Information Processing Letters 29 (1988) 57–66
North-Holland 30 September 1988

A LINEAR ALGORITHM TO SOLVE FIXED-POINT EQUATIONS ON TRANSITION SYSTEMS

André ARNOLD and Paul CRUBILLE

*Département d'Informatique *, Université de Bordeaux I, 351, Cours de la Libération, 33405 Talence, France*

Communicated by L. Kott
Received 12 February 1988
Revised 17 May 1988

This paper gives an algorithm to compute the least fixed-point of a system of equations over a transition system. This algorithm has a time complexity linear in the size of the transition system, thus improving the known algorithms which are quadratic.

Cet article donne un algorithme pour calculer le plus petit point fixe d'un système d'équations sur un système de transitions. Cet algorithme a une complexité en temps linéaire par rapport à la taille du système de transitions alors que les algorithmes connus sont quadratiques.

Keywords: Transition system, fixed-point equation, temporal logics, model-checker

1. Introduction

It is widely acknowledged that *transition systems* (i.e., directed graphs with labels attached to edges and vertices) are a convenient model for (concurrent) processes [4]. It is also widely acknowledged that some properties of processes can be expressed by formulas of some branching-time temporal logics. Then, the verification that a process satisfies some property amounts to verifying that the transition system associated with the process is a model (in the logical meaning) of the formula expressing this property.

In [1], Clarke et al. have shown that, for the branching-time temporal logic CTL, the verification that a given transition system is a model of a given formula can be performed in a time linear in the size of the transition system.

Extending the observation made by Sifakis [6], that properties of processes can be expressed by a branching-time temporal logic where temporal operators are least fixed-points of some recursive equations, Dicky [2] has proposed a system of verification of processes where a property is specified by a set of mutually recursive equations and where the verification of such a property consists in computing the least fixed-point of the corresponding set of equations. This is obviously related to the μ-calculus [5], and more precisely to the formulas of alternation depth 1 [3].

The known algorithms to compute the least fixed-points of systems of equations are quadratic in the size of the transition system [2,3]. However, although every property expressible in CTL (or in most of the

* Associated with C.N.R.S.

0020-0190/88/$3.50

classical branching-time temporal logics) can be defined by a system of equations, the difference between the complexity of model checking for CTL formulas (or formulas of similar logics) and the complexity of computing least fixed-points of systems of equations forbid, from a practical point of view, to consider CTL-like logics as fragments of μ-calculus.

Here we propose another algorithm which is linear in the size of the transition system. Thus, this algorithmic difference between CTL-like logics and alternation-depth-1-μ-calculus vanishes and it becomes possible to increase the power of model-checkers without increasing their time complexity.

Of course, this decrease of the time complexity is possible only by an increase of the space complexity! More precisely, the data structure used by Dicky's algorithm [2] for storing transition systems has to be extended in two directions:

- the data structure has to contain not only the transitions (edges) of the transition system but also the 'reverse' transitions;
- not only boolean variables but also counters are associated with each state (vertex) of the transition system.

Since these counters contain numbers of transitions, the size of the data structure is then multiplied by a factor proportional to the logarithm of the size of the transition system. Such an increase of the size due to the use of counters also appears in the linear-time Tarjan's algorithm for computing strongly connected components of a graph [7].

2. Transition systems

A (finite) transition system is a tuple $G = \langle S, T, \alpha, \beta \rangle$, where S is a finite set of states, T is a finite set of transitions, $\alpha : T \to S$ is the source mapping, and $\beta : T \to S$ is the target mapping.

Any labelling of states and transitions can be specified by adding to G a family $Z_1, \ldots, Z_n$ of subsets of S and a family $Z'_1, \ldots, Z'_m$ of subsets of T. Then, G will be said to be a parametrized transition system.

Let us consider the following sorted signature D with two sorts: σ, for states, and τ, for transitions. The operators of D are:

- the constants 0_σ, 1_σ of sort σ;
- the constants 0_τ, 1_τ of sort τ;
- the binary operators $\cup_\sigma$, $\cap_\sigma$ of sort $\sigma\sigma \to \sigma$;
- the binary operators $\cup_\tau$, $\cap_\tau$ of sort $\tau\tau \to \tau$;
- the unary operators A, A^*, B, B^* of sort $\tau \to \sigma$;
- the unary operators A', B' of sort $\sigma \to \tau$.

