

On the Parallel Complexity of Model Checking in the Modal Mu-Calculus*

Shipei Zhang, Oleg Sokolsky, Scott A. Smolka

Department of Computer Science

State University of New York at Stony Brook

Stony Brook, NY 11794 USA

Abstract

The modal mu-calculus is an expressive logic that can be used to specify safety and liveness properties of concurrent systems represented as labeled transition systems (LTSs). We show that *Model Checking in the Modal Mu-Calculus* (MCMC) — the problem of checking whether an LTS is a model of a formula of the propositional modal mu-calculus — is P-complete even for a very restrictive version of the problem involving the alternation-free fragment. In particular, MCMC is P-complete even if the formula is fixed and alternation-free, and the LTS is deterministic, acyclic, and has fan-in and fan-out bounded by 2. The reduction used is from a restricted version of the circuit value problem (Does a circuit α output a 1 on inputs x_1, \dots, x_n ?) known as *Synchronous Alternating Monotone Fanout 2 Circuit Value Problem*.

Our P-completeness result is tight in the sense that placing any further non-trivial restrictions on either the formula or the LTS results in membership in NC for MCMC. Specifically, we exhibit efficient NC-algorithms for two potentially useful versions of the problem, both of which involve alternation-free formulas containing a constant number of fixed point operators: 1) the LTS is a finite tree with bounded fan-out; and 2) the formula is \wedge -free and the LTS is deterministic and over an action alphabet of bounded size.

In the course of deriving our algorithm for 2), we give a parallel constant-time reduction from the alternation-free modal mu-calculus to Datalog. We also provide a polynomial-time reduction in the other direction thereby establishing an interesting link between the two formalisms. We conclude by pointing out an open problem involving the parallel complexity of model checking in CTL.

Coordinates of the corresponding author:

*Research supported in part by NSF Grants CCR-9120995 and CCR-9208585, and AFOSR Grant F49620-93-1-0250DEF.

Name: ShiPei Zhang
Address: Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400
USA

Phone: (+1) 516-632-7674
Fax: (+1) 516-632-8334
E-mail: szhang@cs.sunysb.edu

1 Introduction

The Concurrency Factory is a joint project between the State University of New York at Stony Brook and North Carolina State University to develop an integrated toolset for the specification, verification, and implementation of concurrent and distributed systems. Like the Concurrency Workbench [CPS93], the Factory employs bisimulation, preorder, and model checking as its main avenues of analysis.

A major underlying goal of the project is that the Factory be suitable for industrial application. One manner in which we are striving to achieve such applicability is through the parallelization of the analysis routines. For example, [ZS92] describes our efforts at parallelizing the bisimulation checking algorithm of [KS90].

This current paper is concerned with parallelizing the model checking routine of the Factory, or, more generally, the *parallel complexity of the propositional modal mu-calculus*. The modal mu-calculus is a highly expressive logic that can be used to specify safety and liveness properties of concurrent systems represented as labeled transition systems (LTSs). Syntactically, it consists of atomic propositions, standard logical connectives \wedge and \vee , dual modal operators \Box (necessarily) and \Diamond (possibly), and dual fixpoint operators μ (least fixed point) and ν (greatest fixed point). This logic is sometimes referred to as L_μ , and so it shall be here.

The modal mu-calculus is originally due to Kozen [Koz83] although in its current incarnation (and the one considered here), modalities are parameterized by action names [RRSV87, RdS90]. It can alternatively be viewed as the logic obtained by adding recursion to Hennessy-Milner Logic [HM85]. A number of different branching-time logics have uniform, linear-time encodings into L_μ , including CTL and CTL* [CES86], and Propositional Dynamic Logic [FL79].

The particular problem we study is called MCMC, for *Model Checking in the Modal Mu-Calculus*, the problem of checking whether an LTS is a model of a formula of the propositional modal mu-calculus, and a number of our results are specifically concerned with L_{μ_1} , the *alternation-free* fragment of L_μ [EL86]. Intuitively, a formula is in L_{μ_1} if the “level” (read *alternation depth*) of mutually recursive greatest and least fixed-point operators is one. When dealing with the alternation-free fragment, we will make use of the dialect of the modal mu-calculus considered in [CS93], which we refer to as *CS-logic*. Propositions in CS-logic are defined by least and greatest fixed points of mutually recursive systems of equations and the dependencies between systems are restricted in such a way that their logic coincides with L_{μ_1} .

