Efficient Model Checking via the Equational μ-Calculus

Girish Bhat *
Rance Cleaveland*
Department of Computer Science,
North Carolina State University,
Raleigh, NC 27695-8206, USA.
E-mail: {gsbhat1@eos,rance@csc}.ncsu.edu.
Tel: (919) 515-7862.

Abstract

This paper studies the use of an equational variant of the modal μ-calculus as a unified framework for efficient temporal logic model checking. In particular, we show how an expressive temporal logic, CTL*, may be efficiently translated into the μ-calculus. Using this translation, one may then employ μ-calculus model-checking techniques, including on-the-fly procedures, BDD-based algorithms and compositional model-checking approaches, to determine if systems satisfy formulas in CTL*.

1. Introduction

Recent years have witnessed a surge of work on automatic approaches for establishing whether or not finite-state reactive systems satisfy specifications given in temporal logics [5, 8, 9, 14, 18, 21, 24, 26, 29]. Reactive systems typically maintain an ongoing interaction with their environments; temporal logics permit the formulation of requirements on a system's evolving behavior and hence are well-suited for specifying system properties. The task of verifying if a system satisfies a temporal formula is referred to as *model checking*.

One particularly expressive logic is the *modal μ-calculus* [20], which, in addition to its syntactic minimality, is capable of encoding a variety of linear- and branching-time logics [17]. These facts have led Emerson and others to view μ-calculus model checking as a unifying framework for temporal-logic model checking; in this framework the μ-calculus serves as an "intermediate language" into which other logics may be translated for model-checking purposes. In addition to its theoretical elegance, this program has had practical implications for pure branching-time logics such as PDL-δ and CTL: the complexity of model checking via μ-calculus matches the complexity of the best existing model checkers for these logics [8, 17], and the translation routines permit the application of μ-calculus-based heuristics, such as BDD-based approaches [7], partial model checking [4], and on-the-fly algorithms [3], for coping with state-explosion. However, for logics such as LTL, CTL* and ECTL*, which are capable of expressing linear-time properties, the practical effects of this framework have been limited by the absence of efficient translations for these logics. In particular, existing approaches [15, 17] yield formulas which are double-exponential in input size for CTL* formulas and single exponential for ECTL* formulas, and these complexities substantially exceed those for model-checking algorithms that have been developed for these logics.

In this paper we elaborate on the μ-calculus program by showing how an *equational* variant of the μ-calculus [3, 4, 11, 13] may be used as a uniform basis for efficient model checking in linear- as well as branching-time logics. More specifically, we show how formulas in CTL* may be translated into the equational μ-calculus in time that is exponential in the size of the input formula and then efficiently checked using existing μ-calculus algorithms. When applied to different sublogics of CTL*, our procedure matches the best existing translations for these logics in both efficiency and size of output. We also give a translation for ECTL* that has linear complexity. As a result, our approach provides an efficient unifying framework for temporal-logic model checking, and it permits the extension of space-efficient techniques developed for μ-calculus to these logics as well.

The organization of the paper is as follows. In Section 2 we present the syntax and semantics of the equational μ-calculus and CTL*. The following section introduces spe-

*Research supported by NSF/DARPA grant CCR-9014775, NSF grant CCR-9120995, ONR Young Investigator Award N00014-92-J-1582, NSF Young Investigator Award CCR-9257963, NSF grant CCR-9402807, and AFOSR grant F49620-95-1-0508.

cial tree automata and shows how they may be translated into the μ-calculus, and the section after then shows how CTL* may be efficiently translated into these automata. Section 5 briefly discusses the efficiency of model-checking in our μ-calculus, and the final section contains our conclusions and directions for future research.

2. The Equational μ-calculus and CTL*

This section introduces Kripke structures, which serve as models for the temporal logics we consider in this paper. We then present the syntax and semantics of the equational μ-calculus and CTL*. In the remainder of the paper we fix a set $(a, b \in)\mathcal{A}$ of *atomic propositions*.

2.1. Kripke Structures

Kripke structures are defined as follows.

Definition 2.1 *A Kripke structure is a triple $\langle S, R, L\rangle$ where S is the set of states, $R \subseteq Q \times Q$ is the transition relation, and $L : S \to 2^{\mathcal{A}}$ is the labeling.*

Intuitively, a Kripke structure encodes the operational behavior of a system, with S representing the possible system states, R describing the "execution steps", and L indicating which atomic propositions hold in a given state. For technical convenience, throughout this paper we assume that transition relations R are *total*: for every $s \in S$ there is an $s' \in S$ such that $\langle s, s'\rangle \in R$. On occasion we write $\langle S, R, L, s_0\rangle$ to represent a Kripke structure with "start state" $s_0 \in S$. We also use the following notions.

Definition 2.2 *Let $\mathcal{T} = \langle S, R, L\rangle$ be a Kripke structure.*

1. *A* path *from s_0 in $\mathcal{T}$ is a maximal sequence $\langle s_0, s_1, \ldots\rangle$ such that for all $i \geq 0$, $\langle s_i, s_{i+1}\rangle \in R$.*
2. *If $x = \langle s_0, s_1, \ldots\rangle$ is a path in $\mathcal{T}$ then $x(i) = s_i$ and $x^i = \langle s_i, s_{i+1}, \ldots\rangle$.*

2.2. μ-Calculus Syntax

In addition to the set $\mathcal{A}$, the syntax of the equational μ-calculus is parameterized with respect to a set $(X, Y) \in \mathcal{V}$ of propositional variables. The set of *basic formulas* may now be described by the following grammar.

$$\psi ::= a \mid \neg a \mid X \mid X \vee Y \mid X \wedge Y \mid \langle\cdot\rangle X \mid [\cdot]X$$

The operators $\neg$, $\vee$ and $\wedge$ have the usual interpretation, while $\langle\cdot\rangle$ and $[\cdot]$ are (single-step) modalities. Note that negation may only be applied to atomic propositions and that the proper subformulas a basic formula are variables.