Given a transition system G, an operator ω of D is interpreted by ω_G in the following way:

$$(0_\sigma)_G = (0_\tau)_G = \emptyset, \qquad (1_\sigma)_G = S, \quad (1_\tau)_G = T,$$

$$(\cup_\sigma)_G : \mathscr{P}(S) \times \mathscr{P}(S) \to \mathscr{P}(S) \qquad (\cap_\sigma)_G : \mathscr{P}(S) \times \mathscr{P}(S) \to \mathscr{P}(S)$$
$$X_1, X_2 \mapsto X_1 \cup X_2, \qquad X_1, X_2 \mapsto X_1 \cap X_2,$$

$$(\cup_\tau)_G : \mathscr{P}(T) \times \mathscr{P}(T) \to \mathscr{P}(T) \qquad (\cap_\tau)_G : \mathscr{P}(T) \times \mathscr{P}(T) \to \mathscr{P}(T)$$
$$Y_1, Y_2 \mapsto Y_1 \cup Y_2, \qquad Y_1, Y_2 \mapsto Y_1 \cap Y_2,$$

$$A_G : \quad \mathscr{P}(T) \to \mathscr{P}(S) \qquad B_G : \quad \mathscr{P}(T) \to \mathscr{P}(S)$$
$$Y \mapsto \{\alpha(t) \mid t \in Y\}, \qquad Y \mapsto \{\beta(t) \mid t \in Y\},$$

$$A^*_G : \quad \mathscr{P}(t) \to \mathscr{P}(S) \qquad B^*_G : \quad \mathscr{P}(T) \to \mathscr{P}(S)$$
$$Y \mapsto \{s \in S \mid \forall t : s = \alpha(t) \Rightarrow t \in Y\}, \quad Y \mapsto \{s \in S \mid \forall t : s = \beta(t) \Rightarrow t \in Y\},$$

$$A'_G:\quad \mathscr{P}(S)\to\mathscr{P}(T) \qquad\qquad B'_G:\quad \mathscr{P}(S)\to\mathscr{P}(T)$$
$$X\mapsto\{t\in T\mid \alpha(t)\in X\}, \qquad\qquad X\mapsto\{t\in T\mid \beta(t)\in X\}.$$

It is clear that all these interpretations are monotonic for inclusion. It is also easily checked that

$$S-A_G(Y)=A_G^*(T-Y),\qquad S-B_G(Y)=B_G^*(T-Y),$$
$$T-A'_G(X)=A'_G(S-X),\qquad T-B'_G(X)=B'_G(S-X).$$

Now, let us consider two sets of 'parameters' : $\mathscr{Z}_p=\{z_1,\dots,z_p\}$ of sort σ, $\mathscr{Z}'_{p'}=\{z'_1,\dots,z'_{p'}\}$ of sort τ, and two sets of variables $\mathscr{X}_n=\{x_1,\dots,x_n\}$ of sort σ and $\mathscr{Y}_n=\{y_1,\dots,y_m\}$ of sort τ. If a subset $(z_i)_G$ of S (respectively $(z'_i)_G$ of T) is associated with every parameter z_i (respectively z'_i) (i.e., G is a $(\mathscr{Z}_p, \mathscr{Z}'_{p'})$-parametrized transition system), then with every well-formed term w over $D\cup\mathscr{Z}_p\cup\mathscr{Z}'_{p'}\cup\mathscr{X}_n\cup\mathscr{Y}_m$ we can associate a mapping

$$w_G:\mathscr{P}(S)^n\times\mathscr{P}(T)^m\to\begin{cases}\mathscr{P}(S) & \text{if } w \text{ is of sort } \sigma,\\ \mathscr{P}(T) & \text{if } w \text{ is of sort } \tau,\end{cases}$$

defined by induction on w. This mapping is obviously monotonic.

