THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

AN EFFICIENT ALGORITHM FOR CONTAINMENT PROBLEM

Vaclav Rajlich

CRL-TR-9-84

JANUARY 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1]Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author.

An Efficient Algorithm for Containment Problem

Vaclav Rajlich

Department of Computer and Communication Science

University of Michigan

Ann Arbor, MI 48109

Abstract

Containment problems are important problems, because verification of specifications can be formally stated as a containment problem. The paper introduces a necessary and sufficient condition for containment of two languages. Disjoint-future automata are defined, and a one-pass algorithm for the containment problem is introduced.

1. INTRODUCTION

The containment problems can be characterized in the following way: Let P and Q be automata, and $L(P)$ and $L(Q)$ be languages they accept, respectively, then does $L(P) \subset L(Q)$? There is considerable literature which deals with containment problems; for overview, see [1,3].

It should be noted that the containment problems play a very important role in the context of the so-called specifications and top-down program design. In that situation, the behavior of data structures, subsystems, modules, etc. is specified by a specification language. The language specifies what is a correct sequence of data structure access, or what is a correct sequence of procedure calls, etc. This is particularly important in top-down program design [5,6], where the only thing known about lower modules may be the specifications, i.e. the sequencing of the procedure calls. When we want to debugg the design, then we have to rely on specifications as the only aid. For an overview of regular languages as specification languages, see [2,9].

To prove that a program satisfies the specifications means to prove that the set of all traces of that program is a subset of the specification language, i.e. it means to solve the containment problem. However in their full generality, containment problems are difficult problems; even for nondeterministic finite-state automata, the problem is *P*-SPACE complete and hence it is intractable. Therefore it is important to find special cases which make the problem easier to solve. One of these cases was investigated in [7], where it was proved that containment of regular languages in deterministic context-free languages is solvable in polynomial time. Results of this kind are important because the specifications are usually given as deterministic automata, while the analyzed program is given in its uninterpreted form, formally defined as a nondeterministic automaton.[8]

In this paper, we shall deal with a containment problem in situations where specifications are given as deterministic automata, and derive a necessary and sufficient condition for containment in the situations. In cases of disjoint-future automata, this condition leads to a linear one-pass algorithm for the containment problem

Section 2 of the paper contains all relevant definitions and observations. Section 3 contains the main result of the paper. Section 4 contains corollaries, including the one-pass algorithm for disjoint-future automata.

2. DEFINITIONS

Let us start this chapter with a definition of automaton:

Definition 2.1

Automaton is a 5-tuple $P = < A, K, f, s, M >$, where A is an ***alphabet,*** i.e. a set of ***symbols,*** K is a (possibly infinite) set of ***states,*** $f: K \times A \to 2^K$ is ***transition function,*** $s \in K$ is ***starting state,*** and $M \subset K$ is set of ***final states.***

The transition function can be extended in the following way:

Definition 2.2

Let λ denote empty word, A^* denote all words over A. Whenever no ambiguity arises, denote singletons as $a = \{a\}$. Then f can be extended to $f: K \times A^* \to 2^K$ in the following way: $f(q,\lambda) = q$, and for $a \in A$,

$$f(q,w \cdot a) = \{\, p \mid \text{there exists } r \in f(q,w) \text{ such that } p \in f(r,a) \,\}.$$

We say q is ***reachable*** from r (in automaton P), denoted $r \underset{P}{\leq} q$, iff there exists $w \in A^*$ such that $q \in f(p,w)$. Note that relationship $\underset{P}{\leq}$ is reflexive and transitive. If

P is obvious from the context, the same relationship will be denoted by $\leq$. If N is a set of states, then $p \leq N$ means there exists $r \in N$ such that $p \leq r$, and $N \leq p$ means there exists $r \in N$ such that $r \leq p$.

In the following definition, we shall define several notions:

Definition 2.3

Let $P = < A, k, f, s, M >$ be an automaton. Then *set of traces* (which bring P from $G \subset K$ to $H \subset K$) is defined as

$$T(P,G,H) = \{ t \mid p \in G, q \in f(p,t), \text{and } q \in H \} .$$

Set of states reached by P from $G \subset K$ and traces $T \subset A^*$ is defined in the following way:

If there is $g \in G$, $t \in T$ such that $f(g,t) = \Phi$, then $K(P, G, T) = \Phi$, otherwise

$$K(P,G,T) = \{ q \mid p \in G, t \in T, \text{and } q \in f(p,t) \} .$$

Language accepted by P is defined as $L(P) = T(P,s,M)$. *Future* of state $p \in K$ is defined as $F(P,p) = T(P,p,M)$.

The following lemma captures a property of the previous notions:

Lemma 2.1

Let $P = < A, K, f, s, M >$ be an automaton, and let $s \leq q \leq M$. Then for every $v \in F(P,q)$ and $m \in T(P,s,q)$, $m \cdot v \in L(P)$.

