

Model Checking in the Modal μ -Calculus by Substitutions

K. Kalorkoti

Department of Computer Science, University of Edinburgh,
Edinburgh, Scotland, U.K. EH9 3JZ (kk@dcs.ed.ac.uk)

October 3, 2001

Abstract: We give a simple algebraic method for model checking in the modal μ -calculus and use it to deduce a game theoretic approach to the problem.

Keywords: Algorithms, concurrency, programming calculi.

1 Introduction

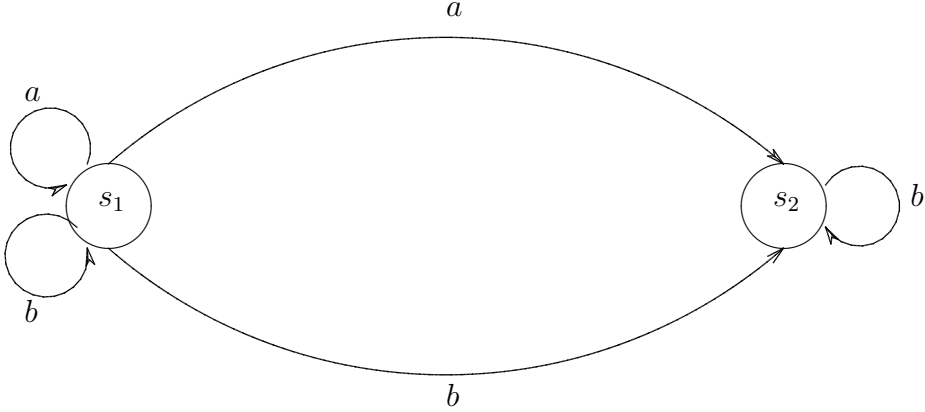
The modal μ -calculus was introduced by Kozen [5] and has been widely studied due to its ability to express a very wide range of interesting properties of finite state concurrent systems (e.g., liveness, safety and fairness). A key question is that of model checking: given an expression Ψ , labeled transition system T with initial state s , and valuation function V does Ψ hold at s ? Emerson and Lei [2] gave an algorithm based on the Tarski-Knaster fixed point theorem while Emerson Jutla and Sistla [3] showed that the problem is in $\text{NP} \cap \text{co-NP}$. Other approaches are based on translating the question to boolean equations combined with a fixed points approach, e.g., Andersen [1] or completely to boolean equations, Mader [6]. Finally Stirling [9] gives an approach based on games. In this note we provide a simple algebraic notation for a solution to the problem based on boolean equations and show that the approach based on games can be deduced quite easily from it. In fact the solution we give turns out to be essentially the same as that of Mader [6]. Our hope is that the simple notation provided will prove useful in theoretical studies of such boolean equation systems.

In the following we use 0 for ‘false’, 1 for ‘true’ and order these by $0 < 1$. Every formula of the modal μ -calculus can be put into positive form (see Emerson [4] or Stirling [8] for general background). Given such a formula Ψ , labeled transition system T and valuation V we can translate it to an equivalent system of boolean

equations of the form:

$$\begin{aligned}
\sigma_1 x_1 &= \Phi_1(x_1, \dots, x_n), \\
\sigma_2 x_2 &= \Phi_2(x_1, \dots, x_n), \\
&\vdots \\
\sigma_n x_n &= \Phi_n(x_1, \dots, x_n),
\end{aligned}
\tag{\dagger}$$

where each quantifier σ_i is one of μ, ν and each Φ_i is a monotone boolean function of x_1, \dots, x_n . For example let $\Psi = \mu Y_1. \langle a \rangle. \nu Y_2. (([b] Y_1 \vee Q) \wedge \mu Y_3. (Y_3 \wedge Y_2))$ and consider the transition system:



where the initial state is s_1 . First we translate the formula into

$$\begin{aligned}
\mu Y_1 &= \langle a \rangle Y_2 \\
\nu Y_2 &= ([b] Y_1 \vee Q) \wedge Y_3 \\
\mu Y_3 &= Y_3 \wedge Y_2.
\end{aligned}$$

Now we remove modalities by introducing new variables X_{ij} , one for each variable Y_i and state s_j and translate each equation at each state. An atomic proposition Q is replaced by 1 (i.e., true) if $Q \in V(s_j)$ and by 0 otherwise. In the following we assume that $V(Q) = \{s_2\}$ so that the preceding equations become:

$$\begin{aligned}
\mu X_{11} &= X_{21} \vee X_{22} \\
\mu X_{12} &= 0 \\
\nu X_{21} &= (X_{11} \wedge X_{12}) \wedge X_{31} \\
\nu X_{22} &= X_{32} \\
\mu X_{31} &= X_{31} \wedge X_{21} \\
\mu X_{32} &= X_{32} \wedge X_{22}.
\end{aligned}$$

A system (\dagger) denotes a unique solution given by:

1. If $n = 1$ set $S = \{b \mid b = \Phi_1(b)\}$ and note that S is not empty since Φ_1 is monotonic. If $\sigma_1 = \mu$ then we choose the least element of S otherwise the greatest.