We show that all our results still apply if recursion is specified through the explicit use of the μ and ν fixed-point operators, as in L_{μ_1} . Although expressively equivalent, CS-logic is much more succinct than L_{μ_1} : in general, an exponential blowup is suffered when translating a CS-proposition into L_{μ_1} . On the other hand, L_{μ_1} has a linear-time encoding into CS-logic.

Our results can be summarized as follows. We first show that MCMC is P-complete even for a very restricted version of the problem involving the alternation-free fragment. In particular, MCMC is P-complete even if the formula is fixed and alternation-free, and the LTS is deterministic, acyclic, and has fan-in and fan-out bounded by 2. The linear-time algorithm of [CS93] provides the necessary membership in P and we prove P-hardness via a reduction from a restricted version of the circuit value problem (Does a circuit α output a 1 on inputs x_1, \dots, x_n ?) known as *Synchronous Alternating Monotone Fanout 2 Circuit Value Problem* (SAM2CVP). Moreover, the recently discovered polynomial-time algorithm of [BC93] extends our P-completeness result to the full modal mu-calculus, i.e., L_μ .

Our P-completeness result is tight in the sense that placing any further non-trivial restrictions on either the formula or the LTS results in membership in NC for MCMC. In particular, we exhibit efficient NC-algorithms for two potentially useful versions of the problem, both of which involve alternation-free formulas containing a constant number of fixed point operators: 1) the LTS is a finite tree with bounded fan-out; and 2) the formula is \wedge -free and the LTS is deterministic and over an action alphabet of bounded size.

Our algorithm for 1) can be seen as a parallelization of the model checking algorithm of [CS93], using the parallel tree contraction algorithm of Miller and Reif [RMMM92] to perform computations on the *product graph* of the LTS and CS-logic formula. Our algorithm for 2) is obtained through a reduction to a Datalog program having the *polynomial fringe property* of Ullman and van Gelder [UvG88, Ull92]; their NC algorithm for evaluating Datalog programs can then be used. We show that an alternative algorithm for 2) can be obtained via a reduction to Proplog.

In the course of deriving our NC algorithm for 2), we give a parallel constant-time reduction from the alternation-free modal mu-calculus to Datalog. We also provide a polynomial-time reduction in the other direction thereby establishing some interesting links between the two formalisms.

In terms of related work, Balcazar et al. [BGS92] have shown that the problem of checking bisimulation of two finite-state LTSs is also P-complete. This is particularly relevant, for one can use this result as the basis for another proof of the P-completeness of MCMC: as shown in [CS91], bisimulation checking can be reduced to MCMC by model checking the *characteristic formula* [Ste89] of one of the LTSs against the other LTS. Moreover, it is not difficult to see that only log-space is needed. The resulting P-completeness result, however, is weaker than our own on a number of counts: the modal mu-calculus formula required is input-dependent (in our reduction, the formula is fixed), the structural restrictions we place on the LTS are not reflected in a reduction of this nature, and the reduction is not log-space for L_{μ_1} . It can also be argued that by reducing a fundamental P-complete problem such as the circuit value problem to MCMC, more is learned about the sequential nature of model checking in the modal-mu-calculus.

Other relevant work includes the vector-processing-based CTL model checking algorithms of [?, ?], the parallel BDD minimization algorithms of [KC90, LR93] (BDDs can be used in model checking to succinctly represent the LTS in question), the optimal parallel algorithms of [?] for bisimulation checking, and the body of literature on the parallel complexity of logics such as Datalog [Kan85, Ull92, AP93] (as stated above, there are close connections between Datalog and the alternation-free modal mu-calculus).

The structure of the rest of this paper is as follows. Section 2 defines labeled transition systems, the syntax and semantics of L_{μ} and CS-logic, and the model checking problem. Section 3 gives our P-completeness results, and Section 4 presents our NC algorithms. Finally, Section 5 concludes and identifies an open problem concerning the parallel complexity of CTL.

2 Definitions

A *Labeled Transition System* (LTS) is a 4-tuple $\langle \mathcal{S}, Act, \rightarrow, s_0 \rangle$ where \mathcal{S} is the set of *states*, Act is the set of *actions*, $\rightarrow \subseteq \mathcal{S} \times Act \times \mathcal{S}$ is the *transition relation* and s_0 is the *start state*.

