EFFICIENT RECOGNITION OF RATIONAL RELATIONS

Jan van Leeuwen and Maurice Nivat

RUU-CS-80-12

December 1980

Rijksuniversiteit Utrecht

Vakgroep informatica

Princetonplein 5
Postbus 80.002
3508 TA Utrecht
Telefoon 030–53 1454
The Netherlands

EFFICIENT RECOGNITION OF RATIONAL RELATIONS

Jan van Leeuwen and Maurice Nivat

Technical Report RUU-CS-80-12
December 1980

Department of Computer Science
University of Utrecht
P.O. Box 80.002
3508 TA Utrecht, the Netherlands

EFFICIENT RECOGNITION OF RATIONAL RELATIONS

Jan van Leeuwen* and Maurice Nivat**

Abstract. Let R be a k-ary rational relation over a fixed size alphabet. We show that elements of R can be recognized in $O(n^k/\log^{\frac{1}{k-1}} n)$ steps on a random access machine.

1. Introduction.

Let $R \subseteq \Sigma_1^* \times \ldots \times \Sigma_k^*$ $(k \geq 2)$ be a rational relation. The easiest way to comprehend R is to view it as the set of accepted tuples of inputs of a k-tape nondeterministic finite-state acceptor A with ε-moves. In this paper we consider the complexity of deciding whether tuples $\langle u_1, \ldots, u_k \rangle \in \Sigma_1^* \times \ldots \times \Sigma_k^*$ belong to R, using a deterministic computing device.

While known to automata theorists for years, rational relations have recently become of interest in the study of parallel processes (cf. Nivat [2]). Consider k processes A_1 to A_k and view each process as a nondeterministic finite state machine. The possible "behaviours" of process A_i (strings of actions from a finite repertoire Σ_i) will form a rational set R_i which describes how A_i can function. When processes A_1 to A_k cooperate in one system, the necessary synchronization among them is likely to eliminate many tuples $\langle u_1, \ldots, u_k \rangle \in R_1 \times \ldots \times R_k$ as admissible behaviours. To describe the set R of possible behaviours, given a particular synchronization mechanism, one can usually distinguish a rational set $K \subseteq \Delta^*$ and homomorphisms $\varphi_i : \Delta^* \to \Sigma_i^*$ $(1 \leq i \leq k)$ such that

$$R = R_1 \times \ldots \times R_k \cap \{\langle \varphi_1(v), \ldots, \varphi_k(v) \rangle \mid v \in K\}$$

This makes R into a rational relation and it is not hard to see that every rational relation can be so characterized.

Rather than recognizing R directly, we will introduce in section 2 a very special type of relations (called discrete finite-state transductions) from which all rational relations can easily be obtained. In section 3 we

---

* Department of Computer Science, University of Utrecht, P.O. Box 80.002, 3508 TA Utrecht, the Netherlands.

**Laboratoire Informatique Theoretique et Programmation, Université Paris VII, 2 Place Jussieu, 75251 Paris Cedex 05, France.

present a fairly standard tabular technique to recognize discrete k-ary finite-state transductions in $O(n^k)$ steps on a random access machine with unit cost per instruction (cf. Aho, Hopcroft and Ullman [1]). In section 4 we improve this to an $O(n^k/\log^{\frac{1}{k-1}} n)$ algorithm. A simple application is that for strings u_1, u_2 and w over a fixed size alphabet with $|u_1|, |u_2| \leq |w| = n$, one can determine in $O(\frac{n^2}{\log n})$ steps whether v is a shuffle of u_1 and u_2.

2. Discrete finite-state transductions.

In addition to its usual one-way input tapes (tapes 1 to k), a finite-state transducer has a one-way write-only output tape (tape 0) as well. Finite control contains instructions of the form

$$[p, a_1, \ldots, a_k] \Rightarrow [q, w] \qquad (a_i \in \Sigma_i \cup \{ \varepsilon \}, w \in \Delta^*)$$

which indicate that the transducer, when in state p, can read symbols a_i (or ε) from tape i (while moving the input heads accordingly), print w on tape 0 and go to state q. Throughout this paper we only consider nondeterministic transducers.