An *equational system* E consists of a finite set of equations $\{X_i = \psi_i\}$ where the X_i are distinct and each ψ_i is a basic formula. If E is an equational system and $X \in \mathcal{V}$, then we use $\mathsf{lhs}(E)$ to represent the left-hand sides of equations in E and $\mathsf{rhs}(E, X)$ to denote the right-hand side of X in E if $X \in \mathsf{lhs}(E)$. A *block* B has form νE or μE, where E is an equational system; we extend the functions lhs and rhs to blocks in the obvious manner, and we call block B a ν-block if it has the form νE and a μ-block otherwise. Intuitively, ν-blocks denote the largest "solution" to the given equations, while μ-blocks represent least solutions.

Finally, formulas in the equational μ-calculus have form $X\ in\ \mathcal{B}$, where $\mathcal{B} = \langle B_1, \ldots, B_n\rangle$ is a *list* of equational blocks satisfying the following: if $i \neq j$ then $\mathsf{lhs}(B_i) \cap \mathsf{lhs}(B_j) = \emptyset$. Define $\mathsf{lhs}(\mathcal{B}) = \bigcup_{i=1}^{n} \mathsf{lhs}(B_i)$ and extend the definition of rhs in the obvious way. If $Y \in \mathsf{lhs}(\mathcal{B})$ then we say that Y is *bound* in $\mathcal{B}$; a basic formula ψ is *closed with respect to* $\mathcal{B}$ if its proper subformulas (which are variables, recall) are bound in $\mathcal{B}$. We say that a formula $X\ in\ \mathcal{B}$ is closed if X is bound in $\mathcal{B}$ and $\mathsf{rhs}(\mathcal{B}, Y)$ is closed in $\mathcal{B}$ for each $Y \in \mathsf{lhs}(\mathcal{B})$. Finally, if we have formulas $\Phi = X\ in\ \mathcal{B}$ and $\Phi' \equiv X'\ in\ \mathcal{B}'$ then we abuse notation and write $\Phi \vee \Phi'$, $\Phi \wedge \Phi'$, etc. to stand for the equational formulas obtained in the obvious manner.

2.3. μ-Calculus Semantics

We interpret equational μ-calculus formulas with respect to Kripke structures and environments that assign meaning to propositional variables. If $\mathcal{T} = \langle S, R, L\rangle$ is a Kripke structure then we write $\mathsf{env}(\mathcal{T})$ for the set of functions $\mathcal{V} \to 2^S$; intuitively, if $e \in \mathsf{env}(\mathcal{T})$ then $e(X)$ records which states in $\mathcal{T}$ are assumed to "satisfy" X. We also use the following operation on environments. Let $e, e' \in \mathsf{env}(\mathcal{T})$, with $\mathcal{V}' \subseteq \mathcal{V}$. Then $e[e'/\mathcal{V}'] \in \mathsf{env}(\mathcal{T})$ is defined as follows.

$$e[e'/\mathcal{V}'](X) = \begin{cases} e'(X) & \text{if } X \in \mathcal{V}' \\ e(X) & \text{otherwise} \end{cases}$$

To define the semantics of formulas we first give the meaning of basic formulas and then of block lists. The semantics of the basic formulas are given in Fig 1; the semantic function, denoted by $[\![\cdot]\!]_{\mathcal{T}}e$, maps from basic formulas to sets of states satisfying the given formula.

We now give an inductive definition of the meaning, $[\![\mathcal{B}]\!]_{\mathcal{T}}e$, of a block list $\mathcal{B}$ as an environment in $\mathsf{env}(\mathcal{T})$. When $\mathcal{B} = \langle\rangle$, we take $[\![\mathcal{B}]\!]_{\mathcal{T}}e = e$. Now assume that $\mathcal{B} = \langle B_1, \ldots, B_n\rangle$ is nonempty. Our intention is to define a function based on $\mathcal{B}$ and e and then interpret $[\![\mathcal{B}]\!]_{\mathcal{T}}e$ as an appropriate fixpoint of this function. To this end, let $f_{\mathcal{B},e} \in \mathsf{env}(\mathcal{T}) \to \mathsf{env}(\mathcal{T})$ be defined as follows.

$$
\begin{array}{rcl}
[\![A]\!]_{\mathcal{T}} e & = & L(A) \\
[\![X]\!]_{\mathcal{T}} e & = & e(X) \\
[\![\neg\phi]\!]_{\mathcal{T}} e & = & S - [\![\phi]\!]_{\mathcal{T}} e \\
[\![X_1 \vee X_2]\!]_{\mathcal{T}} e & = & e(X_1) \cup e(X_2) \\
[\![X_1 \wedge X_2]\!]_{\mathcal{T}} e & = & e(X_1) \cap e(X_2) \\
[\![\langle\cdot\rangle X]\!]_{\mathcal{T}} e & = & \{\, s \mid \exists s'.\, \langle s, s'\rangle \in R \wedge s' \in e(X) \,\} \\
[\![[\cdot] X]\!]_{\mathcal{T}} e & = & \{\, s \mid \forall s'.\, \langle s, s'\rangle \in R \Rightarrow s' \in e(X) \,\}
\end{array}
$$

Figure 1. Semantics of basic formulas for $\mathcal{T} = \langle \mathcal{S}, R, L\rangle$, where $e \in \mathsf{env}(\mathcal{T})$.

$$
f_{\mathcal{B},e}(e')(X) = \begin{cases} [\![\mathsf{rhs}(B_1, X)]\!]_{\mathcal{T}}([\![\langle B_2, \ldots, B_n\rangle]\!]_{\mathcal{T}} \\ \quad e[e'/\mathsf{lhs}(\mathcal{B})])\ if\ X \in \mathsf{lhs}(B_1) \\ ([\![\langle B_2, \ldots, B_n\rangle]\!]_{\mathcal{T}} e[e'/\mathsf{lhs}(\mathcal{B})])(X) \\ \quad otherwise \end{cases}
$$