We next define the syntax and semantics of CS-logic. The definition of L_{μ} will follow by

$$\begin{aligned}
[[A]]e &= \mathcal{V}(A) \\
[[X]]e &= e(X) \\
[[\Phi_1 \wedge \Phi_2]]e &= [[\Phi_1]]e \cap [[\Phi_2]]e \\
[[\Phi_1 \vee \Phi_2]]e &= [[\Phi_1]]e \cup [[\Phi_2]]e \\
[[[a]\Phi]]e &= \{s \mid \forall s'. s \xrightarrow{a} s' \Rightarrow s' \in [[\Phi]]e\} \\
[[\langle a \rangle \Phi]]e &= \{s \mid \exists s'. s \xrightarrow{a} s' \Rightarrow s' \in [[\Phi]]e\}
\end{aligned}$$

Figure 1: Semantics of the mu-calculus

extension. Formulas in CS-logic are of two types: basic formulas and equational blocks. The syntax of *basic formulas* is given by the following grammar:

$$\Phi ::= A \mid X \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid [a]\Phi \mid \langle a \rangle \Phi$$

where $A \in \mathcal{AP}$, a fixed set of atomic propositions, and $X \in \text{Var}$, a countably infinite set of variables.

Basic formulas are interpreted with respect to an LTS $\mathcal{L} = \langle \mathcal{S}, \text{Act}, \rightarrow, s_0 \rangle$, a *valuation mapping* $\mathcal{V} : \mathcal{AP} \rightarrow \mathcal{P}(\mathcal{S})$, relating every atomic proposition A to the set of states in which A holds, and an *environment* $e : \text{Var} \rightarrow \mathcal{P}(\mathcal{S})$, mapping each variable X to the set of states that satisfy X . Then the meaning of a basic formula is given by the semantical function $[[\cdot]] : \Phi \rightarrow \mathcal{P}(\mathcal{S})$, defined in Figure 1.

Equational blocks B are formed by either *min* or *max* operators applied to a set E of mutually recursive equations of the form

$$\begin{aligned}
X_1 &= \Phi_1 \\
&\vdots \\
X_n &= \Phi_n,
\end{aligned}$$

where each Φ_i is a basic formula and the X_i are distinct. Following [CS93], we further assume that Φ_i is *simple*, i.e., an atomic proposition, or constructed by the application of exactly one operator to variables. Every formula can be made simple with at most a linear blow-up in size.

Operators *min* and *max* are understood respectively as the least and greatest fixed points of the sets of equations. Finally, a *formula* $\mathcal{B} = \{B_1, \dots, B_m\}$ is a set of blocks, with the following syntactic restrictions: all variables appearing on the left-hand sides in the set of blocks are distinct, and the formula's block graph is acyclic. The *block graph* of \mathcal{B} is the directed graph with nodes B_1, \dots, B_m and edges $\langle B_i \rangle B_j$ whenever a variable appearing as a left-hand side of an equation in B_i is used in B_j (we say that B_j *depends* on B_i in this case). Restricting the block graph to be acyclic ensures that no alternating fixed points [EL86] can occur.

Semantically blocks are understood as functions from environments to environments. Let $\mathcal{B} = \{B_1, \dots, B_m\}$ with distinct variables X_1, \dots, X_N defined within it. Let $\overline{\mathcal{S}} = \langle S_1, \dots, S_N \rangle \in (2^{\mathcal{S}})^N$ and let $e_{\overline{\mathcal{S}}} = e[X_1 \mapsto S_1, \dots, X_N \mapsto S_N]$. Then the function

$$f_e(\overline{\mathcal{S}}) = \langle [[\Phi_1]]e_{\overline{\mathcal{S}}}, \dots, [[\Phi_N]]e_{\overline{\mathcal{S}}} \rangle,$$

defined on the lattice of tuples of sets of states ordered by point-wise set inclusion, is monotonic and continuous. By the Tarski fixed-point theorem, f_e has both least and greatest fixed points given by:

$$\begin{aligned}\nu f_e &= \bigcup \{\bar{S} \mid \bar{S} \subseteq f_e(\bar{S})\} \\ \mu f_e &= \bigcap \{\bar{S} \mid f_e(\bar{S}) \subseteq \bar{S}\}\end{aligned}$$

Blocks can now be interpreted as environments in the following fashion:

$$\begin{aligned}[[maxE]]e &= e_{\nu f_e} \\ [[minE]]e &= e_{\mu f_e},\end{aligned}$$

Then, the meaning $[[\mathcal{B}]]e$ of the formula \mathcal{B} containing blocks B_1, \dots, B_m , topologically sorted by the dependency relation, can be computed through a sequence of environments

$$\begin{aligned}e_1 &= [[B_1]]e \\ &\vdots \\ e_m &= [[B_m]]e_{m-1}\end{aligned}$$

with $[[\mathcal{B}]]e = e_m$. Due to the acyclicity restriction on block graphs, we are ensured that $[[\mathcal{B}]]e_m = e_m$.