Definition. $R \subseteq \Sigma_1^* \times \ldots \times \Sigma_k^* \times \Delta^*$ is called a finite-state transduction if and only if there exists a finite-state transducer A such that $\langle u_1, \ldots, u_k, w \rangle \in R$ just when on inputs u_1 to u_k on tapes 1 to k, A can print w on tape 0 and finish in an accepting state, with the input heads running off the right end of the input tapes.

Finite-state transductions obviously are rational relations. A "discrete" transducer will move just one input head over a square per instruction and prints a single symbol rather than a "word" of output every time.

Defintion. A finite-state transducer A is called discrete if and only if in all of its instructions $[p, a_1, \ldots, a_k] \Rightarrow [q, w]$ (i) precisely one of the a_i is non-ε and (ii) $|w| = 1$.

The transductions defined by discrete finite-state transducers will be called discrete finite-state transductions. For example, $S = \{\langle u_1, u_2, w \rangle \mid w \text{ is a shuffle of } u_1 \text{ and } u_2\} \subseteq \Sigma^* \times \Sigma^* \times \Sigma^*$ is a discrete finite-state transduction of arity 2.

Definition. The companion of a k-ary string-relation $R \subseteq \Sigma_1^* \times \ldots \times \Sigma_k^*$ is the relation $\hat{R} = \{\langle u_1, \ldots, u_k, a^{n_1+\ldots+n_k} \rangle \mid \langle u_1, \ldots, u_k \rangle \in R \text{ and } |u_i| = n_i \text{ for } 1 \leq i \leq k\} \subseteq \Sigma_1^* \times \ldots \times \Sigma_k^* \times a^*$.

Proposition 2.1. R is a rational relation if and only if its companion $\hat{R}$ is a discrete finite-state transduction.

Proof

The "if" part should be obvious. To prove the "only if" part, let R be a rational relation and let $\mathcal{A}$ be a k-tape nondeterministic finite-state acceptor for R with instruction set I. For each state p define

$$\text{FOLLOW}(p) = \{p\} \cup \{q \mid [p, \varepsilon, \ldots, \varepsilon] \Rightarrow q \text{ is in } I\}$$

Regardless of what $\mathcal{A}$'s initial state is, transform I as follows:

(i) add $[p, a_1, \ldots, a_k] \Rightarrow q$ for every instruction $[r, a_1, \ldots, a_k] \Rightarrow q$ in I with at least one a_i non-ε and $r \in \text{FOLLOW}(p)$,
(ii) add every p which has an accepting state in its FOLLOW set to the set of accepting states,
(iii) drop the instructions $[p, \varepsilon, \ldots, \varepsilon] \Rightarrow \ldots$, with all inputs ε, from I.

It is obvious that $\mathcal{A}$ still accepts the same relation, but its pure ε-moves have been eliminated. If $\mathcal{A}$ isn't discrete yet, then it must contain an instruction

$$[p, \ldots, a_i, \ldots, a_j, \ldots] \Rightarrow q$$

with both an a_i and an a_j $(i \neq j)$ non-ε. It can be eliminated by introducing a "new" state qnew and replacing the instruction by the pair

$$[p, \ldots, a_i, \ldots, \varepsilon, \ldots] \Rightarrow \text{qnew}$$
$$[\text{qnew}, \ldots, \varepsilon, \ldots, a_j, \ldots] \Rightarrow q$$

Repeating this yields an instruction set for $\mathcal{A}$ that has precisely one a_i non-ε in every instruction $[p, a_1, \ldots, a_k] \Rightarrow q$ it contains. Printing an "a" with every such instruction makes $\mathcal{A}$ into a discrete finite-state transducer for $\hat{R}$. □

Proposition 2.1. shows that for recognizing rational relations we may restrict attention to the more well-behaved discrete finite-state transductions.

3. A tabular recognition method for discrete finite-state transductions.

Let $R \subseteq \Sigma_1^* \times \ldots \times \Sigma_k^* \times \Delta^*$ be a discrete finite-state transduction and $\mathcal{A}$ a fair transducer for R. To recognize whether $\langle u_1, \ldots, u_k, w\rangle$ belongs to R one may proceed as follows. Let $|u_i| = n_i$ $(1 \leq i \leq k)$ and $|w| = n = n_1 + \ldots + n_k$. We may assume without loss of generality that $n_i > 0$ for all i. (Otherwise we could do the procedure below as if $\mathcal{A}$ had fewer input tapes and obtain a better time-bound.)