To understand this function, first note that $[\![\langle B_2, \ldots, B_n\rangle]\!]_{\mathcal{T}}$ $e[e'/\mathsf{lhs}(\mathcal{B})]$ represents the meaning of $\langle B_2, \ldots, B_n\rangle$ in an environment in which the left-hand sides of B_1 are interpreted according to e' and other free variables are interpreted according to e. Then $f_{\mathcal{B},e}(e')$ yields an environment interpreting variables $X \in \mathsf{lhs}(B_1)$ as the meanings of $\mathsf{rhs}(B_1, X)$ with respect to this recursively calculated environment and other variables using this environment alone. Now suppose that B_1 is a ν-block. Then $[\![\mathcal{B}]\!]_{\mathcal{T}} e = \nu f_{\mathcal{B},e}$, where $\nu f_{\mathcal{B},e}$ is the greatest fixpoint of $f_{\mathcal{B},e}$ interpreted over the lattice obtained by ordering $\mathsf{env}(\mathcal{T})$ using pointwise set inclusion. Dually, if B_1 is a μ-block then $[\![\mathcal{B}]\!]_{\mathcal{T}} e = \mu f_{\mathcal{B},e}$, where μ is the least fixpoint operator. The existence of these fixpoints follows from the Tarski-Knaster theorem [27].

A formula $X\ in\ \mathcal{B}$ may now be interpreted as follows: $[\![X\ in\ \mathcal{B}]\!]_{\mathcal{T}} e = ([\![\mathcal{B}]\!]_{\mathcal{T}} e)(X)$. If a formula is closed then its meaning is independent of e; in this case we write $[\![X\ in\ \mathcal{B}]\!]_{\mathcal{T}}$.

Finally, it should be noted that although this logic contains no negation, it is possible to "generate" negated formulas by exploiting the duality of $\vee/\wedge$, $\langle\cdot\rangle/[\cdot]$, and μ/ν.

Lemma 2.3 *Let $\mathcal{T} = \langle S, R, L\rangle$ be a Kripke structure and Φ a closed equational μ-calculus formula. Then there is a closed μ-calculus formula $\mathsf{neg}(\Phi)$ such that $[\![\mathsf{neg}(\Phi)]\!]_{\mathcal{T}} = S - [\![\Phi]\!]_{\mathcal{T}}$.*

2.4. CTL*

The following BNF-like grammar describes the syntax of CTL*.

$$
\begin{array}{rcl}
\mathcal{S} & ::= & a \mid \neg a \mid \mathcal{S} \wedge \mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid A\mathcal{P} \mid E\mathcal{P} \\
\mathcal{P} & ::= & \mathcal{S} \mid \mathcal{P} \wedge \mathcal{P} \mid \mathcal{P} \vee \mathcal{P} \mid \mathsf{X}\mathcal{P} \mid \mathcal{P}\mathsf{U}\mathcal{P} \mid \mathcal{P}\mathsf{V}\mathcal{P}
\end{array}
$$

We sometimes call formulas of the form a or $\neg a$ *literals*; we use $(l \in)\mathcal{L}$ to represent the set of all literals. We refer to the formulas generated from $\mathcal{S}$ as *state* formulas and those from $\mathcal{P}$ as *path* formulas; we define the CTL* formulas to be the set of state formulas. We use $p, p_1, q, \ldots$ to range over the set of state formulas and $\phi, \phi_1, \gamma, \ldots$ to range over the set of path formulas. We also call A and E *path quantifiers* and the X, U and V constructs *path modalities*. The sublogic CTL [8] consists of those CTL* formulas in which every occurrence of a path modality is immediately preceded by a path quantifier, while the sublogic LTL [21] contains CTL* formulas of the form $E\phi$, where the only state subformulas of ϕ are literals.

Let $\mathcal{T} = \langle S, R, L\rangle$ be a Kripke structure. Then the meaning of CTL* formulas is given in terms of a relation $\models_{\mathcal{T}}$ relating states to state formulas and paths to path formulas. Intuitively, $s \models_{\mathcal{T}} p$ ($x \models_{\mathcal{T}} \phi$) if state s (path x) satisfies state formula p (path formula ϕ). The meaning of most of the constructs is straightforward. A state satisfies $A\phi$ ($E\phi$) if every path (some path) emanating from the state satisfies ϕ, while a path satisfies a state formula if the initial state in the path does. X represents a "next-time operator", while $\phi_1 \mathsf{U} \phi_2$ holds of a path if ϕ_1 remains true until ϕ_2 becomes true. The constructor V may be thought of as a "release operator"; a path satisfies $\phi_1 \mathsf{V} \phi_2$ if ϕ_2 remains true until ϕ_1 "releases" the path from the obligation. The formal definition of $\models_{\mathcal{T}}$ is standard and appears in the appendix.

Finally, although we only allow a restricted form of negation in this logic ($\neg$ may only be applied to atomic propositions), we do have the following result.

Lemma 2.4 *Let $\mathcal{T} = \langle S, R, L\rangle$ be a Kripke structure.*

1. *For any state formula p there is a state formula $\mathsf{neg}(p)$ such that for all $s \in S$, $s \models_{\mathcal{T}} \mathsf{neg}(p)$ iff $s \not\models_{\mathcal{T}} p$.*
2. *For any path formula ϕ there is a path formula $\mathsf{neg}(\phi)$ such that for all paths x in $\mathcal{T}$, $x \models_{\mathcal{T}} \mathsf{neg}(\phi)$ iff $x \not\models_{\mathcal{T}} \phi$.*

2.5. ECTL*

Our definition of the syntax and semantics of ECTL* closely follows the definitions given in [28]. Before we present the syntax we give the following definition.

Definition 2.5 *A Büchi Automaton $\mathcal{B}$ has form $\langle Q, \Sigma, \rightarrow, q_0, F\rangle$ where Q is the set of states, Σ is the alphabet, $\rightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation, q_0 is the start state, and $F \subseteq Q$ is the set of final states.*

A run of $\mathcal{B}$ on an ω-word $\alpha : \omega \rightarrow \Sigma$ is an ω-word $r : \omega \rightarrow Q$ such that $r(0) = q_0$ and for every i, $r(i) \stackrel{\alpha(i)}{\rightarrow} r(i+1)$. A run r of $\mathcal{B}$ is successful iff a state $q \in F$ occurs infinitely often on the run. $\mathcal{B}$ accepts an ω-word α iff it has a successful run on α.