2. If $n > 1$ then let $(a_{b_2}, \dots, a_{b_n})$ be the solution to the system consisting the last $n - 1$ equations with b in place of x_1 where $b = 0, 1$. Set $S = \{(b, a_{b_2}, \dots, a_{b_n}) \mid b = \Phi_1(b, a_{b_2}, \dots, a_{b_n})\}$. Again S is nonempty because Φ_1 is monotonic. If $\sigma_1 = \mu$ then we choose the member of S with least first coordinate otherwise the greatest.

This is a direct translation of the semantics for modal μ -calculus; the value of the first variable gives the value of the translated formula. The order in which the equations are given is important; from now on we will assume that the variables are ordered by $x_1 < x_2 < \dots < x_n$ which determines the order on the equations (in fact we need only impose a partial order as the example shows). Clearly such a system can be translated back to model checking and the translations each way are in log-space.

Finally we note that the log-space translation from boolean equations to model checking shows immediately that the problem is P-hard since a monotone boolean circuit can trivially be expressed as a set of such equations (see Papadimitriou [7] for background in complexity). Moreover the quantifiers are irrelevant, i.e., they do not affect the value of x_1 , so that the alternation free fragment is P-complete since Emerson and Lei [2] show that it has a polynomial time algorithm. These facts were observed by Zhang, Sokolsky and Smolka [10].

2 Solution by Substitutions

It will be convenient to have the following convention for a boolean formula Φ , variable x and quantifier $\sigma \in \{\mu, \nu\}$:

$$\Phi_{[\sigma/x]} = \begin{cases} \Phi_{[0/x]}, & \text{if } \sigma = \mu; \\ \Phi_{[1/x]}, & \text{if } \sigma = \nu. \end{cases}$$

Lemma 2.1 *Suppose that $\Phi(x)$ is a monotone boolean formula in the single variable x . Then the solution of $\sigma x = \Phi(x)$ is given by $\Phi_{[\sigma/x]}$.*

Proof Suppose that $\sigma = \mu$. The solution to $\mu x = \Phi(x)$ is 0 if and only if $\Phi(0) = 0$. Now if $\Phi(0) \neq 0$ then $\Phi(0) = 1$ and, since Φ is monotonic, $\Phi(1) = 1$. Thus the solution is given by $\Phi(0)$ in any case.

If $\sigma = \nu$ then the same argument applies but with the roles of 0 and 1 interchanged. \square

Given a system (\dagger) define a sequence of substitutions S_n, S_{n-1}, \dots, S_1 as follows:

1. S_n is given by

$$x_n \mapsto \Phi_{n[\sigma_n/x_n]}.$$

2. For $i = n - 1, n - 2, \dots, 1$, the substitution S_i is given by

$$x_i \mapsto (S_{i+1}S_{i+2} \cdots S_n \Phi_i)_{[\sigma_i/x_i]}.$$

Note that S_i only affects x_i and keeps all other variables fixed. Substitutions have the following simple properties:

1. $S_i x_j = x_j$ if $i \neq j$.
2. For any formula $\Phi(x_1, \dots, x_n)$ we have $S_i \Phi(x_1, \dots, x_n) = \Phi(S_i x_1, \dots, S_i x_n)$.
3. Let $b \in \{0, 1\}$ and $j \neq i$. Then for any formula $\Phi(x_1, \dots, x_n)$ we have $(S_i \Phi)_{[b/x_j]} = S_{i[b/x_j]} \Phi_{[b/x_j]}$, where $S_{i[b/x_j]}$ denotes the substitution that sends x_i to $(S_i x_i)_{[b/x_j]}$.

Theorem 2.1 *The solution to (\dagger) is given by*

$$x_i = S_1 S_2 \cdots S_i x_i,$$

for $1 \leq i \leq n$.