Design a k-dimensional table $T = \{(i_1, \ldots, i_k) \mid 0 \leq i_j \leq n_j$ for $1 \leq j \leq k\}$ and think of the j^{th} coordinate axis as the set of possible positions of the j^{th} input head on $u_j = u_j[1] \ldots u_j[n_j]$. No special coordinate axis is required for w, because in discrete transductions the position of the write-head on the output-tape is always equal to the sum of the number of symbols read on the joint input tapes! Table entries will be referred to as $T(i_1, \ldots, i_k)$ and will contain finite sets of transducer states. More precisely we demand that

$q \in T(i_1, \ldots, i_k) \Leftrightarrow$ Á can reach position i_j on tape j for $1 \leq j \leq k$ in state q, while the output produced is $w[1] \ldots w[i_1+\ldots+i_k]$

To "fill" a table or a table-entry will mean to fill it according to this characterisation. $T(0, \ldots, 0)$ will contain just the initial state of Á and $\langle u_1, \ldots, u_k, w\rangle \in R$ <u>if and only if</u> $T(n_1, \ldots, n_k)$ <u>contains an accepting state</u>.

<u>Lemma</u> 3.1. A table-entry $T(i_1, \ldots, i_k)$ can be filled once we know the contents of $T(i_1 \dot{-} 1, i_2, \ldots, i_k)$, $T(i_1, i_2 \dot{-} 1, \ldots, i_k)$, ..., and $T(i_1, i_2, \ldots, i_k \dot{-} 1)$.

<u>Proof</u>

The result is trivial when all i_j $(1 \leq j \leq k)$ are zero. Assume $i_j \geq 1$ for at least one j. To reach positions i_1 to i_k on the input tapes, a discrete finite-state transducer must have made its last move with one of the input-heads (say, the j^{th}) going from square i_j-1 to square i_j. Hence

$q \in T(i_1, \ldots, i_k) \Leftrightarrow$ there is a j $(1 \leq j \leq k)$ with $i_j \geq 1$ and a $p \in T(i_1, \ldots, i_j-1, \ldots, i_k)$ such that $[p, \varepsilon, \ldots, a_j, \ldots, \varepsilon] \Rightarrow [q, b]$ is in I, $a_j = u_j[i_j]$ and $b = w[i_1+\ldots+i_k]$.

The result follows. □

With any ordinary address calculation scheme it takes O(k) to compute the address of $T(i_1, \ldots, i_k)$ but only O(1) more steps to get the address of each $T(i_1, \ldots, i_j-1, \ldots, i_k)$ from it. Thus $T(i_1, \ldots, i_k)$ can be filled in O(k) steps, where the constant of proportionality depends only on the size of I. We conclude:

<u>Lemma</u> 3.2. $T(i_1, \ldots, i_k)$ can be filled in O(k) steps once its low-end neighbors in table T are filled.

The lemma is instrumental in obtaining an efficient algorithm to fill T. For k = 1 an immediate "one sweep" computation will do. Proceeding inductively, we can fill $T(*, \ldots, *)$ by filling the slice $T(0, *, \ldots, *)$ first as if it were a (k-1)-dimensional table, followed by filling $T(1, *, \ldots, *)$ and so on until, in a final round, $T(n_1, *, \ldots, *)$ gets filled. The algorithm will visit all table-entries precisely once, spends only O(1) in organisational overhead per entry and whenever an entry is visited, it knows that (due to the recursive set-up) all low-end neighbors are available.

Theorem 3.3. Given a discrete finite-state transduction R, one can decide whether $\langle u_1, \ldots, u_k, w\rangle$ belongs to R in $O(k.\prod_1^k (n_i+1)) = O(n^k)$ steps on a random access machine.

Proof

Implement the above algorithm. By lemma 3.2. it takes O(k) steps to fill an entry and the algorithm visits all $\prod_1^k (n_i+1)$ entries of T precisely once. □

With 2.1. we can immediately conclude

Corollary 3.4. One can recognize elements of k-ary rational relations in only $O(n^k)$ steps on a random access machine.

Observe in the given algorithm that to fill $T(i+1, *, \ldots, *)$ one only needs to have $T(i, *, \ldots, *)$ in storage. Thus, with a proper organisation, only $O(n^{k-1})$ memory-cells need to be in use at any one time. We will see a different table-filling technique next.