The syntax of ECTL* is given as follows.

$$\Phi ::= a|\neg\Phi|\Phi \wedge \Phi|E(\mathcal{B})$$

where $\mathcal{B}$ is a Büchi automaton with alphabet $2^{\{\Phi_1,..,\Phi_n\}}$.

ELTL is the sublogic containing only formulas of form $E(\mathcal{B})$ where the $\mathcal{B}$ is defined over the alphabet $2^{\mathcal{A}}$.

The formal semantics of ECTL*is presented in the appendix.

3. Automata and the μ-Calculus

Our conversion of CTL* into the equational μ-calculus uses a translation of CTL* into intermediate structures that we refer to as *Büchi tableau automata*. We now describe these automata and present a translation of them into the equational μ-calculus.

3.1. Büchi Tableau Automata

The connection between ω-tree automata and temporal logic is well-established and has been fruitfully employed to yield results on time and space bounds for the problems of synthesis and model checking [5, 23, 29]. However, the translations of temporal logic formulas into tree automata have typically been complicated by the strict operational behavior of such automata (each automaton transition must consume an input symbol) and by the lack of logical structure they exhibit. The latter in particular introduces subtleties into the automata-theoretic interpretation of logical connectives. In this paper we use a form of tree automata to serve as an "intermediate" language into which we translate CTL* and ECTL* [28], and from which we generate μ-calculus formulas. These automata, which we call *Büchi tableau automata*, contain more logical structure than traditional tree automata and hence are easier to render in a logical formalism such as the μ-calculus. They are similar in many respects to the amorphous and alternating tree automata of [23] and [5] respectively.

Definition 3.1 *A Büchi tableau automaton (BTA) is a tuple $\langle Q, \rightarrow, q_0, F, \ell\rangle$, where Q is the set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accepting states $\ell : Q \rightarrow \mathcal{L} \cup \{\wedge, \vee, [\cdot], \langle\cdot\rangle\}$ is a state labeling, and $\rightarrow \subseteq (Q \times Q)$ satisfies the following for all $q \in Q$.*

- $|\{\, q' \mid q \rightarrow q' \,\}| = 0$ *if* $\ell(q) \in \mathcal{L}$.
- $|\{\, q' \mid q \rightarrow q' \,\}| \geq 1$ *if* $\ell(q) \in \{\wedge, \vee\}$.
- $|\{\, q' \mid q \rightarrow q' \,\}| = 1$ *if* $\ell(q) \in \{[\cdot], \langle\cdot\rangle\}$.

Intuitively, a BTA represents a property of trees, with the labels on the node indicating what "deduction" steps must be taken to "prove" that an input tree satisfies the property and the acceptance condition indicating when a proof (which may be infinite) is "valid". We formalize these notions in the context of defining when a BTA accepts a Kripke structure (which may be unfolded into an infinite tree in the obvious manner).

Definition 3.2 *A run of BTA $\langle Q, \rightarrow, q_0, F, \ell\rangle$ on a Kripke structure $\langle S, R, L, s_0\rangle$ is a maximal tree with nodes labelled by $Q \times S$ satisfying the conditions below.*

- *the root of the tree is labeled with $\langle q_0, s_0\rangle$.*
- *for a node σ labeled $\langle q, s\rangle$:*
  - *If $\ell(q) \in \mathcal{L}$ then σ is a leaf (i.e. has no successors).*
  - *if $\ell(q) = \vee$ then σ has one successor, σ', and σ' is labeled with $\langle q', s\rangle$ for some $q' \in \{\, q' \mid q \rightarrow q' \,\}$.*
  - *if $\ell(q) = \wedge$ and $\{\, q' \mid q \rightarrow q' \,\} = \{q_1, ..q_m\}$, then σ has successors $\sigma_1, .., \sigma_m$, with σ_i labeled by $\langle q_i, s\rangle$.*
  - *if $\ell(q) = \langle\cdot\rangle$ and $q \rightarrow q'$ then σ has one successor σ', and σ' is labeled by $\langle q', s'\rangle$ for some $s' \in \{\, s' \mid \langle s, s'\rangle \in R\,\}$.*
  - *if $\ell(q) = [\cdot]$, q' is such that $q \rightarrow q'$, and $\{\, s' \mid \langle s, s'\rangle \in R\} = \{s_1, .., s_m\}$ then σ has successors $\sigma_1, .., \sigma_m$, with σ_i is labelled by $\langle q', s_i\rangle$.*

An infinite path $\langle s_0, q_0\rangle\langle s_1, q_1\rangle \ldots$ satisfies the Büchi condition *iff there is an $q' \in F$ that appears infinitely often on the path. A run is* successful *iff: for every leaf $\langle q, s\rangle$ in the run, $s \models \ell(q)$ (recall $\ell(q) \in \mathcal{L}$); and every infinite path in the run satisfies the Büchi condition.*

A BTA $\mathcal{B}$ accepts a Kripke structure $\mathcal{T}$ iff there exists a successful run of B on M.

Following [25] we also introduce *generalized* BTAs, which differ from BTAs only in their acceptance condition: instead of a set of states F, a generalized BTA uses a set $\mathcal{F}$ of sets of states. An infinite path x in a run of a generalized BTA satisfies the acceptance condition $\mathcal{T}$ iff for each $F \in \mathcal{F}$ there is some label $\langle q, s\rangle$ with $q \in F$ that appears infinitely often on x. A generalized BTA $\mathcal{B} \equiv \langle Q, \mathcal{A}, \rightarrow, q_0, \mathcal{F} \equiv \{F_0, .., F_m\}, l\rangle$ can be translated to an ordinary BTA $\mathcal{B}' \equiv \langle Q', \mathcal{A}, \rightarrow_1, q_0', F, l'\rangle$ in a straightforward fashion by introducing an integer counter whose value is bounded by $|\mathcal{F}|$. Formally, $Q' = Q \times \{0, 1, ..., |\mathcal{F}| - 1\}$.