3. Systems of equations

Let us consider two fixed sets of parameters $\mathscr{Z}$ and $\mathscr{Z}'$ and two sets of variables $\mathscr{X}_n$ and $\mathscr{Y}_m$. A system of equations Σ over $\mathscr{Z}$, $\mathscr{Z}'$, $\mathscr{X}_n$, $\mathscr{Y}_m$ is a pair $(\Sigma_\sigma, \Sigma_\tau)$ of sets

$$\Sigma_\sigma=\{x_i=w_i\mid x_i\in\mathscr{X}\},\qquad \Sigma_\tau=\{y_i=w'_i\mid y_i\in\mathscr{Y}\},$$

where w_i (respectively w'_i) is a term of sort σ (respectively τ) over $D\cup\mathscr{Z}\cup\mathscr{Z}'\cup\mathscr{X}_n\cup\mathscr{Y}_m$.

For every $(\mathscr{Z}, \mathscr{Z}')$-parametrized transition system G, $(w_i)_G$ (respectively $(w'_i)_G$) is a monotonic mapping from $\mathscr{P}(S)^n\times\mathscr{P}(T)^m$ into $\mathscr{P}(S)$ (respectively $\mathscr{P}(T)$), hence $\Sigma_G=\langle(w_1)_G,\dots,(w_n)_G, (w'_1)_G,\dots,(w'_m)_G\rangle$ is a monotonic mapping from $\mathscr{P}(S)^n\times\mathscr{P}(T)^m$ into itself and thus has a least fixed-point

$$\mu\Sigma_G=\langle X_1,\dots,X_n, Y_1,\dots,Y_m\rangle.$$

A system of equations Σ is said to be *simple* if each equation has one of the following forms:

(1) $x=k$, where k is a constant or a parameter of sort σ,
(2) $x=x'\cup_\sigma x''$, (3) $x=x'\cap_\sigma x''$,
(4) $x=A(y)$, (5) $x=B(y)$,
(6) $y=k'$, where k' is a constant or a parameter of sort τ,
(7) $y=y'\cup_\tau y''$, (8) $y=y'\cap_\tau y''$,
(9) $y=A'(x)$, (10) $y=B'(x)$,
(11) $x=A^*(y)$, (12) $x=B^*(y)$.

Every system Σ is equivalent to a simple system Σ' in the following sense: $\mu\Sigma_G$ is a sub-vector of $\mu\Sigma'_G$. Moreover, the 'size' of Σ' is the same as the 'size' of Σ if we define the size of Σ as being the number of occurrences of elements of $D\cup\mathscr{Z}\cup\mathscr{Z}'$, which is linearly related to the length of Σ.

Therefore, the algorithm for computing least fixed-points of systems of equations will apply to simple systems.

4. The algorithm

The basic idea of the algorithm is the following: with every state $s \in S$ and every variable $x \in \mathscr{X}$ (respectively every transition $t \in T$ and every variable $y \in \mathscr{Y}$) we associate a boolean variable $s.x$ (respectively $t.y$), named an *attribute* of s (respectively of t). These attributes specify the sets $X_1, \ldots, X_n$, $Y_1, \ldots, Y_m$ associated with the variables $x_1, \ldots, x_n$, $y_1, \ldots, y_m$ by $X = \{s \in S \mid s.x = \mathit{true}\}$, $Y = \{t \in T \mid t.y = \mathit{true}\}$.

All these attributes are initialized to *false* and the algorithm searches the transition system, updating these attributes according to the equations until $\langle X_1, \ldots, X_n, Y_1, \ldots, Y_m \rangle$ is equal to $\mu\Sigma_G$. Because of the monotonicity of Σ_G, these attributes can be changed only from *false* to *true*, therefore the number of updatings is bounded by (number of equations) $\times$ (size of G). The original Dicky's algorithm repeats a depth-first-search algorithm (of time complexity linear in the size of G until stationarity of the attributes); hence, the complexity of this algorithm is quadratic in the size of G. Instead of repeating a depth-first-search algorithm we propose a search directed by previous updatings: only states and transitions whose attributes may be modified because of previous modifications have to be searched again.

The general form of this algorithm is

```
initialization;
for every A in S do visit(s)
```

where visit(s) is a recursive procedure.