If \mathcal{B} is a closed formula, i.e., every variable mentioned in the right-hand side of some equation appears on the left-hand side of an equation in one of the blocks, then for every two environments e and e' , we have $[[\mathcal{B}]]e = [[\mathcal{B}]]e'$. Now, for every variable X defined in the formula we can compute the set of states in which X holds as $[[X]][[\mathcal{B}]]$.

L_μ is obtained by extending the syntax of basic formulas with the fixed point operators μ and ν . Likewise, the semantics of L_μ is obtained by adding the following two clauses to Figure 1, where $e[X \mapsto S]$ denotes the environment that agrees with e on all variables except X , which is bound to S :

$$\begin{aligned}[[\mu X.\Phi]]e &= \bigcap \{S' \subseteq \mathcal{S} \mid [[\Phi]]e[X \mapsto S'] \subseteq S'\} \\ [[\nu X.\Phi]]e &= \bigcup \{S' \subseteq \mathcal{S} \mid S' \subseteq [[\Phi]]e[X \mapsto S']\}\end{aligned}$$

As mentioned in the Introduction, the *alternation depth* of an L_μ formula refers to the nesting level of mutually recursive μ and ν operators. Its formal definition is a bit technical and will appear in the full paper. The alternation-free subset of the mu-calculus, L_{μ_1} , contains all formulas Φ with alternation depth 1. As shown in [CS93], CS-logic coincides with L_{μ_1} .

Model-Checking in the Modal Mu-Calculus problem (MCMMC)

Given: A L_μ formula Φ and an LTS $\mathcal{L} = \langle \mathcal{S}, Act, \rightarrow, s_0 \rangle$.

Problem: Decide whether $s_0 \in [[\Phi]]$.

In the case of a CS-logic formula, the problem becomes one of deciding whether $s_0 \in [[X]][[\mathcal{B}]]$, for a designated variable X defined within \mathcal{B} .

3 The Modal Mu-Calculus is P-Complete

In this section, we present two P-completeness results for MMMC. The first is for the problem in its full generality, and is made possible, in part, by the recently discovered polynomial time-algorithm of [BC93]. The second is for a very restricted version of the problem involving the alternation-free fragment L_{μ_1} . The following P-complete circuit value problems are used to achieve our results.

Monotone Alternating Circuit Value Problem (MACVP) [?]

A *monotone* circuit is a circuit that consists only of \vee and \wedge gates. As the circuit does not contain negation gates, it receives each input together with its negation. A monotone *alternating* circuit is divided into levels so that the inputs to a gate on one level are all outputs of gates of the immediately preceding level. Further, in a monotone alternating circuit, all the gates on the same level are of the same type and the levels alternate between \vee and \wedge levels.

Given: An encoding $\bar{\alpha}$ of a monotone alternating circuit α , plus inputs x_1, \dots, x_n .

Problem: Does α on input x_1, \dots, x_n output 1?

Synchronous Alternating Monotone fanout 2 Circuit Value Problem (SAM2CVP) [GHR91]

A circuit in this problem must satisfy all the requirements for a circuit in MACVP plus the following. All inputs must be connected only to \vee gates and output must come from an \vee gate. The fanout for inputs and internal gates (i.e., non-output gates) must be exactly two. Internal gates also have a fan-in of 2.

Given: An encoding $\bar{\alpha}$ of a circuit α described above, plus inputs x_1, \dots, x_n .

Problem: Does α on input x_1, \dots, x_n output 1?

Theorem 3.1. *MMMC is P-complete.*

Proof Sketch. Membership in P is due to the polynomial-time algorithm of [BC93]. P-hardness is established by a log-space reduction from MACVP in which the circuit is represented by an LTS, and a fixed formula is chosen in such a way that the circuit outputs a 1 if and only if the LTS is a model of the formula.