The start state $q'_0 = \langle q_0, 0\rangle$. For a state $\langle q, i\rangle$ the labelling of the state $l'(\langle q, i\rangle) = l(q)$. The transitions of state $\langle q, i\rangle$ are defined in the following manner. If $q \in F_i$, then corresponding to each transition $q \rightarrow q'$, we introduce the transition $\langle q, i\rangle \rightarrow_1 \langle q', (i+1)\ mod\ |\mathcal{F}|\rangle$. If $q \notin F_i$, then for each transition $q \rightarrow q'$, we introduce the transition $\langle q, i\rangle \rightarrow_1 \langle q', i\rangle$. The set of accepting states of B', $F = \{\, \langle q, 0\rangle \mid q \in F_0 \,\}$. Note that $|\mathcal{B}'| = |\mathcal{B}| * |\mathcal{F}|$.

3.2. Translating BTA into the Equational μ-Calculus

We now show how to translate a BTA into a formula in the equational μ-calculus that is satisfied by exactly the Kripke structures accepted by the BTA and that is the same size as the BTA. The idea behind the translation is straightforward; we introduce a distinct μ-calculus variable for each state in the BTA and assign it a right-hand side based on the state's label. Equations for accepting states are then grouped into a ν-block B_1, while those for non-accepting states become a μ-block B_2. The final formula produced then has the form $X_0\ in\ \langle B_1, B_2\rangle$, where X_0 is the variable for the BTA's the start state. The formal details are as follows. Let $B = \langle Q, \rightarrow, q_0, F, \ell\rangle$ be a BTA, and assume $Q = \{q_0, \ldots, q_n\}$. We define a set of equations as follows.

1. If $\ell(q_i) \in \mathcal{L}$ then generate equation $X_i = \ell(q_i)$.
2. If $\ell(q_i) = \langle\cdot\rangle$ then generate equation $X_i = \langle\cdot\rangle X_j$, where j is such that $q_i \rightarrow q_j$, and similarly for $\ell(q_i) = [\cdot]$.
3. If $\ell(q_i) = \vee$ then we perform a case analysis on $\{q_{i_1}, \ldots q_{i_m}\} = \{\, q' \mid q_i \rightarrow q' \,\}$.
    - If $m = 1$ then generate equation $X_i = X_{i_1}$.
    - If $m = 2$ then generate equation $X_i = X_{i_1} \vee X_{i_2}$.
    - Otherwise ($m > 2$), produce new variables $X_{i,j}, 2 \le j < m$ and generate the following equations.

$$\begin{array}{rcl} X_i & = & X_{i_1} \vee X_{i,2} \\ X_{i,2} & = & X_{i_2} \vee X_{i,3} \\ & \vdots & \\ X_{i,m-1} & = & X_{i_{m-1}} \vee X_{i_m} \end{array}$$

A similar procedure is used to handle $\wedge$-states.

Note that the only subtlety in the translation arises from the fact that $\vee$- and $\wedge$-states in BTAs can have an arbitrary number of derivatives while the $\vee$ and $\wedge$ operators in the μ-calculus are binary.

We now group the equations into ν-block B_1 containing the equations generated as a result of processing states in F and μ-block B_2 holding equations resulting from states in $Q - F$. The μ-calculus formula is now $X_0\ in\ \langle B_1, B_2\rangle$. We have the following.

Theorem 3.3 *Let $\phi_{\mathcal{B}}$ be the μ-calculus formula generated for BTA $\mathcal{B}$ by the above procedure. Then for every Kripke structure $\mathcal{T} \equiv \langle S, R, L, s_0\rangle$, $s_0 \in [\![\phi_{\mathcal{B}}]\!]_{\mathcal{T}} e$ iff $\mathcal{T}$ is accepted by $\mathcal{B}$.*

The procedure is clearly linear in both time and space.

4. CTL* and the Equational μ-calculus

We now turn to the main goal of this paper, which is to give an efficient translation of CTL* formulas into the equational μ-calculus. Our approach involves giving a translation of the sublogic LTL formulas into BTAs that we then "lift" into one for full CTL*. Once we have a BTA for a formula, we can use the procedure outlined in the previous section to generate the desired μ-calculus formula. Before proceeding, we introduce the following notation. We use $E\Phi$, where Φ is a set of path formulas, to denote LTL formula $E(\bigwedge_{\phi\in\Phi}\phi)$, and we sometimes write $E(\Phi, \phi_1, \ldots, \phi_n)$ to represent a formula of the form $E(\Phi \cup \{\phi_1, \ldots, \phi_n\})$. For a formula $p \equiv E\Phi$, we use $\phi \in p$ to denote $\phi \in \Phi$.

To build a BTA $\langle Q, \mathcal{A}, \rightarrow, q_0, \mathcal{F}, l\rangle$ from LTL formula $p \equiv E\Phi$, we first construct the *tableau* for p by exhaustively applying the rules in Figure 2 to p in a manner formalized below. (Similar rules are used in [15].)

Definition 4.1 *Let V be a set of LTL formulas of form $E\Phi'$ and $E \subseteq V \times V$. Then $\langle V, E\rangle$ is a* tableau *for p if it is a maximal graph such that for every $p' \in V$: p' is reachable from p using edges in E, and the set $\{p'' | \langle p', p''\rangle \in E\}$ is the result of applying some rule to p'.*

Let $r(q)$ denote the rule applied at node q. Then a (generalized) BTA B_p for p can be extracted from the tableau for p as follows. The state set of the automaton is V, with the start state taken as p. The transition relation is E. The labeling function ℓ is defined as follows. If q contains no successors (in which case it must be of the form $E(l)$ for some $l \in \mathcal{L}$) then $\ell(q) = l$. Otherwise, the successors of q are given by $rule(q)$, in which case $\ell(q)$ becomes the label of the rule ($\vee$, $\wedge$ or $\langle\cdot\rangle$ in this case). To define the acceptance set $\mathcal{F}$ we do the following. Suppose $\phi \equiv \phi_1 \mathsf{U} \phi_2 \in q$ and let $F_\phi = \{\, q' \in Q \mid (\phi \notin q'\ and\ X\phi \notin q')\ or\ \phi_2 \in q' \,\}$. Then $\mathcal{F} = \{\, F_\phi \mid \phi \equiv \phi_1 \mathsf{U} \phi_2\ and\ \exists q \in V.\phi \in q \,\}$. We now have the following.