Let us remark at this point that we use the control structure: **for every** s **in** S **do**; we shall also use the control structures **for every** t **such that** $\alpha(t) = s$ **do**, **for every** t **such that** $\beta(t) = s$ **do**. Provided an adequate data structure, these control structures can be implemented in such a way that they can be executed in a time linear in the number of elements to deal with, for example by the use of linked lists.

With every state $s \in S$, the following objects are associated:

- the attributes $s.x$ for every x in X, initialized to *false* with the exceptions mentioned below,
- a boolean 'constant' $s.z$, for every parameter z in $\mathscr{Z}$. This constant is *true* if $s \in (z)_G$, and *false* otherwise,
- for every variable x defined by an equation $x = A^*(y)$, a counter $s.C_x$ initialized to $\mathrm{Card}\{t \mid s = \alpha(t)\}$. The corresponding attribute $s.x$ will be initialized to *true* if $s.C_x = 0$, and to *false* otherwise,
- for every variable x defined by an equation $x = B^*(y)$, a counter $s.C'_x$ initialized to $\mathrm{Card}\{t \mid s = \beta(t)\}$. The corresponding attribute $s.x$ is initialized to *true* if $s.C'_x = 0$, and to *false* otherwise;
- a boolean variable s.modified initialized to *true*;
- a boolean variable s.stacked initialized to *false*.

With every transition $t \in T$, the following boolean variables are associated:

- the attributes $t.y$ for $y \in \mathscr{Y}$, initialized to *false*;
- a boolean variable t.modified initialized to *true*.

We sketch the procedure of initialization only for taking it into account in the study of the complexity of the whole algorithm.

initialization

```
for every s in S do
    begin
    initialize the boolean attributes of s;
    d := 0;
    for every t such that α(t) = s do
        begin
        initialize the boolean attributes of t;
        d := d + 1
        end;
    initialize the counters s.C_x to d
        and the corresponding attributes s.x;
    d := 0;
    for every t such that β(t) = s do d := d + 1;
    initialize the counters s.C'_x to d
        and the corresponding attributes s.x;
    end
```

With each equation e of Σ we associate a procedure whose argument is a state or a transition and which modifies the attribute defined by this equation.

- If e is $x = A^*(y)$, we define the procedure dec_e whose argument is a state:

$\text{dec}_e(s)$

```
s.C_x := s.C_x - 1;
if s.C_x = 0 then
  begin s.x := true;
        s.modified := true
  end
```

- If e is $x = B^*(y)$, then dec_e is defined in the same way, substituting C'_x to C_x.

With these procedures dec_e we define the auxiliary procedures $\text{count}_y(t)$ associated with every variable y such that there exists at least one equation $x = A^*(y)$ or $x = B^*(y)$.

Let $\{e_1, \ldots, e_k\}$ be the set of equations in the form $x = A^*(y)$, and $\{e'_1, \ldots, e'_{k'}\}$ the set of equations in the form $x = B^*(y)$, y fixed. Then:

$\text{count}_y(t)$

```
dec_{e_1}(α(t)); ... ; dec_{e_k}(α(t));
dec_{e'_1}(β(t)); ... ; dec_{e'_{k'}}(β(t))
```

- If e is $x = v$ where v is 0_σ, 1_σ or a parameter z, then equation$_e(s)$ is

```
s.x := v̄;
if s.x has been modified then
  s.modified := true.
```

where $\bar{v}$ is *false, true, s.z.*

- If e is $x = x' \cup_\sigma x''$ (respectively $x' \cap_\sigma x''$), equation$_e(s)$ is

```
s.x := s.x' or s.x'';
  (respectively s.x' and s.x'')
if s.x has been modified then
  s.modified := true
```

Now, state(s) is the sequence of equation$_e(s)$ for all equations e of these forms.

- If e is an equation of type (6), (7), (8) we define equation$_e(t)$ in the same way as equation$_e(s)$ for equations of type (1), (2), (3) and transition(t) as state(s) but with the following modification:

```
if t.y has been modified then
  begin t.modified := true;
        count_y(t)
  end
```

- If e is $y = A'(x)$ (respectively $y = B'(x)$), then equation$_e(t)$ is

```
t.y := α(t).x;  (respectively β(t).x)
if t.y has been modified then
  begin t.modified := true;
        count_y(t)
  end
```

and enter(t) is the sequence of equation$_e(t)$ for all such equations.

