Model Checking Meets Theorem Proving: a Situation Calculus Based Approach

Yilan Gu
Dept. of Computer Science
University of Toronto
yilan@cs.toronto.edu

Iluju Kiringa
SITE
University of Ottawa
kiringa@site.uottawa.ca

Abstract

To reason about properties of reactive programs, one may usually follow either an operational or a deductive approach. In this paper, we propose representing the classical model checking approach of Clarke and Emerson in the situation calculus. Doing so, we propose an approach that merges the operational and the deductive approaches into one single framework by translating Kripke models that represent system specifications into theories formulated in the situation calculus and by recasting CTL as a sublanguage of the calculus.

Introduction

The importance of long running, nondeterministic concurrent programs has been emphasized over the past two and half decades since Pnueli proposed using temporal logic for reasoning about them (Pnueli 1977). These, also called reactive systems, as opposed to sequential transformational programs, show ideally nonterminating behaviors (Clarke, Grumberg, & Peled 1999). Their mathematical properties are usually defined using either the operational or the deductive approach. In the operational approach, programs are viewed as generator of computations. Given a program, all the computations associated with it can be generated once by an interpreter, or incrementally by specifying a transition relation that holds between consecutive states of the computation of the program. In summary, the operational semantics is based on the structure of the given program (Plotkin 1981). In the deductive approach, programs are viewed as specifying a set of computations about which some statements can be proven. Dynamic logic (Harel, Tiuryn, & Kozen 2000), Hoare's systems (Hoare 1969), and the situation calculus (McCarthy 1963; Reiter 2001) are examples of formalisms used in this approach.

The semantics of concurrent programs is described in either approaches in terms of infinite behaviors, also called computations. A behavior is a sequence of states that a program moves through while executing. The behaviors are all the possible interleavings of the "atomic" steps of the subprograms running in parallel; that is, given a concurrent program P composed of subprograms $P_1, P_2, \ldots, P_n$, where the P_is, $1 \leq i \leq n$, are sequential programs running in parallel, its execution is usually modeled by nondeterministically executing the atomic steps for each P_i, $1 \leq i \leq n$, in an arbitrary order. So if P is in a state s_k, it nondeterministically goes to the next state s_{k+1} by executing an arbitrary atomic step of any of its subprograms P_i. This procedure is repeated infinitely, or at least indefinitely.

Linear and branching temporal logics are the most commonly used languages for describing computations. The model checking problem (**MC**) can be defined as follows: Given (1) a reactive system $\mathfrak{S}$ represented as a finite-state structure which generates computations, and (2) a temporal logic formula $\mathfrak{P}$ specifying a property of $\mathfrak{S}$, find whether $\mathfrak{S}$ satisfies $\mathfrak{P}$. There are successful algorithmic solutions of **MC**. As an example, in (Clarke & Sistla 1986), Kripke Structures are used to represent the reactive system and Computational Tree Logic (CTL), a branching time temporal logic, is used to represent properties of the system.

In this paper, we propose representing the classical **MC** approach of (Clarke & Sistla 1986) in the situation calculus. Our approach merges both the operational and the deductive approaches into a single framework by translating Kripke models into theories of the situation calculus and by recasting CTL as a sublanguage of the same calculus. This approach can be labeled as deductive-operational in the sense of (Pnueli 1981); that is, it deals with computations arising during program execution and, at the same time, allows us to deductively reason about those computations in the logic of the situation calculus.

In (De Giacomo, Ternovskaia, & Reiter 1997), it is argued that, for non-terminating programs, one needs to rely on a transition semantics, in which one allows for interpreting and quantifying over parts of programs and their executions. In this paper, we show that an evaluation semantics in which one allows for interpreting whole programs is possible for non-terminating programs.

The paper is organized as follows. In the next section, we introduce the situation calculus, and the classical model of concurrent systems in terms of Kripke structures. Then we give an effective method for translating a Kripke structure to a basic action theory. Next, we

show how CTL properties are specified in the situation calculus and treat the model checking of action theories within our framework. This is followed by the presentation of an an example which illustrates our approach of model checking in the situation calculus. Finally, we conclude the paper and indicate avenues for future work.

Preliminaries

The Situation Calculus

The situation calculus (McCarthy 1963; Reiter 2001) is a many-sorted second order language with equality specifically designed for representing dynamically changing world. We consider a version of the situation calculus with three sorts for actions ($\mathcal{A}$), situations ($\mathcal{S}$), and objects ($\mathcal{O}$) other than the first two. **Actions** are first order terms consisting of a $0-ary$ action function symbols corresponding to the transitions of the finite state structures. **Situations** are first order terms denoting a sequence of actions. They are represented using a binary function symbol do: $do(\alpha, s)$ denotes the sequence resulting from adding the action α to the sequence s. The constant S_0 (*initial situation*) denotes the empty sequence [ ]. In modeling systems, situations will correspond to computations. **Objects** constitute a catch-all sort representing everything else depending on the domain of application.

We shall have a finite number of unary predicates called **fluents** which represent properties with truth values varying from state to state. Fluents are denoted by predicate symbols with argument a situation term. In a Reader-Writer example given below (Example 1), $state_1(s)$ is a relational fluent, meaning that the system is in state w_1 after performing the sequence of operations in the computation s. In addition to the fluents, we shall have a finite number of ground situation independent predicates. For example, we will use ground binary predicate $trans(I, J)$ to represent a transition form state w_I to w_J of the system being modeled.