Theorem 4.2 *Let $p = E\Phi$ be an LTL formula with $\mathcal{B}_p$ the associated BTA. Also let $\mathcal{T} = \langle S, R, L, s_0\rangle$ be a Kripke structure. Then $s_0 \models_{\mathcal{T}} p$ iff $\langle S, R, L, s_0\rangle$ is accepted by $\mathcal{B}_p$.*

$$\wedge : \frac{E(\Phi, l)}{E(\Phi) \quad E(l)} \qquad \langle \cdot \rangle : \frac{E(\mathsf{X}\phi_1, \ldots, \mathsf{X}\phi_n)}{E(\phi_1, \ldots, \phi_n)}$$

$$\vee : \frac{E(\Phi, \phi_1 \wedge \phi_2)}{E(\Phi, \phi_1, \phi_2)} \qquad \vee : \frac{E(\Phi, \phi_1 \vee \phi_2)}{E(\Phi, \phi_1) \quad E(\Phi, \phi_2)}$$

$$\vee : \frac{E(\Phi, \phi_1 \mathsf{V} \phi_2)}{E(\Phi, \phi_1, \phi_2) \quad E(\Phi, \phi_2, \mathsf{X}(\phi_1 \mathsf{V} \phi_2))} \qquad \vee : \frac{E(\Phi, \phi_1 \mathsf{U} \phi_2)}{E(\Phi, \phi_2) \quad E(\Phi, \phi_1, \mathsf{X}(\phi_1 \mathsf{U} \phi_2))}$$

Figure 2. Tableau rules for LTL.

Proof. A brief outline of the proof is as follows. First, define an infinite path $p_0, p_1, ...$ in the run of $\mathcal{B}_p$ to be *successful* iff for every formula $\phi \equiv \phi_1 \mathsf{U} \phi_2 \in p_i$ there exists $j \geq i$ such that $\phi_2 \in p_j$. The correctness of the theorem follows from the following lemmas. □

Lemma 4.3 *$s \models_{\mathcal{T}} p$ iff there exists a run r of $\mathcal{B}_p$ on $\langle S, R, L, s \rangle$ such that: all leaves of r have form $\langle l, s' \rangle$ where $s' \models_{\mathcal{T}} l$, and all the infinite paths in the run are successful.*

Lemma 4.4 *An infinite path in a run r of $\mathcal{B}_p$ is successful iff it satisfies the Büchi condition.*

Let the above translation procedure for LTL be referred to as $transLTL$. Then the pseudocode for the CTL* translation procedure is given in Figure 3. Note that when we presented $transLTL$ earlier, we assumed that the input is a formula of form $E\Phi$ such that all state subformulas of Φ are literals. But in the pseudo-code in Fig 3, this need not be true; the argument formula $E\Phi$ to $transLTL$ may contain state subformulas which are not literals. In this case $transLTL$ first translates the top-level state subformulas by invoking $transCTL^*$ and then proceeding as described earlier.

Theorem 4.5 *Let p be a CTL* formula, and let s be a state in Kripke structure $\mathcal{T}$. Then $s \models_{\mathcal{T}} p$ iff $s \in [\![transCTL^*(p)]\!]_{\mathcal{T}} e$.*

Complexity results. We present the complexity results for the LTL translation procedure first. The time complexity of the translation procedure for LTL is determined by the size of the BTA corresponding to the formula. This may be characterized as follows.

Lemma 4.6 *Let $p \equiv E\phi$ be a LTL formula. Then the size of BTA $\mathcal{B}_p$ corresponding to p is bounded by $2^{O(|p|)}$.*

Proof. Follows from the fact that for each formula p' that appears in p's tableau, every $\phi' \in p'$ is either a subformula of p or of form $\mathsf{X}q$ where q is a subformula of p. □
We now give complexity results for CTL* and for the sublogic CTL.

Theorem 4.7 *The time complexity for $transCTL^*$ is bounded by $2^{O(|p|)}$ if p is a CTL* formula. If p is a CTL formula then the bound is $O(|p|)$.*

Proof. By induction on the structure of the formula □

4.1. ECTL* to the equational μ-calculus

We give a translation for the sublogic ELTL. This translation can then be extended to ECTL* using the same technique we used to extend the LTL translation to CTL*. Let $p \equiv E(\mathcal{B})$, where $\mathcal{B} \equiv \langle S, 2^{\mathcal{A}}, \rightarrow, s_0, F \rangle$, be an ELTL formula. The BTA $\langle Q, \mathcal{A}, \rightarrow_1, q_0, F_1, l \rangle$ corresponding to p is defined as follows. For each state $s \in S$ we introduce a state q_s in Q. Also, for each transition $s \xrightarrow{A} s'$ we introduce in Q states $q^{\wedge}_{s \xrightarrow{A} s'}$, $q^{\langle\rangle}_{s \xrightarrow{A} s'}$ and q_a for each $a \in A$. The labelling for these states is defined as follows.

- $\ell(q_s) = \vee$
- $\ell(q^{\wedge}_{s \xrightarrow{A} s'}) = \wedge$
- $\ell(q^{\langle\rangle}_{s \xrightarrow{A} s'}) = \langle\rangle$
- $\ell(q_a) = a$

For each transition $s \xrightarrow{A} s'$, we introduce a transition $q_s \rightarrow_1 q^{\wedge}_{s \xrightarrow{A} s'}$. State $q^{\wedge}_{s \xrightarrow{A} s'}$ has transitions $q^{\wedge}_{s \xrightarrow{A} s'} \rightarrow_1 q^{\langle\rangle}_{s \xrightarrow{A} s'}$ and $q^{\wedge}_{s \xrightarrow{A} s'} \rightarrow_1 q_a$ for each $a \in A$. State $q^{\langle\rangle}_{s \xrightarrow{A} s'}$ has one transition $q^{\langle\rangle}_{s \xrightarrow{A} s'} \rightarrow_1 q_{s'}$. The set of final states $F_1 is \{ q_s \mid s \in F \}$.