- If e is $x = A(y)$ (respectively $x = B(y)$), then equation$_e(t)$ is

```
α(t).x := α(t).x or t.y;
if α(t).x is modified then
  α(t).modified := true
```

(respectively the same with $\beta(t)$ substituted for $\alpha(t)$), and leave(t) is the sequence of all these equation$_e$(t).

Now we define:

update_transition(t)

```
enter(t);
while t.modified do
      begin t.modified := false;
            transition(t)
      end;
leave(t)
```

update state(s)

```
s.modified := false;
state(s);
for every transition t such that α(t) = s do
   begin update_transition(t);
         visit(β(t))
   end;
for every transition t such that β(t) = s do
   begin update_transition(t);
         visit(α(t))
   end;
```

visit(s)

```
if not s.stacked then
  begin s.stacked := true;
        while s.modified do update_state(s);
        s.stacked := false
  end
```

5. Partial correctness

We assume that the algorithm terminates. (This will be proved in the next section by giving an upper bound on its time complexity.)

Since the execution algorithm of this algorithm consists of a sequence of executions of equation$_e$, we denote by $s.x(n)$ and $t.y(n)$ the value of these attributes upon the termination of the nth execution of equation$_e$. We denote by $X(n)$ and $Y(n)$ respectively the sets of states and transitions they define.

Let us denote by $\mu(k)$ the vector $\langle X_1(k), \ldots, X_n(k), Y_1(k), \ldots, Y_m(k)\rangle$. The following properties are immediate:

- $X(0) = Y(0) = \emptyset$;
- $\mu(n)$ is an increasing sequence;
- for every k and every equation $x = w$ (or $y = w$) of type (1), (2), ..., (10), we have

$$\boxed{X(k+1) \subset w_G(\mu(k)) \quad \text{and} \quad Y(k+1) \subset w_G(\mu(k))}. \qquad (*)$$

Indeed, after execution of equation$_e(s)$ we have $s \in X(k+1)$ implies $s \in X(k)$ or $s \in w_G(\mu(k))$, hence, $X(k+1) \subset X(k) \cup w_G(\mu(k))$. Since $X(0) \subset w_G(\mu(0))$ and since w_G is monotonic, we get the result by induction on k. The same holds for Y.

- For every k and every equation $x = A^*(y)$ we have $s.C_x(k) = \text{Card}\{t \mid \alpha(t) = s \text{ and } t \notin Y(k)\}$ and $s.x(k) = \mathit{true}$ iff $s.C_x(k) = 0$ (similarly for equations $x = B^*(y)$). This is true for $k = 0$. As soon as a transition t is put in $Y(k+1)$, then $\alpha(t).C_x$, $\alpha(t).x$, $\beta(t).C_x$, $\beta(t).x$ are modified accordingly.

Hence, we get, for every k,

$$\boxed{X(k) = A^*(Y(k)) \quad \text{and} \quad X(k) = B^*(Y(k))}. \qquad (**)$$

If we denote by μ the least fixed-point of Σ_G and by $\bar{\mu} = \langle \bar{X}_1, \ldots, \bar{X}_n, \bar{Y}_1, \ldots, \bar{Y}_m\rangle$ the result of the algorithm (note that there exists some k such that $\bar{\mu} = \mu(k)$), we get the following lemma.