Proof obvious.

□

In the proof of the main result, we shall use the following notion:

Definition 2.4

Automaton $P = < A, K, f, s, M >$ is *prefixed* iff for every state $q \in K$, every input $a \in A$, $s \notin f(q,a)$.

In the following lemma, we show a property of prefixed automata:

Lemma 2.2

For a prefixed automaton $P' = < A, K, f, s, M >$, $T(P', s, s) = \lambda$.

Proof: Obvious.

□

An important subclass of automata are the deterministic automata of the following definition:

Definition 2.5

Automaton $P = < A, K, f, s, M >$ is *deterministic* iff for every $r \in K$, $a \in A$, there exists at most one q such that $q \in f(r,a)$.

For deterministic automata, we have the following lemma:

Lemma 2.3

Let $P = < A, K, f, s, M >$ be a deterministic automaton, and let $r \in K$, and $v, m \in A^*$, $f(s,m) = r$. Then if $f(r,v) = \Phi$, then $m \cdot v \notin L(P)$.

Proof

For $v = \lambda$, the lemma is vacuously true. For $v \neq \lambda$, and $m = \lambda$, the lemma is obviously true.

For induction step, suppose the lemma is true for m and $v = a \cdot v'$, where $a \in A$. Assume $f(r, a) = q$, then $f(q,v') = \Phi$, and $m \cdot a \cdot v' \notin L(P)$ by the induction assumption.

□

3. THE MAIN RESULT

This section contains the main results of the paper and its proof.

Theorem 3.1

Let $P = < A, K, f, s, M >$ be an arbitrary automaton, and let $Q = < A, K_Q, f_Q, s_Q, M_Q >$ be a deterministic automaton. Then $L(P) \subset L(Q)$ iff for every q for which $s \underset{P}{\leq} q \underset{P}{\leq} M$,

$$F(P,q) \subset \{w \mid \text{for every } r \in K(Q, s_Q, T(P,s,q)),\ w \in F(Q,r)\} .$$

Proof will be done in three steps. In the first step, we will prove that if the Theorem 3.1 is true for prefixed automata, then it is true for all automata also:

(i) We will construct prefixed automata P' , Q' in the following way:

Let $b \notin A$, $s' \notin K$, then

$$P' = < A \cup \{b\}, K \cup \{s'\}, f', s', M>$$

where for every $p \in K$, $a \in A$, $f'(p,a) = f(p,a)$ and $f'(s',b) = s$.

Let $s'_Q \notin K_Q$, then

$$Q' = < A \cup \{b\}, K_Q \cup \{s'_Q\}, f'_Q, s'_Q, M_Q >$$

where for every $q \in K_Q$, $a \in A$, $f'_Q(q,a) = f_Q(q,a)$, and $f'_Q(s'_Q, b) = s_Q$.

Now let us assume that Theorem 3.1 is true for the prefixed automata, i.e. we will suppose that it is true for P' and Q'. Take any $q \neq s'$, then

$$F(P',q) \subset \{w \mid \text{for every } r \in K(Q', s'_Q, T(P', s', q)),\ w \in F(Q', r)\}.$$

Then the following equalities are easy to verify:

$$F(P',q) = F(P,q),$$

$$F(Q',r) = F(Q,r),$$

$$T(P', s', q) = b \cdot T(P,s,q),$$

$$K(Q', s'_q, b \cdot T(P,s,q)) = K(Q, s_q, T(P,s,q))$$

and this proves step (i).

(ii) Suppose that for every q for which

$$s \underset{P}{\leq} q \underset{p}{\leq} M,\ F(P,q) \subset \{w \mid \text{for every } r \in K(Q, s_Q, T(P, s, q)),\ w \in F(Q,r)\}.$$

We will prove that $L(P) \subset L(Q)$.

Obviously from Definition 2.3, $L(P) = F(P,s)$, and by the assumption,

$$F(P,q) \subset \{w \mid \text{for every } r \in K(Q, s_Q, T(P, s, s)),\ w \in F(Q,r)\}.$$

Then: $T(P, s, s) = \lambda$ by Lemma 2.2, $K(Q, s_Q, \lambda) = s_Q$, and $F(Q, s_Q) = L(Q)$, which proves this implication.

(iii) In this step, we will prove the opposite implication. Proof will be done by contradiction.