4. Saving time in recognizing discrete finite-state transductions.

The remainder of this paper relies on the assumption that we can bound the size of tape-alphabets by a fixed constant σ. We have noted that the contents of an entry $T(i_1, \ldots, i_k)$ is completely determined by the contents of its low-end neighbors and by $w[i_1+\ldots+i_k]$. Since the number of configurations of this type is bounded, T will contain many identical entries just because their environment is identical. Pushing this to a logical extreme, we can divide T into blocks of some size and claim that T must have many identical blocks just because these blocks have identical environments.

Let $d_1, \ldots, d_k$ be non-zero integers with $d_j \leq n_j$ $(1 \leq j \leq k)$ and let $d = d_1 + \ldots + d_k$. We shall not fix a particular choice for the integers d_j until the proof of theorem 4.5. A block is any sub-table of T with indices

restricted to a set of the form

$$v + \{(i_1, \ldots, i_k) \mid 0 \le i_j \le d_j \text{ for } 1 \le j \le k\}$$

, where v is a nonnegative displacement vector. We shall always identify a block and its set of indices in T. Every block as given has k low-end faces L_s $(1 \le s \le k)$, which are the sub-blocks defined by

$$L_s \equiv v + \{(i_1, \ldots, i_k) \mid i_s = 0 \text{ and } 0 \le i_j \le d_j \text{ for } j \ne s\}$$

, and likewise it has k high-end faces H_s $(1 \le s \le k)$ which are the sub-blocks defined by

$$H_s \equiv v + \{(i_1, \ldots, i_k) \mid i_s = d_s \text{ and } 0 \le i_j \le d_j \text{ for } j \ne s\}$$

While a block has $\prod_1^k (d_j+1) = O(d^k)$ entries, its low-end faces (and likewise, its high-end faces) only have $\sum_1^k d_1 \ldots \not{d}_s \ldots d_k = O(d^{k-1})$ entries. Consider a block embedded in T.

Lemma 4.1. The contents of a block (and thus the contents of its high-end faces) can be computed in $O(k.\prod_1^k (d_j+1)) = O(d^k)$ steps, once the contents of its low-end faces and the appropriate portions of u_1 to u_k and w are known.

Proof.

This is 3.2. in a different guise. It is obvious for k = 1. Proceeding inductively, we can fill the block by filling the subsets $v + \{(i_1, \ldots, i_k) \mid i_1 = \lambda \text{ and } 0 \le i_j \le d_j \text{ for } 2 \le j \le k\}$ for λ from 0 to d_1, in that order, as if it were (k-1)-dimensional blocks. The algorithm visits every entry of the block once, spends O(k) per entry and (hence) finishes within $O(k.\prod_1^k (d_j+1))$ steps. □

Once we have computed a block it will take only $O(d^{k-1})$ steps to shift the contents of its high-end faces by a certain displacement to any other place where a block occurs with the same environment and there is no need to really compute the entire block over again. We shall see in a moment that this is sufficient for our purposes.

It is tempting to divide T into blocks $B(t_1, \ldots, t_k)$ defined by

$$B(t_1, \ldots, t_k) \equiv \{(i_1, \ldots, i_k) \mid t_j d_j \le i_j \le (t_j+1)d_j \text{ for } 1 \le j \le k\}$$

, for $0 \le t_j \le \left\lceil \frac{n_j}{d_j} \right\rceil - 1$ $(1 \le j \le k)$. The blocks cover T exactly when d_j divides n_j for all j, but include empty entries beyond the border of T whenever there is a d_j not dividing n_j (some j). To avoid this impediment to uniformity, we shall assume that all inputs are extended with a few $-signs on which the transducer copies and maintains state, just to make n_j a proper multiple of

d_j for every j. We proceed on the assumption that the $B(t_1, \ldots, t_k)$ cover T exactly.

Note that the s^{th} low-end face of a $B(t_1, \ldots, t_k)$ with $t_s = 0$ is "available" once we have all entries $T(*, \ldots, 0, \ldots, *)$ in store, with a "0" in at least one coordinate position. For the remaining faces and blocks, the following observation is crucial.