5.1. Lemma. $\bar{\mu} \subset \mu$.

Proof. Since $\mu(0) \subset \mu$ we can prove by induction that $\mu(k) \subset \mu$ for every k:

- For equations of type (1), (2), ..., (10), because of $(*)$:

$$X(k+1) \subset w_G(\mu(k)) \subset w_G(\mu) = X, \qquad Y(k+1) \subset w_G(\mu(k)) \subset w_G(\mu) = Y_0.$$

- For equations of type (11) and (12), because of $(**)$:

$$X(k+1) = A^*(Y(k+1)) \subset A^*(Y) = X, \qquad X(k+1) = B^*(Y(k+1)) \subset B^*(Y) = X. \qquad \square$$

Therefore, in order to prove $\bar{\mu} = \mu$ we need to prove that $\bar{\mu}$ is a fixed point of Σ_G. We already know, because of $(*)$ and $(**)$,

$$\bar{X} \subset w_G(\bar{\mu}), \qquad \bar{Y} \subset w_G(\bar{\mu})$$

for every equation $x = w$ of type (1), (2), ..., (10), and

$$\bar{X} = A^*(\bar{Y}), \qquad \bar{X} = B^*(\bar{Y})$$

for other equations.

Thus, it remains to prove that

$$w_G(\bar{\mu}) \subset \bar{X} \quad \text{and} \quad w_G(\bar{\mu}) \subset \bar{Y}$$

for every equation $x = w$ of type (1), (2), ..., (10).

The proof of this inclusion relies upon the following observations.

5.2. Fact. *For any state s,* update_state(s) *and* state(s) *are executed at least once; for any transition t,* update_transition(t) *and* transition(t) *are executed at least once.*

At the first execution of visit(s), because of the initialization to *true* of s.modified, and to *false* of s.stacked, update_state(s) is executed, and visit(s) is executed at least once for every state.

Since update_state(s) is executed at least once for every state s, update_transition(t) is executed at least once for every t and because of the initialization to *true* of t.modified, transition(t) is executed at least once.

5.3. Fact. *If an attribute of some state s is modified, then* update_state(s) *will be executed again later on. If an attribute of some transition t is modified, then* transition(t) *will be executed again later on.*

When $s.x$ is modified (and therefore s.modified becomes *true*) either s is stacked and, on resumption of visit(s), update_state(s) will be executed again, or s is not stacked, and then the modification has been performed by some update_transition(t) with $s = \alpha(t)$ or $s = \beta(t)$, and visit(s) will be executed afterwards and also update_state(s).

If $t.y$ is modified, this modification can take place only in an execution of enter(t) or transition(t) and then transition(t) will be executed afterwards at least once more again.

5.4. Fact. *If the execution of* update_transition(t) *modifies an attribute of* $\alpha(t)$ *or* $\beta(t)$, *then* update_transition(t) *will be executed again later on.*

Because of Fact 5.3, if an attribute of $s \in \{\alpha(t), \beta(t)\}$ is modified, then update_state(s) will be executed again and also update_transition(t).

Let us consider the last execution of update_state(s); the attributes of s are not modified by this execution and will not be modified later on, so for any variable x the unmodified attribute $s.x$ is *true* iff $s \in \overline{X}$. It follows that, for every equation e: $x = w$ of type (1), (2), (3), the last execution of equation$_e$(s) does not modify anything, hence it follows that

$$s \in \overline{X} \quad \text{iff} \quad s \in w_G(\bar{\mu}).$$

Since this is true for every s, we get $\overline{X} = w_G(\bar{\mu})$ for every equation $x = w$ of type (1), (2), (3).

Let us consider the last execution of transition(t). The attributes of t are not modified by this execution and will not be modified later on: if they are modified, it is by a new execution of update_transition(t) which contains no execution of transition(t), i.e., by the sequence enter(t); leave(t); but only enter(t) can modify these attributes and then transition(t) will be executed; a contradiction.

By the same reasoning as above we get $\overline{Y} = w_G(\bar{\mu})$ for every equation $y = w$ of type (6), (7), (8).

Let us consider an equation $e: x = A(y)$. equation$_e$(t) is executed only in leave(t); let us consider the last execution of equation$_e$(t) which occurs in the last execution of update_transition(t). Since it follows the last execution of transition(t), we have $t.y = \mathit{true}$ iff $t \in \overline{Y}$, and since the attributes of $\alpha(t)$ will not be modified we get

$$\alpha(t) \in \overline{X} \quad \text{iff} \quad \alpha(t) \in \overline{X} \text{ or } y \in \overline{Y}.$$

It is true for every t and we get $A(\overline{Y}) \subset \overline{X}$. Similarly, $B(\overline{Y}) \subset \overline{X}$ for every equation $x = B(y)$.