Proof We use induction on n . The base case $n = 1$ follows from Lemma 2.1. For the induction step recall that we can solve (\dagger) by taking each $b \in \{0, 1\}$ and solving

$$\begin{aligned} \sigma_2 x_2 &= \Phi_2(b, x_2, \dots, x_n), \\ \sigma_3 x_3 &= \Phi_3(b, x_2, \dots, x_n), \\ &\vdots \\ \sigma_n x_n &= \Phi_n(b, x_2, \dots, x_n), \end{aligned} \tag{\dagger}$$

to produce two solutions (a_{b2}, \dots, a_{bn}) . After this we choose between the two values of $(b, a_{b2}, \dots, a_{bn})$ according to the first equation.

For each value of b let $S_{bn}, S_{b,n-1}, \dots, S_{b2}$ be the substitution sequence given by (\dagger) . By induction the solution to (\dagger) is given by:

$$x_2 = S_{b2} x_2, x_3 = S_{b2} S_{b3} x_3, \dots, x_n = S_{b2} \cdots S_{bn} x_n.$$

From the simple properties of substitutions we have $S_{bi} = S_{i[b/x_1]}$, for $1 \leq i \leq n$, where S_n, S_{n-1}, \dots, S_1 is the sequence of substitutions given by (\dagger) . Thus, for $2 \leq i \leq n$, we have:

$$\begin{aligned} S_{b2} \cdots S_{bi} x_i &= S_{2[b/x_1]} \cdots S_{i[b/x_1]} x_i \\ &= (S_2 \cdots S_i)_{[b/x_1]} x_{i[b/x_1]} \\ &= (S_2 \cdots S_i x_i)_{[b/x_1]}. \end{aligned}$$

Now in order to choose the appropriate value of b we look at:

$$\begin{aligned}\Phi_1(b, (S_2x_2)_{[b/x_1]}, \dots, (S_2 \cdots S_n x_n)_{[b/x_1]}) &= \Phi_1(x_1, S_2x_2, \dots, S_2 \cdots S_n x_n)_{[b/x_1]} \\ &= (S_2 \cdots S_n \Phi_1)_{[b/x_1]}.\end{aligned}$$

Thus the appropriate value of b is the solution to:

$$\sigma_1 x_1 = S_2 \cdots S_n \Phi_1.$$

Thus, by Lemma 2.1, $b = (S_2 \cdots S_n \Phi_1)_{[\sigma_1/x_1]}$, i.e., $b = S_1 x_1$ while the value of x_i , for $2 \leq i \leq n$ is given by:

$$\begin{aligned}x_i &= (S_2 \cdots S_n x_i)_{[b/x_1]} \\ &= S_1 S_2 \cdots S_i x_i,\end{aligned}$$

and the proof is complete. \square

The theorem also shows that we can give a *generic* solution to the system (\dagger); we regard the quantifiers as unknowns and build the substitutions but leave out the replacement of x_i by σ_i at each stage.

3 Connection with Games

We extend the order on 0, 1 to a partial order on n -tuples of such values by component-wise comparison. This can then be extended to a partial order on boolean functions by declaring that $\Phi \leq \Psi$ if and only if $\Phi(\mathbf{a}) \leq \Psi(\mathbf{b})$ for all $\mathbf{a} \leq \mathbf{b}$. Given an n -tuple (Φ_1, \dots, Φ_n) of monotone boolean functions we can obtain another n -tuple of such functions by use of generic substitutions. The underlying operator is monotonic since it never uses negation. From this it follows that if we have two systems (\dagger) with the same sequence of quantifiers and the n -tuple of functions in the first is no larger than that of the second then their solutions are also related in the same way. (In fact we can generalize this to the situation where we put the obvious partial order on quantifiers and then we just require that the n -tuple of quantifiers of the first system is no larger than that of the second rather than that they should have the same quantifiers.)

Now we consider systems (\dagger) where each Φ_i is either a nonempty conjunction or nonempty disjunction of variables but not a mixture. This is not a restriction as any system can be put into this form with the introduction of extra variables and equations (without a large increase in the overall size). Note that for simplicity we assume, as we may, that there are no constants. We define an *or-refinement* of such a system to be a system with the same quantifiers but such that if Φ_i is a disjunction then it is replaced by exactly one of its arguments while if it is a conjunction then it is left unaltered. An *and-refinement* is defined similarly. Let

\mathbf{a}_0 , \mathbf{a}_1 be the solutions to an or-refinement and an and-refinement of a system (\dagger) respectively. Let \mathbf{a} be the solution of the system. Then it follows from the preceding paragraph that $\mathbf{a}_0 \leq \mathbf{a} \leq \mathbf{a}_1$. In fact it is easy to show by induction on n that there must be at least one refinement of each kind whose solution is exactly \mathbf{a} . (This is one way of seeing that the problem of determining the value of x_1 is in NP and co-NP.)