Lemma 4.2. For any s ($1 \le s \le k$) with $t_s \ge 1$, the s^{th} low-end face of $B(t_1, \ldots, t_k)$ precisely coincides with the s^{th} high-end face of $B(t_1, \ldots, t_s-1, \ldots, t_k)$.

Proof

The high-end face H_s of $B(t_1, \ldots, t_s-1, \ldots, t_k)$ covers the indices in the set

$$(t_1d_1, \ldots, (t_s-1)d_s, \ldots, t_kd_k) + \{(i_1, \ldots, i_k) \mid i_s = d_s \text{ and } 0 \le i_j \le d_j \text{ for } j \ne s\} \equiv (t_1d_1, \ldots, t_sd_s, \ldots, t_kd_k) + \{(i_1, \ldots, i_k) \mid i_s = 0 \text{ and } 0 \le i_j \le d_j \text{ for } j \ne s\}$$

, which precisely defines the low-end face L_s of $B(t_1, \ldots, t_s, \ldots, t_k)$. □

The lemma and the argument preceding it show that it is sufficient to work with just the faces of the blocks and that we might as well build the substructure consisting only of the faces of the constituent $B(t_1, \ldots, t_k)$ of T. After all, we are not interested in the contents of every single entry of T but just in $T(n_1, \ldots, n_k)$ and this entry lies exactly in the outermost position on the high-end faces of $B(\frac{n_1}{d_1} - 1, \ldots, \frac{n_k}{d_k} - 1)$.

Use the following algorithm to fill T. First fill $T(0, *, \ldots, *)$, $T(*, 0, *, \ldots, *)$ and so on up to $T(*, \ldots, *, 0)$ as if it were (k-1)-dimensional tables, using the method of section 3. Next "fill" the blocks $B(t_1, \ldots, t_k)$ in an order determined by the enumeration of all applicable tuples $(t_1, \ldots, t_k)$ with "rightmost coordinates varying fastest". In this way we achieve that whenever a next block must be filled its low-end faces are available. Observe also that we do not need the interior of blocks anymore, as only their high-end faces are of interest for later computations. Let $D = \prod_1^k d_j$, $E = \sum_1^k d_1 \ldots \not{d_s} \ldots d_k = (\frac{1}{d_1} + \ldots + \frac{1}{d_k})D$ and let A have Q states.

The computation of each block can be viewed as mapping the relevant sections of d_j symbols from u_j ($1 \le j \le k$) and d symbols from w and the contents of the appropriate low-end faces (a list of E symbols alltogether) to a list (of again E symbols long) representing the contents of the high-end faces. Time can be saved if we precompute this mapping and fill T by table

look-up. A convenient way to represent the mapping is to build a tree TR with labeled edges, such that lists of symbols leading to leaves read as arguments and associated pointers refer to mapped values (length E lists).

Lemma 4.3. TR can be constructed in $O(k.\sigma^{2d}.Q^E.D)$ steps on a random access machine.

Proof

There are $\sigma^{d_1} \ldots \sigma^{d_k} \cdot \sigma^d.Q^E = \sigma^{2d}.Q^E$ possible arguments that must be entered, using standard enumerations of faces. It takes about $2d + E = O(E)$ steps to lay out a single argument so the block-filling method of section 3 may be applied. It takes $O(k.D)$ to fill the block (compare 3.3.) and another $2d + E + O(E)$ steps to actually enter the argument and the resulting high-end faces as a value. Thus we spend about $2d + E + O(kD) + 2d + E + O(E) = O(kD)$ steps for each argument. □

A more practical method would not list all lists of length d_j over Σ_j into the arguments, but first compile a inventory of the distinct substrings $u_j[td_j+1] \ldots u_j[(t+1)d_j]$ there actually are. It shows that our estimates are crude especially when $\lceil n_j/d_j \rceil$ is "small" compared to σ^{d_j}. This applies to the exhaustive consideration of all possible fillings of low-end faces as well, but here no apparent saving can be made. In general one should list arguments as soon as they come up in the computation but no sooner, to avoid unnecessary work. Look-ups in TR take $2d + E + E = O(E)$ steps.

Lemma 4.4. Once TR has been constructed, the contents of $T(n_1, \ldots, n_k)$ can be determined in $O(kn^{k-1} + E.\prod_i^k \lceil n_j/d_j \rceil)$ steps.