Theorem 4.8 *Let $p \equiv E\Phi$ be an ELTL formula with $\mathcal{B}_p$ as the BTA obtained by the translation procedure described above. Let $\mathcal{T} \equiv \langle S, R, L \rangle$ be a Kripke structure. Then for $s \in S$, $s \models_{\mathcal{T}} p$ iff $\langle S, R, L, s \rangle$ is accepted by $\mathcal{B}_p$.*

It is clear that the above translation gives a BTA with size linear in the size of the input formula. It then follows that the translation from ELTL for the equational μ-calculus has time complexity and succintness linear in the size of the input formula. For ECTL* the same result can be proved by induction on the structure of the formula.

procedure $transCTL^*(p)$
case p
$a : a$
$p_1 \vee p_2 : transCTL^*(p_1) \vee transCTL^*(p_2)$
$p_1 \wedge p_2 : transCTL^*(p_1) \wedge transCTL^*(p_2)$
$E\Phi : transLTL(E\Phi)$
$A\Phi : \mathsf{neg}(transLTL(\mathsf{neg}(A\Phi)))$
endcase
end $transCTL^*$

Figure 3. The translation procedure for CTL*

5. Efficient CTL* Model Checking via the μ-Calculus

So far in this paper we have addressed the complexity issues involved in converting CTL* and ECTL* formulas into the equational μ-calculus. We now investigate efficiency issues involved in checking Kripke structures against these translations.

It is straightforward to extend existing model-checkers for the μ-calculus to handle the equational variant; indeed, many already work on similar equational notation [3, 4, 11, 13]. It is also known that using these model checkers in conjunction with translation procedures into the μ-calculus gives routines that are equivalent in terms of worst-case behavior to the most efficient model-checking algorithms for "pure branching" logics CTL and PDL-δ. This follows from the fact that existing translators (and ours) yield alternation-free μ-calculus formulas in these cases.

For CTL* and ECTL* it is not immediately clear that combining a μ-calculus model checker with our translation will necessarily yield a CTL* model checker that performs as well as existing algorithms. The reason for this is that the formulas we generate have alternation depth two, and existing general-purpose procedures in general exhibit quadratic behavior when applied to such propositions. However, our formulas have a special form that permit us to exploit the *fixpoint inversion* technique of [2] to achieve a linear-time algorithm for them. We defer a discussion of the details to the full version of the paper, but the intuition is the following. Call a variable X' in μ-calculus formula $X\ in\ \mathcal{B}$ *disjunctive* if $\mathsf{rhs}(\mathcal{B}, X')$ has form $X'_1 \vee X'_2$ or $\langle\cdot\rangle X'_1$. Also define $X' \rightsquigarrow X''$ to hold if X' appears in $\mathsf{rhs}(\mathcal{B}, X'')$. Now, it can be shown that (modulo a technical detail regarding conjunctions containing literals) formulas produced by $transLTL$ contain $\rightsquigarrow$-cycles that only involve disjunctive variables. Results in [2] show that such cycles may be "eliminated" during model checking; the net effect is that model-checking of alternation-depth-two formulas may be in essence "reduced" to alternation-free model checking. Thus, model checking on formulas produced by $transLTL$ requires time linear in the size of the formula. The same also holds for formulas produced by $transCTL^*$ and by our ECTL* translator. Combining this model-checking technology with our translator, we obtain model checkers for these logics whose complexities match those of the best existing model-checking algorithms for this these logics. Anderson et al. [2] have also shown that fixpoint inversion may be used to reduce the number of iterations in a BDD-based implementation of their model-checking algorithm. As a result, our approach yields an efficient BDD-based model checker for both CTL* and ECTL*.

6. Conclusions

In this paper we demonstrated the feasibility of using an equational variant of the modal μ-calculus as an efficient basis for temporal-logic model checking. Our approach relies on an efficient translation of the temporal logic CTL* [18] and ECTL* [22, 28]into this logic coupled with observations about the complexity of model-checking the resulting formulas. Our routine also yields efficient procedures for various sublogics of CTL*, including CTL and LTL. Based on this translation, one may use techniques developed in the context of the μ-calculus for coping with state explosion, such as BDD-based algorithms [6], on-the-fly routines [10, 19, 26, 30], and partial model-checking [4], to these logics as well. The translation procedure is very simple and easy to implement; this is the subject of ongoing work in the context of the Concurrency Workbench [12].

The translation relies on the use of a kind of tree automata that we call Büchi tableau automata. As future work, we would like to investigate using these automata as a basis for developing compositional model-checking techniques based on "automaton factorization". In particular, it appears that using these automata one may fruitfully generalize the partial model-checking approach of [4].

References

[1] *Symposium on Logic in Computer Science (LICS '86)*, Cambridge, Massachusetts, June 1986. Computer Society Press.

[2] Henrik R. Andersen and Bart Vergauwen. Efficient checking of behavioural relations and modal assertions using fixed-point inversion. In Pierre Wolper, editor, *Proceedings of the 7th International Conference on Computer Aided Verification, CAV'95*, volume 939 of *Lecture Notes in Computer Science*, pages 142–154, Liège, Belgium, July 1995. Springer-Verlag.

[3] H.R. Andersen. Model checking and boolean graphs. In *Proceedings of the European Symposium on Programming*, volume 582 of *Lecture Notes in Computer Science*, pages 1–19, Rennes, France, March 1992. Springer-Verlag.

[4] H.R. Anderson. Partial model checking. In *Tenth Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1995.

[5] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In Dill [16], pages 142–155.

[6] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Proceedings of the IEEE/ACM Design Automation Conference*, 1990.

[7] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Fifth Annual Symposium on Logic in Computer Science (LICS '90)*, pages 428–439, Philadelphia, June 1990. Computer Society Press.

[8] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[9] E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In Dill [16], pages 415–427.