The LTS $\langle \mathcal{S}, Act, \rightarrow, s_0 \rangle$ is constructed as follows. Every logic gate and input gate in the circuit α is turned into a state of the LTS; when there is no confusion, we will refer to the elements of \mathcal{S} by the names they have in α . \mathcal{S} also contains n auxiliary states, named y_1, \dots, y_n , one for each of the n input variables. $Act = \{a, 1\}$ and s_0 is the state that corresponds to the output gate in α .

We add $g_2 \xrightarrow{a} g_1$ to \rightarrow whenever the output of gate g_1 in α is fed into the input of gate g_2 . For each pair of inputs, x_i and \bar{x}_i , exactly one of them evaluates to 1. We add to \rightarrow the transition $x_i \xrightarrow{1} y_i$ if x_i is 1, and otherwise add $\bar{x}_i \xrightarrow{1} y_i$ to \rightarrow , where y_i is an auxiliary state mentioned earlier. \rightarrow contains no further transitions.

The circuit-independent L_μ formula is the following (tt is the atomic proposition that is true of every state):

$$\begin{aligned} \min\{X &= [a]\langle 1 \rangle tt \vee [a]Y \\ Y &= \langle a \rangle \langle 1 \rangle tt \vee \langle a \rangle X\} \end{aligned} \tag{1}$$

Intuitively, in order for the output of an \wedge -gate in a monotone alternating circuit to output 1, its inputs must be either all from input gates that are 1, or all from gates of the immediately

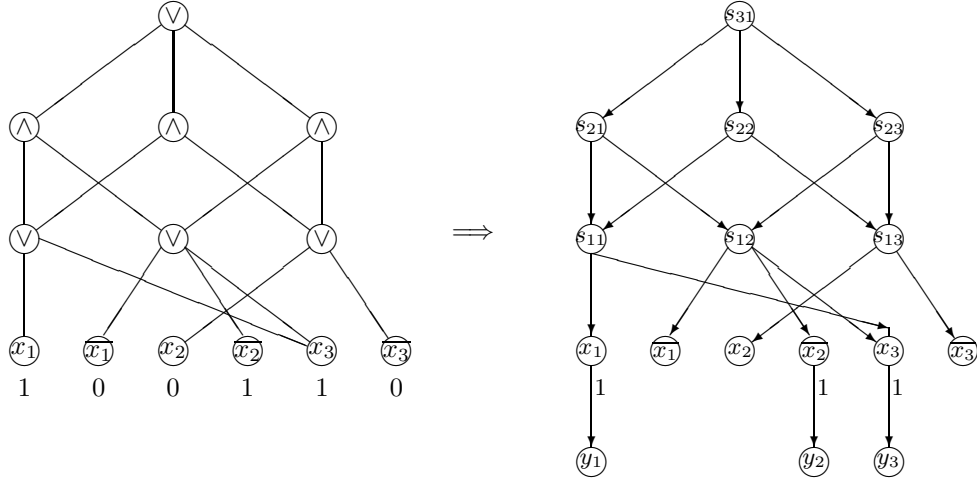


Figure 2: An example of the reduction from MACVP to MCMMC.

preceding level that output a 1. Dually for an \vee -gate.

Let $\llbracket X \rrbracket$ and $\llbracket Y \rrbracket$ denote the least fixed-point solution to equation (1). Intuitively, $\llbracket X \rrbracket$ contains all the states of the LTS that correspond to \wedge -gates that output 1 and none of the states that correspond to \wedge -gates that output 0. Similarly, $\llbracket Y \rrbracket$ contains those states corresponding to \vee -gates that output 1 but none of the states corresponding to \vee -gates that output 0. We claim that if the output gate of the circuit is an \wedge -gate, then the circuit outputs 1 if and only if $s_0 \in \llbracket X \rrbracket$; and if the output gate of the circuit is an \vee -gate, then the circuit outputs 1 if and only if $s_0 \in \llbracket Y \rrbracket$.

