
[t_b]_{ababc} = [pt]_{babc} = [pt]_{bc} &= \{(\langle b\beta\rangle, \langle\beta\rangle) : \beta \in \Gamma^*\} \\
[t_c]_{ababc} = [pt]_c &= \{(\langle c\beta\rangle, \langle\beta\rangle) : \beta \in \Gamma^*\} \\
[\mathit{fin}]_\Lambda &= \{(\langle\beta\rangle, \langle\beta\rangle) : \beta \in \Gamma^*\}
\end{aligned}
$$

Figure 2 shows the resulting active graph, where $\overline{a} = [t_a]_{ababc}$, $\overline{b} = [t_b]_{ababc}$, $\overline{c} = [t_c]_{ababc}$, and $\mathbf{1} = [\mathit{fin}]_\Lambda$.

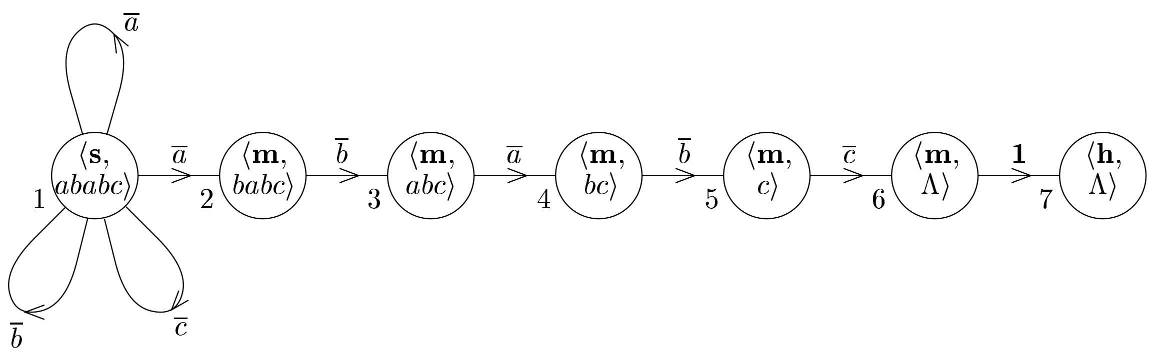

Figure 2: *An active graph for matching the string ababc.*

In general, our projection function must face the problem of partitioning the variables into "static" and "dynamic." This problem has been studied in [12], to where we refer the reader.

Theorem Let G be an active graph with domain $D_1 \times \cdots \times D_n$, let $x_0 \in D_1$, and let $D' = D_2 \times \cdots \times D_n$. Then $\pi_{D'}(\mathcal{R}(G)|_{x_0}) = \mathcal{R}(\Pi(G, x_0))$.

Proof sketch The sequence

$$(\mathbf{s}, x_0),\ (n_1, x_1), \ldots$$

is a computation of G iff the sequence

$$(\langle\mathbf{s}, x_0\rangle, \pi_{D'}(x_0)),\ (\langle n_1, x_1\rangle, \pi_{D'}(x_1)), \ldots$$

is a computation of $\Pi(G, x_0)$, where $\pi_{D'}(x_i)$ is the tuple projection of x_i on D'. $\square$

2.4 Application of a formal-language result

It is well known that the Knuth–Morris–Pratt algorithm is intimately related to finite automata recognizing languages of the form $\Sigma^* p$ [10, 17, 20]. Note that the diagram in Figure 2 can be regarded either (1) as the $\mathcal{Q}$ active graph of a finite automaton

A or (2) as the transition diagram of A. The automaton A accepts, by the theorem, precisely the language $\Sigma^* ababc$. From an implementation point of view, however, A has the disadvantage of being nondeterministic. We can circumvent this difficulty by obtaining an equivalent deterministic finite automaton by using, for instance, the *subset construction* method. Figure 3 exhibits the resulting automaton.

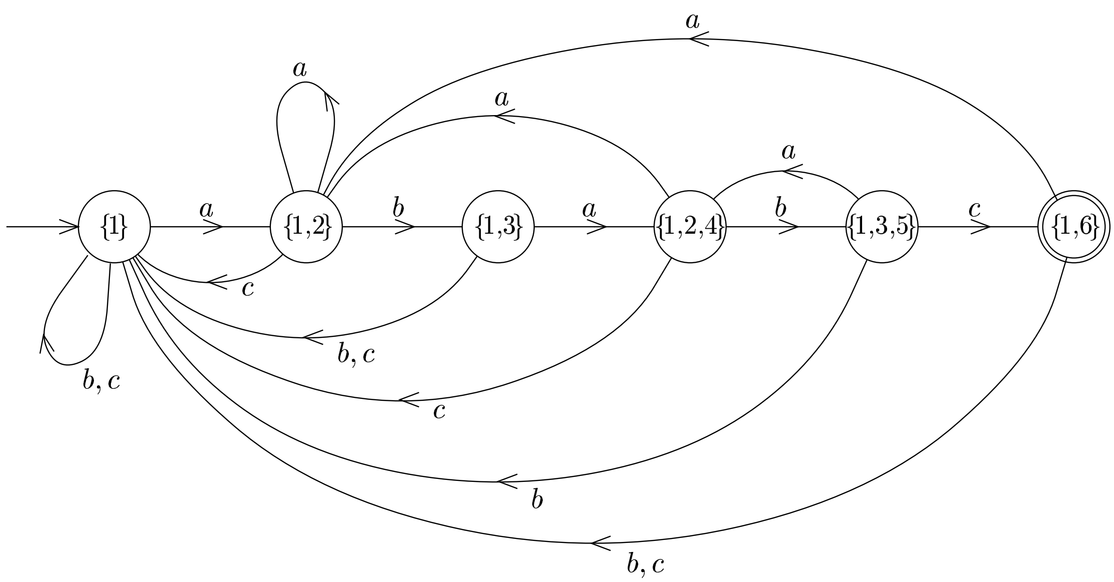

Figure 3: *A deterministic finite automaton accepting* $\Sigma^* ababc$.