From now on we focus on the problem of determining the value of x_1 in a given system (\dagger). We shall refer to this value as b_1 in order to avoid confusion with the value of x_1 in refinements of (\dagger). If there is an or-refinement in which x_1 has value 1 then it follows from the preceding paragraph that $b_1 = 1$. The converse also holds. Thus $b_1 = 1$ if and only if x_1 has value 1 in at least one or-refinement of (\dagger). Now given an or-refinement we know that it must have an and-refinement with the same solution and all and-refinements yield an upper bound to the solution of the given or-refinement. Of course the system resulting from an and-refinement of an or-refinement is rather special in that each function on the r.h.s. of the equations consists of just a single variable. To sum up we have that $b_1 = 1$ if and only if there is an or-refinement of the system such that for all and-refinements x_1 has value 1. We can phrase the observation in the language of games: on each of his moves player A chooses a single argument for some disjunction and player B does the same for conjunctions. The claim that $b_1 = 1$ is then the same as the claim that player A has a winning strategy.

How do we recognize a win? One way is to use substitutions: this will run in time proportional to n^2 (at worst) since each substitution merely replaces one variable by another. In fact we can detect a winning strategy by using graphs as follows. Build a directed graph in which each vertex is labeled by a variable and for each equation $\sigma_i x_i = x_j$ introduce a directed edge (x_i, x_j) . Note that each vertex has precisely one successor. Now start at x_1 and follow the unique directed path until a vertex is met twice. Thus we now have a single circuit. Let x_ν be the smallest vertex in the circuit. The value of x_1 is then given by the quantifier σ_ν . The correctness of this method can be seen by following the effect of the substitutions in graph theoretic terms.

In fact given the or-refinement we can check in time proportional to n^3 if $x_1 = 1$. One way is to use substitutions. Alternatively we can apply the above analysis to obtain a graph theoretical solution. First of all we introduce the dependency graph of a system (\dagger) where we assume as above that each Φ_i is either a nonempty conjunction or disjunction of variables (and for simplicity we do not allow constants). We build a directed graph with vertices x_1, x_2, \dots, x_n as follows. For each equation $\sigma_i x_i = \Phi_i$ we introduce an edge (x_i, x_j) for each variable x_j appearing in Φ_i . Moreover if the right hand side is a conjunction then we say that vertex x_i is of and-type otherwise it is of or-type. We define R to be the set of all vertices x_i of the graph such that $\sigma_i = \nu$. Given a circuit of the graph we define the \hat{r} oot of the circuit to be the smallest vertex on it (recall that we have the order $x_1 < x_2 < \dots < x_n$). Now an or-refinement of (\dagger) simply

means that for each vertex of the dependency graph of or-type we keep exactly one outgoing edge to obtain a subgraph H . Given such an or-refinement the analysis above shows that x_1 has value 1 if and only if every circuit reachable from x_1 by a directed path in H has its root in R . We can decide this as follows: for each vertex $u \notin R$ reachable from x_1 by a directed path in H let H_u be the graph consisting of all vertices v of H such that $u \leq v$ and all edges of H with both endpoints in the vertices of H_u . Then x_1 is 1 if and only if for each such u the graph H_u has no circuits involving u . This discussion motivates the following decision problem for directed graphs:

GIVEN: A directed graph G on ordered vertices $x_1 < x_2 < \dots < x_n$ in which each vertex has outdegree at least one. Two subsets C and R of the vertices.

DECIDE: Is there a subgraph H of G , which is the same as G except that each vertex in C keeps exactly one outgoing edge, such that each circuit reachable from x_1 by a directed path in H has its root in R ?

We have shown that finding the value of x_1 in (\dagger) can be reduced (in log-space) to this problem. The converse is also true; given a graph as above we define an equation for each vertex as follows. If $x \in C$ then we introduce $\sigma x = \bigvee_y y$ where y ranges over all vertices such that (x, y) is an edge and $\sigma = \nu$ if $x \in R$ otherwise $\sigma = \mu$. If $x \notin C$ then define the equation similarly but use conjunction in the r.h.s. The ordering of the equations is given by the order on the vertices. Since G is the dependency graph of the equations it follows from the preceding discussion that $x_1 = 1$ if and only if the graph has the property described above with C consisting of all the vertices whose right hand side is a disjunction and R consisting of all vertices whose quantifier is ν .