In the case of the LTS being finite, $\llbracket X \rrbracket$ and $\llbracket Y \rrbracket$ can be computed by setting $X_0 = \emptyset$, $Y_0 = \emptyset$ and $X_i = [a] \langle 1 \rangle tt \vee [a] Y_{i-1}$, $Y_i = \langle a \rangle \langle 1 \rangle tt \vee \langle a \rangle X_{i-1}$ until we reach a fixed point. The proof of the claim is by induction on i , the number of iterations of the fixed point computation. The inductive hypothesis states, among other things, that on the i -th iteration, only states corresponding to gates of level i in the circuit are added to the sets being computed. Thus by the end of the d th iteration, where d is the depth of the circuit, we will have correctly computed the least fixed point of the equational block. ■

Figure 2 illustrates the reduction process outlined above by showing a circuit on the left and its corresponding LTS on the right. Level 1 gates receive their inputs directly from the input gates. The output gate is the topmost \vee -gate. Input variables x_1, x_2, x_3 are assigned the values of 1,0,1, respectively. In the LTS on the right, transitions without an action label are assumed to have a label of a . The three auxiliary states associated with the three input gates that evaluate to 1 are at the bottom of the figure.

It is easy to verify that the circuit outputs a 1 and $s_{31} \in \llbracket Y \rrbracket$. The actual fixed point computation of $\llbracket X \rrbracket$ and $\llbracket Y \rrbracket$ exhibit the following sequence. $X_0 = \emptyset$, $X_1 = \{s_{11}\}$, $X_2 = \{s_{11}, s_{21}\}$, $X_3 = \{s_{11}, s_{21}, s_{31}\}$ and $Y_0 = \emptyset$, $Y_1 = \{s_{11}, s_{12}\}$, $Y_2 = \{s_{11}, s_{12}, s_{21}, s_{22}\}$, $Y_3 = \{s_{11}, s_{12}, s_{21}, s_{22}, s_{31}\}$.

Theorem 3.2. *MCMMC is P-complete even when the formula is fixed and alternation-free,*

and the LTS is deterministic, acyclic, and has fan-in and fan-out bounded by 2.

Proof Sketch. Membership in P is again due to [BC93] or, alternatively, the linear-time algorithm of [CS93]. For P-hardness, we reduce an arbitrary instance of SAM2CVP to an instance of MCMC. The fan-in (fan-out, resp.) of a gate in the circuit translates into the fan-out (fan-in, resp.) of a state in the LTS. To make the LTS deterministic, we label the two transitions out of each internal state a and b instead of labeling both by a . The LTS is acyclic because the circuit is acyclic.

Finally, formula(1) of the previous reduction has to be modified slightly to reflect the fact that an internal state now has a b -transition as well as an a -transition. The new formula is:

$$\begin{aligned} \min\{X &= [a]\langle 1 \rangle tt \wedge [b]\langle 1 \rangle tt \vee [a]Y \wedge [b]Y \\ Y &= \langle a \rangle \langle 1 \rangle tt \vee \langle b \rangle \langle 1 \rangle tt \vee \langle a \rangle X \vee \langle b \rangle X\} \end{aligned} \quad (2)$$

■

The modal mu-calculus formulas used in the above two reductions have been given in the syntax of CS-logic to facilitate the presentation of the proofs of the theorems. The reductions are still valid and log-space for L_{μ_1} since (1) and (2) can be easily translated into L_{μ_1} formulas. In general, the equational block $\min\{X = \Phi_1(Y), Y = \Phi_2(X)\}$ in CS-logic is equivalent to the L_{μ_1} formula $\mu X. \Phi_1(\Phi_2(X))$, assuming that we are interested in variable X of the block. The correctness of the translation follows from the semantics of equational blocks and the monotonicity of Φ_1 and Φ_2 .

4 NC Algorithms for the Modal Mu-Calculus

Next we present NC algorithms for another restricted version of MCMC.

Theorem 4.1. *MCMC is in NC when the alternation-free formula is an \wedge -free block, and the LTS is deterministic and over an action alphabet of bounded size.*

Proof Sketch. We show that the problem can be reduced to a Datalog problem in constant parallel time, using $O(|\mathcal{B}| + |S|)$ processors, where $|\mathcal{B}|$ is the number of variables in the block \mathcal{B} and $|S|$ is the number of states in the LTS. $s_0 \in \llbracket \mathcal{B} \rrbracket(X)$ if and only if the IDB fact of $T(s_0, X)$ can be proved in the Datalog problem.