Finally, to recover an active graph, we can give to this automaton the same interpretation $\mathcal{I}$ used for the previous $\mathcal{Q}$ active graph, thus arriving at a $\mathcal{Q}$ active graph which is closely related to the Knuth–Morris–Pratt algorithm.

It is worth noting that the Knuth–Morris–Pratt algorithm has two stages, both of which are efficient and independent of the alphabet size. The first stage constructs a representation of a finite automaton in time $O(m)$, where m is the length of the pattern. The second stage simulates this automaton to search for the pattern in time $O(n)$, where n is the length of the text. Here, we have only achieved the desired efficiency in the second stage. The first stage of this algorithm corresponds to (1) projecting the active graph in Figure 1 and then (2) computing the subset construction of the finite automaton in Figure 2. Apparently, both the projection and the subset construction can be performed, using hashing, for instance, in time proportional to the number of edges of the finite automata, which depends not only on m, but on the alphabet size as well.

3 Concluding remarks

Program transformation studies the derivation of programs through correctness-preserving steps. Techniques developed by this school of thought have been applied to deterministic, state-oriented (imperative) programming, functional programming, and logic programming. Nondeterministic, state-oriented programming, however, has long been recognized [7] as a correctness-oriented programming paradigm. Hence, it is puzzling that program transformation has not been extensively applied to such a paradigm.

Here, we have argued that nondeterministic programs can be transformed in a natural way. Moreover, nondeterministic, state-oriented program-transformation has the additional advantage of having a formal-language basis.

Acknowledgment

We are grateful to IIMAS, UNAM for support.

References

[1] R.S. Bird, J. Gibbons, and G. Jones. Formal derivation of a pattern matching algorithm. *Science of Computer Programming*, 12:93–104, 1989.

[2] A. Blikle. Nets: complete lattices with a composition. *Bull. Acad. Polon. Sci., Sér. Sci., Math., Astr. Phys.*, 19:1123–1127, 1971.

[3] Andrzej Blikle. A comparative review of some program verification methods. In J. Gruska, editor, *Mathematical Foundations of Computer Science 1977*, pages 17–33. Springer-Verlag, 1977. Lecture Notes in Computer Science No. 53.

[4] Keith L. Clark and M.H. van Emden. Consequence verification of flowcharts. *IEEE Transactions on Software Engineering*, SE-7(1):52–60, January 1981.

[5] Charles Consel and Olivier Danvy. Partial evaluation of pattern matching in strings. *Information Processing Letters*, 30:79–86, 1989.

[6] Joost Engelfriet. *Simple Program Schemes and Formal Languages.* Springer-Verlag, 1974. Lecture Notes in Computer Science No. 20.

[7] Robert W. Floyd. Nondeterministic algorithms. *Journal of the ACM*, 14(4):636–644, 1967.

[8] J.P. Gallagher. A system for specialising logic programs. Technical report, University of Bristol, U.K., November 1991.

[9] Robert Glück and Andrei V. Klimov. Occam's razor in metacomputation: the notion of a perfect process tree. In P. Cousot, M. Falaschi, G. Filè, and A. Ranzy, editors, *Static Analysis Proceedings*, pages 112–123. Springer-Verlag, 1993. Lecture Notes in Mathematics 724.

[10] Christophe Hancart. On Simon's string searching algorithm. *Information Processing Letters*, 47(2):95–99, 1993.

[11] Neil D. Jones. The essence of program transformation by partial evaluation and driving. In Neil D. Jones, Masami Hagiya, and Masahiko Sato, editors, *Logic, Language and Computation*, pages 206–224. Springer-Verlag, 1994. Lecture Notes in Computer Science No. 792.

[12] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice-Hall, 1993.

[13] Donald M. Kaplan. Regular expressions and the equivalence of programs. *Journal of Computer and System Sciences*, 3(4):361–386, 1969.

[14] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[15] A. Mazurkiewicz. Recursive algorithms and formal languages. *Bull. Acad. Polon. Sci., Sér. Sci., Math., Astr. Phys.*, 20:799–803, 1972.

[16] Fernando C.N. Pereira and David H.D. Warren. Definite clause grammars for language analysis —A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[17] Dominique Perrin. Finite automata. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 3–57. Elsevier Science Publishers B.V., 1990.

[18] Alberto Pettorossi, Maurizio Proietti, and Sophie Renault. Enhancing partial deduction via unfold/fold rules. In *6th Int. Workshop on Logic Program Synthesis and Transformation*, 1996.

[19] John C. Reynolds. *The Craft of Programming.* Prentice Hall, 1981.

[20] Imre Simon. String matching algorithms and automata. In J. Darhumäki, H. Maurer, and G. Rozenberg, editors, *Results and Trends in Theoretical Computer Science. Colloquium in Honor of Arto Salomaa*, pages 386–395, Graz, Austria, June 1994. Springer-Verlag. Lecture Notes in Computer Science Vol. 812.