Finally, let us consider an equation $y = A'(x)$, and the last execution of the corresponding equation$_e$(t) in the last execution of update_transition(t). Since the attributes of $\alpha(t)$ remain unmodified, we get

$$t.y(k) = \mathit{true} \quad \text{iff} \quad \alpha(t) \in \overline{X},$$

hence $\alpha(t) \in \overline{X} \Rightarrow t \in Y(k) \subset \overline{Y}$. Since this is true for every t we get $A'(\overline{X}) \subset \overline{Y}$. Similarly, for any equation $y = B'(x)$ we get $B'(\overline{X}) \subset \overline{Y}$.

6. Complexity

Let

$$d^+(s) = \text{Card}\{t \mid \alpha(t) = s\} \quad \text{and} \quad d^-(s) = \text{Card}\{t \mid \beta(t) = s\}.$$

If the control structures **for every t such that** can be executed in a time linear in the number of transitions, the time to execute update_state(s) is $k_0 + k_1 d^+(s) + k_2 d^-(s) + k_3 \Sigma_t k_s(t) +$ the time to execute some visit($\alpha(t)$), visit($\beta(t)$), where k_0, k_1, k_2 are constants (linearly) depending on the size of Σ, k_3 is the time to execute transition(t) which is a constant linearly depending on the size of Σ and $k_s(t)$ is the number of executions of transition(t) in update_transition(t).

But visit($\alpha(t)$), visit($\beta(t)$) results in executing update_state. Thus, let k_i be the time to execute initialization, $k(s)$ the number of times the procedure update_state(s) is entered, and $k(t)$ the number of times the procedure transition(t) is entered. Then, the time to execute the algorithm is

$$c = k_i + \sum_s k(s)(k_0 + k_1 d^+(s) + k_2 d^-(s)) + \sum_t k(t) k_3.$$

But, $k(s)$ is bounded by the number n_σ of attributes of states and $k(t)$ by the number n_τ of attributes of transitions, hence

$$c \leqslant k_i + \sum_s n_\sigma (k_0 + k_1 d^+(s) + k_2 d^-(s)) + \sum_t n_\tau k_3.$$

But, $\Sigma_s d^+(s) = \Sigma_s d^-(s) = |T|$, thus

$$c \leqslant k_i + |S| n_\sigma k_0 + n_\sigma (k_1 + k_2)|T| + k_3 n_\tau |T|.$$

Now,

$$k_i = \sum_s k'_0 + k'_1 d^+(s) + k'_2 d^-(s) = k'_0 |S| + (k'_1 + k'_2)|T|.$$

Since the constants k_i and k'_i depend on the size $|\Sigma|$ of the system of equations, and since n_σ and n_τ are bounded by $|\Sigma|$, we get

$$c \leqslant K |\Sigma|^2 (|S| + |T|).$$

Hence, we have the following result.

Theorem. *The least fixed-point of a system Σ of equations over a transition system G can be computed in time bounded by $K|\Sigma|^2|G|$.*

References

[1] E.M. Clarke, E.A. Emerson and A.P. Sistla, Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach, *ACM Trans. Programming Languages & Systems* **8** (1986) 244–263.

[2] A. Dicky, An algebraic and algorithmic method of analyzing transition systems, *Theoret. Comput. Sci.* **46** (1986) 285–303.

[3] E.A. Emerson and C.-L. Lei, Efficient model checking in fragments of the propositional μ-calculus, In: *Symp. on Logic in Computer Science*, Cambridge, MA (1986) 267–278.

[4] R.M. Keller, Formal verification of parallel programs, *Comm. ACM* **19** (1976) 371–384.

[5] D. Kozen, Results on the propositional μ-calculus, *Theoret. Comput. Sci.* **27** (1983) 333–354.

[6] J. Sifakis, A unified approach for studying the properties of transition systems, *Theoret. Comput. Sci.* **18** (1982) 227–258.

[7] R.E. Tarjan, Depth first search and linear graph algorithms, *SIAM J. Comput.* **1** (1972) 146–160.