Each equation in the block is translated into one or two EDB facts, according to the rules in the following table. The translation takes constant time if we assign one processor to each of the $|\mathcal{B}|$ equations. The size of the resulting EDB is $O(|\mathcal{B}|)$.

equation type	EDB facts generated
$X_k = A$	$F_{AP}(X_k, A)$
$X_k = X_i \vee X_j$	$F_{\vee}(X_k, X_i)$ and $F_{\vee}(X_k, X_j)$
$X_k = \langle a \rangle X_i$	$F_{\langle \rangle}(X_k, a, X_i)$
$X_k = [a]X_i$	$F_{[\]}(X_k, a, X_i)$

To encode the LTS, we assign one processor to each of the $|S|$ states in the LTS. Because the LTS is deterministic and the action alphabet is of bounded size, the number of outgoing transitions for every state is also bounded. Thus each processor can, in constant time, translates

every transition $s \xrightarrow{a} s'$ to an EDB fact of $L(s, a, s')$. Meanwhile for every action in the alphabet that doesn't appear as the label of any outgoing transition for s , an EDB fact of $L(s, a', s_{tt})$ is added. It is clear that the size of the resulting EDB is $O(|S|)$.

s_{tt} is a distinguished state different from any state in the LTS. Any variable in \mathcal{B} is true of s_{tt} and this is reflected by the EDB fact of $G(s_{tt})$. The introduction of s_{tt} and the G predicate forces each state to have exactly one outgoing transition for every action, making the Datalog rule for the $[a]$ operator the same as that for $\langle a \rangle$.

The following fixed Datalog program, which encodes the semantics of the equation block, can be generated in constant time. The intuitive meaning of the IDB predicate $T(s, X_k)$ is that X_k is true of state s . $ATOM(s, A)$, the predicate for truth values of atomic propositions with respect to every state, is given as part of the input to the model checking problem.

$$\begin{aligned}
T(s, X_k) &\leftarrow F_{AP}(X_k, A), ATOM(s, A) \\
T(s, X_k) &\leftarrow F_{\vee}(X_k, X_i), T(s, X_i) \\
T(s, X_k) &\leftarrow F_{\langle \rangle}(X_k, a, X_i), T(s', X_i), L(s, a, s') \\
T(s, X_k) &\leftarrow F_{[\]}(X_k, a, X_i), T(s', X_i), L(s, a, s') \\
T(s, X_k) &\leftarrow G(s)
\end{aligned} \tag{3}$$

It is shown in [UvG88, Ull92] showed that any fixed Datalog problem with the so called *polynomial fringe property* is in NC, meaning that it can be solved in polylog time with a polynomially many number of processors (both in the size of the EDB). It is further shown that a Datalog program in which every rule has at most one IDB predicate on the right hand side always has the polynomial fringe property. Our Datalog program is obviously in NC.

The above reduction of MCMC to Datalog (with an EDB size of $O(|\mathcal{B}| + |S|)$) and the existence of an NC algorithm for Datalog problem combine to give an NC algorithm for MCMC. ■

An alternative NC algorithm for the same problem can be derived via a reduction to Proplog. We assign one processor to each of the $|\mathcal{B}| * |S|$ equation–state pairs. In the following table, the intuitive meaning of a proposition sX_i is that variable X_i is true of state s .

equation type	Proplog rules generated
$X_k = A$	sX_k if $ATOM(s, A)$ is true
$X_k = X_i \vee X_j$	$sX_k \rightarrow sX_i$ and $sX_k \rightarrow sX_j$
$X_k = \langle a \rangle X_i$	$sX_k \rightarrow s'X_i$ if s has a -derivative s'
$X_k = [a]X_i$	$sX_k \rightarrow s'X_i$ if s has a -derivative s' sX_k if s has no a -derivative

The above reduction can be done in constant parallel time and the size of the resulting Proplog program is $O(|\mathcal{B}| * |S|)$. It is important to note that each rule in the program can have at most one subgoal.

Our NC algorithm is adapted from the one presented in [Ull92] and [UvG88]. [Ull92] and [UvG88] deal with a fixed Datalog program with a variable size EDB, whereas we have a Proplog program of variable size but no database. The idea remains the same: processors work simultaneously on all possible proof trees. The existence of an NC algorithm depends on the fact that every true proposition has a proof tree whose size is bounded by a polynomial in the

size of the program. This is obviously true for our Proplog program, where each rule has at most one subgoal: any proof tree for a proposition is a chain and no proposition can appear more than once along the chain in a minimum size proof tree.

5 Conclusions and Open Problem

In this paper, we have analyzed the parallel complexity of the problem of model checking in the modal mu-calculus. We have shown that the problem is P-complete even for a very restricted case involving the alternation-free fragment. Further restrictions have led to two efficient NC algorithms.