[10] R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–747, September 1990.

[11] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In G.v. Bochmann and D.K. Probst, editors, *Computer Aided Verification (CAV '92)*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422, Montréal, June/July 1992. Springer-Verlag.

[12] R. Cleaveland, J. Parrow, and B. Steffen. A semantics-based tool for the verification of finite-state systems. In *Proceedings of the IFIP Symposium on Protocol Specification, Testing and Verification*, pages 287–302, Enschede, The Netherlands, June 1989. North-Holland.

[13] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In K.G. Larsen and A. Skou, editors, *Computer Aided Verification (CAV '91)*, volume 575 of *Lecture Notes in Computer Science*, pages 48–58, Aalborg, Denmark, July 1991. Springer-Verlag.

[14] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[15] M. Dam. CTL* and ECTL* as fragments of the modal mu-calculus. In *Proceedings of the Colloquium on Trees and Algebra in Programming*, volume 581 of *Lecture Notes in Computer Science*, pages 145–164. Springer-Verlag, February 1992.

[16] D.L. Dill, editor. *Computer Aided Verification (CAV '93)*, volume 818 of *Lecture Notes in Computer Science*, Stanford, California, June 1994. Springer-Verlag.

[17] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Symposium on Logic in Computer Science (LICS '86)* [1], pages 267–278.

[18] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[19] G.Bhat and R.Cleaveland. Efficient local model checking for fragments of the μ-calculus. In *To appear in Proceedings TACAS 96'*, 1996.

[20] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[21] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Twelfth Annual ACM Symposium on Principles of Programming Languages (PoPL '85)*, pages 97–107, Orlando, Florida, January 1985. Computer Society Press.

[22] M.Y.Vardi and P.Wolper. Yet another process logic. *Lecture Notes in Computer Science*, 164:501–512, 1984.

[23] O.Bernholtz and O.Grumberg. Branching time temporal logics and amorphous tree automata. In E. Best, editor, *CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, Hildesheim, Germany, August 1993. Springer-Verlag.

[24] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Int'l Symp. on Programming*, volume 137, pages 337–351. Springer-Verlag, Lecture Notes in Computer Science, 1981.

[25] R.Gerth, D.Peled, M.Vardi, and P.Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV'95*, 1995.

[26] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In *TAPSOFT*, volume 352 of *Lecture Notes in Computer Science*, pages 369–383, Barcelona, March 1989. Springer-Verlag.

[27] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 25(2):285–309, 1955.

[28] W. Thomas. Computation tree logic and regular ω-languages. volume 354 of *Lecture Notes in Computer Science*, pages 690–713. 1988.

[29] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS '86)* [1], pages 332–344.

[30] G. Winskel. A note on model checking the modal ν-calculus. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Automata, Languages and Programming (ICALP '89)*, volume 372 of *Lecture Notes in Computer Science*, pages 761–772, Stresa, Italy, July 1989. Springer-Verlag.

A. Semantics of CTL*

Definition A.1 *Let $\mathcal{T} = \langle S, R, L\rangle$ be a Kripke structure, with $s \in S$ and x a path in M. Then $\models_\mathcal{T}$ is defined inductively as follows.*

- $s \models_\mathcal{T} a$ *if* $a \in L(s)$ *(recall* $a \in \mathcal{A}$*).*
- $s \models_\mathcal{T} \neg a$ *if* $s \not\models_\mathcal{T} a$.
- $s \models_\mathcal{T} p_1 \wedge p_2$ *if* $s \models_\mathcal{T} p_1$ *and* $s \models_\mathcal{T} p_2$.
- $s \models_\mathcal{T} p_1 \vee p_2$ *if* $s \models_\mathcal{T} p_1$ *or* $s \models_\mathcal{T} p_2$.
- $s \models_\mathcal{T} A\phi$ *if for every* $x \in \Pi_\mathcal{T}(s)$, $x \models_\mathcal{T} \phi$.
- $s \models_\mathcal{T} E\phi$ *if there exists* $x \in \Pi_\mathcal{T}(s)$ *such that* $x \models_\mathcal{T} \phi$.
- $x \models_\mathcal{T} p$ *if* $x(0) \models p$ *(recall* p *is a state formula).*
- $x \models_\mathcal{T} \phi_1 \wedge \phi_2$ *if* $x \models_\mathcal{T} \phi_1$ *and* $x \models_\mathcal{T} \phi_2$.
- $x \models_\mathcal{T} \phi_1 \vee \phi_2$ *if* $x \models_\mathcal{T} \phi_1$ *or* $x \models_\mathcal{T} \phi_2$.
- $x \models_\mathcal{T} \mathsf{X}\phi$ *if* $x^1 \models_\mathcal{T} \phi$.
- $x \models_\mathcal{T} \phi_1 \mathsf{U} \phi_2$ *if there exists* $i \geq 0$ *such that* $x^i \models_\mathcal{T} \phi_2$ *and for all* $j < i$, $x^j \models_\mathcal{T} \phi_1$.
- $x \models_\mathcal{T} \phi_1 \mathsf{V} \phi_2$ *if for all* $i \geq 0$, $x^i \models_\mathcal{T} \phi_2$ *or if there exists* $i \geq 0$ *such that* $x^i \models_\mathcal{T} \phi_1$ *and for every* $j \leq i$, $x^j \models_\mathcal{T} \phi_2$.

B. Semantics of ECTL*

ECTL* formulas are also interpreted with respect to Kripke structures. The semantics of atomic propositions, conjunction and disjunction is as usual. To define the semantics of the $E(\mathcal{B})$ formulas we do the following. For a path $x \equiv x_0, x_1, \ldots$ in the Kripke structure $M \equiv \langle S, R, L\rangle$ define $trans(x)$ to be the ω-word such that $trans(x)(i) = \{\, \Phi \in \{\Phi_1, .., \Phi_n\} \mid s \models_M \Phi \,\}$. Now, $s \models_M E(\mathcal{B})$ iff there exists a path $x \in \Pi_M(s)$ such that $trans(x)$ is accepted by $\mathcal{B}$.