The games approach and the use of graphs as described here was introduced by Stirling [9].

4 Solutions of the Underlying System

Given a system (\dagger) we can view it as a set of straightforward boolean equations by ignoring the quantifiers. We will refer to this as the underlying system of equations. Assume that (\dagger) is in the form described in §3 and let G be its dependency graph. Let C be the connected component of G that contains x_1 . Then it is clear that, assuming our interest is in the value of x_1 , we do not need any other equations so we assume that G is connected. Let (b_1, b_2, \dots, b_n) be the solution of (\dagger) . Certainly this is a solution of the underlying system. One possible approach to solving (\dagger) fast is to hope that the number of solutions of the underlying system is small and then devise some method to pick out the correct one for (\dagger) . More realistically we might hope to derive new conditions on the solution space that cut down the number of candidates. In fact this will certainly be necessary as the following example shows. Let $n > 0$ be an integer and define

B_n to be the following system in the $4n$ variables x_1, x_2, \dots, x_{4n} :

$$\begin{aligned}
x_1 &= x_2 \wedge x_{4n}, \\
x_2 &= x_1 \wedge x_3, \\
x_3 &= x_2 \vee x_4, \\
x_4 &= x_3 \vee x_5, \\
x_5 &= x_4 \wedge x_6, \\
x_6 &= x_5 \wedge x_7, \\
&\vdots \\
x_{4n-3} &= x_{4n-4} \wedge x_{4n-2}, \\
x_{4n-2} &= x_{4n-3} \wedge x_{4n-1}, \\
x_{4n-1} &= x_{4n-2} \vee x_{4n}, \\
x_{4n} &= x_{4n-1} \vee x_1.
\end{aligned}$$

The dependency graph for B_2 is *strongly* connected. However it is easy to see that B_n has at least 2^n solutions: setting the variables $x_3, x_4, x_7, x_8, \dots, x_{4n-1}, x_{4n}$ to 1 leaves a free choice for $x_1, x_5, \dots, x_{4n-3}$ (note that every solution must satisfy $x_1 = x_2, x_3 = x_4, \dots, x_{4n-1} = x_{4n}$). However even if we could cut down the underlying solutions to (\dagger) we are still left with the intriguing question of how to pick the correct one from them fast!

Acknowledgement: I would like to thank Colin Stirling for introducing the problem to me and for many discussions on the topic. I have also benefited from discussions with Angelika Mader.

References

- [1] H.R. Andersen, Model Checking and Boolean Graphs, in *Proc. ESOP'92*, LNCS 582, (1992) pp. 1–19.
- [2] E.A. Emerson and C.-L. Lei, Efficient Model Checking in Fragments of the Mu-calculus, *Proc. IEEE Symp. on Logic in Computer Science*, Cambridge, Mass., (1986) pp. 267–278.
- [3] E.A. Emerson, C.S. Jutla and A.P. Sistla, On Model Checking for Fragments of the Mu-calculus, in *Proc. 5th Inter. Conf. on Computer Aided Verification*, LNCS 697, (1993) pp. 385–396.
- [4] E.A. Emerson, Automated Temporal Reasoning about Reactive Systems, in *Logic for Concurrency, Structure versus Automata*, F. Moller and G. Birtwistle (Eds.), LNCS 1043, Springer, (1996) pp. 41–101.

- [5] D. Kozen, Results on the propositional μ -calculus, *Theor. Comp. Sci.*, **27** (1983) pp. 333–354.
- [6] A. Mader, Modal μ -Calculus, Model Checking and Gauß Elimination, in *Tools and algorithms for the construction and analysis of systems : first International Workshop, TACAS '95*, LNCS 1019, (1995), pp. 72–85.
- [7] C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, Mass., (1994).
- [8] C. Stirling, Modal and Temporal logics, in *Handbook of Logic in Computer Science*, D. Gabbay (Ed.), Oxford University Press, (1992).
- [9] C. Stirling, Local model checking games, in *CONCUR '95: Concurrency Theory*, LNCS 962, Springer, (1995) pp. 1–11.
- [10] S. Zhang, O. Sokolsky and S. A. Smolka, On the Parallel Complexity of Model Checking in the Modal Mu-Calculus, *Proc. 9th IEEE Symp. on Logic in Computer Science*, Paris, France., (1994) pp. 154–163.