In the course of deriving one of our NC algorithms, we have given a parallel constant-time reduction from the alternation-free modal mu-calculus to Datalog. A polynomial-time reduction in the other direction is also possible. The basic idea is to construct an LTS whose states correspond to ground instances of the predicates of the Datalog program, and whose transitions reflect the dependencies among predicates in ground instances of the rules of the program. The mu-calculus formula to be checked states, intuitively, that a fact can be derived because it is true initially or because all facts appearing in the right-hand side of the defining rule are true. These reductions establishes some interesting links between the mu-calculus and Datalog that up till now have gone unnoticed. Moreover, because of the results of [Ul92], the reduction from Datalog to the mu-calculus provides the basis for another proof of the P-completeness of model checking in the modal mu-calculus.

In closing, we note that we have not found any obvious way to encode in CTL the property expressed by (1), the alternation-free formula used in our P-hardness reduction from the circuit value problem MACVP to MMMC. This leaves open the parallel complexity of model checking in CTL.

References

- [AP93] F. Afrati and C.H. Papadimitriou. The parallel complexity of simple logical programs. *JACM*, 40(4):891–916, September 1993.
- [BC93] G. S. Bhat and R. Cleaveland. Polynomial-time model checking in the modal mu-calculus. Submitted to LICS '94, December 1993.
- [BGS92] J. L. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is p-complete. *Formal Aspects of Computing*, 4(6A):638–648, 1992.
- [CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2), 1986.
- [CPS93] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM TOPLAS*, 15(1), 1993.
- [CS91] Rance Cleaveland and Bernhard Steffen. Computing behavioural relations, logically. In *ICALP '91*, pages 127–138. LNCS 510, 1991.

- [CS93] Rance Cleaveland and Bernhard Steffen. A linear-time model checking algorithm for the alternation-free modal μ -calculus. *Formal Methods in System Design*, 2, 1993.
- [EL86] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ -calculus. In *Proceedings of the First Annual Symposium on Logic in Computer Science*, pages 267–278, 1986.
- [FL79] M. J. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.
- [GHR91] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. A compendium of problems complete for p. Technical Report TR-91-05-01, Department of Computer Science and Engineering, University of Washington, 1991.
- [HM85] M. C. B. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.
- [Kan85] P. C. Kanellakis. Logic programming and parallel complexity. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufman, 1985.
- [KC90] S. Kimura and E. M. Clarke. A parallel algorithm for constructing binary decision diagrams. In *Proc. IEEE International Conference on Computer Design*, pages 220–223, 1990.
- [Koz83] D. Kozen. Results on the propositional μ -calculus. *Theoretical Computer Science*, 27:333–354, 1983.
- [KS90] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, May 1990.
- [LR93] Insup Lee and Sanguthevar Rajasekaran. Fast parallel deterministic and randomized algorithms for model checking. Technical Report MS-CIS-93-69, University of Pennsylvania, 1993.
- [RdS90] Valerie Roy and Robert de Simone. Auto/autograph. In *CAV '90*. LNCS 531, 1990.
- [RMMM92] M. Reid-Miller, G. L. Miller, and F. Modugno. List ranking and parallel tree contraction. In John Reif, editor, *Synthesis of Parallel Algorithms*, pages 115–194. Morgan Kaufman, 1992.
- [RRSV87] J. Richier, C. Rodriguez, J. Sifakis, and J. Voiron. Verification in XESAR of the sliding window protocol. In *Proceedings of the Seventh IFIP Symposium on Protocol Specification, Testing, and Verification*. North-Holland, 1987.
- [Ste89] Bernhard Steffen. Characteristic formulae. In *ICALP '89*. LNCS 372, 1989.

- [Ull92] J. D. Ullman. Parallel complexity of logical inference. In John Reif, editor, *Synthesis of Parallel Algorithms*, pages 759–779. Morgan Kaufman, 1992.
- [UvG88] J. D. Ullman and A. van Gelder. Parallel complexity of logical query programs. *Algorithmica*, 3:5–52, 1988.
- [ZS92] S. Zhang and S. A. Smolka. Towards efficient parallelization of equivalence checking algorithms. In M. Diaz and R. Groz, editors, *Proceedings of FORTE '92 – Fifth International Conference on Formal Description Techniques*, pages 133–146, October 1992.