Proof

$O(kn^{k-1})$ steps are needed to fill all entries $T(*, \ldots, 0, \ldots, *)$ with at least one coordinate zero. (Just run the (k-1)-dimensional algorithm of section 3 k times.) The remainder of the algorithm visits the $\prod_1^k \lceil n_j/d_j \rceil$ blocks one after the other and spends a total of $O(E)$ steps per block to determine the precise argument, do the look-up in TR and read out the contents of the high-end faces. □

Suppressing constants of proportionality depending on k only, the entire recognition algorithm takes

$$\sigma^{2d}.Q^E.D + n^{k-1} + (\frac{1}{d_1} + \ldots + \frac{1}{d_k}) \prod_1^k n_i$$

steps. More precise estimates will depend on knowledge of n_1 to n_k.

Theorem 4.5. Given a discrete finite-state transduction R of arity $k \geq 2$, one can decide whether $\langle u_1, \ldots, u_k, w\rangle$ belongs to R in $O(n^k/\log^{\frac{1}{k-1}} n)$ steps on a random access machine (where $|w| = n$).

Proof

Extend u_1 to u_k by \$-signs to give all strings length n. Take $\alpha = \left(\frac{1}{\log 2Q\sigma^2}\right)^{1/k-1} . \log^{1/k-1} n$ and let n be large enough so $\alpha \geq 4$. Hence $2\log\alpha \leq \alpha^{k-1} \cong \log n/\log 2Q\sigma^2$. Choose $d_1 = \ldots = d_k = \alpha$ (rounded) in the preceding algorithm. It yields a recognition algorithm that takes time proportional to

$$\sigma^{2k\alpha} . Q^{k\alpha^{k-1}} . \alpha^k + n^{k-1} + \frac{k}{\alpha} n^k \cong$$

$$\cong (Q\sigma^2)^{k\alpha^{k-1}} . \alpha^k + O\left(n^k/\log^{\frac{1}{k-1}} n\right) \cong$$

$$\cong (Q\sigma^2)^{k\alpha^{k-1}} . 2^{(k+1)\log\alpha/\alpha} + O\left(n^k/\log^{\frac{1}{k-1}} n\right) \cong$$

$$\cong (2Q\sigma^2)^{k\alpha^{k-1}} . \frac{1}{\alpha} + O\left(n^k/\log^{\frac{1}{k-1}} n\right) \cong$$

$$\cong n^k/\alpha + O\left(n^k/\log^{\frac{1}{k-1}} n\right) \cong$$

$$\cong O\left(n^k/\log^{\frac{1}{k-1}} n\right)$$

. □

Corollary 4.6. Elements of k-ary rational relations can be recognized in $O(n^k/\log^{\frac{1}{k-1}} n)$ steps on a random access machine.

5. Final comments.

Rational relations are of interest in automata theory and, more recently, in describing the synchronisation of processes. The recognition algorithm in section 3 was obtained through a common dynamic programming approach. The saving of a factor of $\log^{\frac{1}{k-1}} n$ in recognition time was obtained by economizing in the construction and eliminating repetitious computations. The technique as such has been applied before (see e.g. Paterson [3]) and may be of wider use in complexity studies. Note that the set $\{\langle u_1, u_2, w\rangle \mid w \text{ is a shuffle of } u_1 \text{ and } u_2\} \subset \Sigma^* \times \Sigma^* \times \Sigma^*$ is an example of a discrete finite-state transduction. Hence shuffles can be recognized in $O(n^2/\log n)$ rather than $O(n^2)$ time on a random access machine. No better bound is presently known to us. As another example, take $\{\langle u_1, u_2, u_3, v\rangle \mid v \text{ is a shuffle of } u_1, u_2 \text{ and } u_3\}$. As a discrete finite-state transduction it can be recognized in $O(n^3/\log^{\frac{1}{2}} n)$ steps.

6. References.

[1] Aho, A.V., J. Hopcroft and J.D. Ullman, The design and analysis of computer algorithms, Addison-Wesley Publ. Comp., Reading, Mass., 1974.

[2] Nivat, M., Multi-morphisms and synchronisation of concurrent processes (in preparation).

[3] Paterson, M., referred to on p.673 of D.S. Hirschberg, Algorithms for the longest common subsequence problem, J.ACM 24 (1977) pp. 664-675.