Suppose $L(P) \subset L(Q)$ and there exists q, for which $s \leq_p q \leq_p M$ and

$$F(P,q) \not\subset \{w \mid \text{for every } r \in K(Q,s_Q,\ T(P,s,q)),\ w \in F(Q,r)\}\ .$$

This means that there exists $v \in F(P,q)$ and $r \in K(Q,\ s_Q,\ T(P,s,q))$ such that $v \notin F(Q,r)$. Let $m \in T(P,s,q)$ such that $f_Q(s_Q,m) = r$. Then by Lemma 2.1, $m \cdot v \in L(P)$. However by Lemma 2.3, $m \cdot v \notin L(Q)$, which is the contradiction with the assumption.

□

4. DISJOINT-FUTURE AUTOMATA

The Theorem 3.1 can be used for speed-up of containment algorithms. As an example, we will consider a special class of automata, the so-called disjoint-future automata of the following definition.

Definition 4.1

Let $P = < A, K, f, s, M >$ be an automaton. It is a *disjoint-future automaton* iff it is deterministic and $r, q \in K, r \neq q$ implies $F(P,r) \cap F(P,q) = \Phi$.

The definition is illustrated by the following example:

Example 4.1

Consider the automaton $P = < A, K, f, s, M >$ when K is a set of all nonnegative integers, $M = s = 0$, $A = \{ <, > \}$, and for every $r \geq 0, j \geq 1$, $f(i, <) = i+1, f(j, >) = j-1$. Then for $i \neq j$, $F(P,i) \cap F(P,j) = \Phi$ and hence the automaton is disjoint-future. $L(P)$ consists of all well-formed bracket expressions like $<>$, $<<><>><>$, etc.

For disjoint-future automata, we have the following corollary:

Corollary 4.1

Let $P = < A, K, f, s, M >$ be an arbitrary automaton, and let $Q = < A, K_Q, f_Q, s_Q, M_Q >$ be a disjoint-future automaton. Then $L(P) \subset L(Q)$ iff for every q for which $s \underset{P}{\leq} q \underset{P}{\leq} M$, there exists a unique $r \in K(Q, s_Q, T(P,s,q))$.

Proof

Suppose $x \not\equiv y$ and $x, y \in K(Q, s_Q, T(P,s,q))$, then by Theorem 3.1, $F(P,q) = \Phi$ and hence it is not true that $s \underset{P}{\leq} q \underset{P}{\leq} M$.

□

On the basis of Corollary 4.1, we will define the following algorithm:

Algorithm 4.1

Let $P = < A, K, f, s, M >$ be an arbitrary automaton, and let $Q = < A, K_Q, f_Q, s_Q, M_Q >$ be a disjoint-future automaton. Suppose P is represented by a graph with K being the set of nodes, and f being represented by labeled arcs: if $q \in f(p,a)$, then arc $<p,a,q>$ will be in the graph. Then the containment problem $L(P) \subset L(Q)$ can be solved in the following way:

1. Find all reachable states of K, i.e. all states q such that there exists $t \subset A^*$ such that $q \in f(s,t)$. Label each state q by a corresponding state $r \in K_Q$ such that $r = f_Q(s_Q,t)$.
2. If any reachable state of K is labeled by two or more states of K_Q, then $L(P) \not\subset L(Q)$, otherwise $L(P) \subset L(Q)$.

The algorithm is in fact a variant of marking algorithm [4], with additional labeling of nodes (i.e. states) by states of K_Q. It's complexity is $0(m)$ where m is the number

of the arcs in the graph. Note that the algorithm is one-pass, because each arc will be traversed at most once.

5. REFERENCES

[1] Hopcroft, J.E., Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley, Reading, MA.

[2] Horejs, J., "Finite semantics: a technique for program testing," *Proc. 4th International Conference on Software Engineering,* (1979) pp.433-440.

[3] Hunt, H.B.III., Rosenkrantz, D.J., Szymanski, T.G., "On the equivalence, containment, and covering problems for the regular and context free languages," *J. of Computer and System Sci.* Vol.12, (1976), pp.222-268.

[4] Knuth, D.E., *The Art of Computer Programming Vol. 1: Fundamental Algorithms,* Addison-Wesley, Reading, MA 1968.

[5] Rajlich, V., "Stepwise refinement revisited," to be published in *The Journal of Systems and Software,* Vol.4, No.1, (1984).

[6] Riddle, W.E., Wileden, J.C., Sayler, J.H., Segal, A.R., Stavely, A.M., "Behavior modeling during software design," *IEEE Trans. Software Engineering,* Vol. SE-4, 283-292.

[7] Rosenkrantz, D.J., Hunt, H.B. III, "Polynomial algorithms for deterministic pushdown automata," *SIAM J. Comput.,* Vol.7, (1978), pp.405-412.

[8] Rutlege, J.D., "On Ianov's program schemata," *J. of the ACM,* (1964), 1-9.

[9] Shaw, A.C., "Software specification languages based on regular expressions," in *Software Development Tools,* Riddle, W.E., Fairley, R.E., eds. Springer-Verlag, New York (1980), pp.148-175.