The language also includes special predicates $Poss$, and $\sqsubset$; $Poss(a, s)$ means that the action a is possible in the situation s, and $s \sqsubset s'$ states that the situation s' is reachable from s by performing some sequence of actions. In system modeling terms, $s \sqsubset s'$ means that s is a proper subcomputation of the computation s'. The predicate $\sqsubset$ will be useful in formulating properties of systems. We call this fragment of the situation calculus ${\mathcal{L}_0}^0$. In general, we can define fragments ${\mathcal{L}_j}^i$, where i (j) is the maximum number of arguments of sort $\mathcal{O}$ that an action function (fluent predicate) may have.

Axiomatizing a Domain Theory

A domain theory is axiomatized in the situation calculus with four classes of axioms which constitute a *basic action theory* (BAT – More details in (Pirri & Reiter 1999)):[1]

[1]There are also **unique names axioms** which guarantee that primitive actions of the domain are pairwise unequal.

Foundational axioms for situations ($\mathcal{D}_f$). These guarantee an infinite tree structure for the situations, and are the same for all BATs.

Action precondition axioms ($\mathcal{D}_{ap}$). There is one for each action function A, with syntactic form $Poss(A, s) \equiv \Pi_A(s)$. Here, $\Pi_A(s)$ is a formula with free variable s. These characterize the preconditions for doing action A in the situation s.

In the Reader-Writer example, the following states that it is possible for the system to move from state 1 to state 2 relative to the system computation s iff there is a transition from state 1 to state 2, and as a result of performing the actions in that computation, the system is in state 1.

$$Poss(tr_{1,2}, s) \equiv trans(1, 2) \wedge state_1(s).$$

Successor state axioms ($\mathcal{D}_{ss}$). There is one for each relational fluent $F(s)$, with syntactic form $F(do(a, s)) \equiv \Phi_F(a, s)$, where $\Phi_F(a, s)$ is a formula with free variables among a and s. These characterize the truth values of the fluent F in the next situation $do(a, s)$ in terms of the current situation s, and they embody a solution to the frame problem for deterministic actions (Reiter 2001).

In the Reader-Writer example, the following states that the system will be in the state 1 relative to the computation $do(a, s)$ iff the last system operation a in the computation was $tr_{4,1}$ or $tr_{6,1}$, or it was already in state 1 relative to the computation s, and a does not lead the system to another state.

$$state_1(do(a, s)) \equiv a = tr_{4,1} \vee a = tr_{6,1} \vee$$
$$state_1(s) \wedge a \neq tr_{1,2} \wedge a \neq tr_{1,3}.$$

Initial database ($\mathcal{D}_{S_0}$). This is a set of first order sentences whose only situation term is S_0; it specifies the initial state of the domain, in our case, the initial system state. Notice that while these initial system axioms specify a complete initial system state (as is normal for reactive systems), this is not a requirement of the theory we are presenting. Therefore our account could, for example, accommodate incomplete initial system states.

Notations

We now introduce further notations used later in the paper. Suppose $\mathcal{D}$ is a basic action theory. Furthermore, suppose that $\mathcal{U}_\mathcal{A} = \{A_1, \ldots, A_k\}$ is the (finite) set of actions of $\mathcal{D}$, ${\mathcal{U}_\mathcal{A}}^*$ is the set of action sequences, and $\mathcal{F} = \{F_1, \ldots, F_n\}$ is the (finite) set of fluents of $\mathcal{D}$, and $V = \{0, 1\}$ is a set of labels denoting the truth values. Then $\vec{v}_0 = \langle v_{0_1}, \ldots, v_{0_n} \rangle$ denotes the vector of initial truth values of fluents of $\mathcal{D}$, where v_{0_j} with $1 \leq j \leq n$ is the initial value of fluent F_j, and $c_i(\alpha)$ specifies whether the fluent F_i holds in the situation represented by the action sequence α; c_i is called the *characteristic function* (Ternovskaia 1999) of fluent F_i.

Checking a Situation Calculus System

Let $\mathcal{D}$ be a background situation calculus axiomatization for some reactive system, as described above, and

let $Q(s)$ be a situation calculus formula – a property – with one free situation variable s.

Let $S = do(\alpha_n, do(\alpha_{n-1}, \cdots, do(\alpha_1, S_0) \cdots))$ be a situation term that mentions no free variables. We treat this as a system computation, and define the *answer* to Q *relative to this computation* to be "yes" iff $\mathcal{D} \models Q(S)$. The answer is "no" iff $\mathcal{D} \models \neg Q(S)$. So on this definition, model checking is performed relative to a system computation, and, in the most general setting, it is a theorem-proving task. In particular, the executability problem is to check whether $\mathcal{D} \models executable(S)$, where

$$executable(s) =_{df} (\forall a, s').do(a, s') \sqsubseteq s' \supset Poss(a, s').$$

It is important to notice the following property of basic action theories formulated in the language of Subsection "The Situation Calculus".

Theorem 1 *The basic action theories formulated in the fragment ${\mathcal{L}_0}^0$ of the situation calculus are decidable. That is, suppose $\mathcal{D}$ is a BAT, and ϕ is a formula, all of which are formulated in ${\mathcal{L}_0}^0$; then there is an algorithm for establishing whether $\mathcal{D} \models \phi$.*

Proof (outline):
Similar to the idea in This is a corollary of Theorem 3 in (Ternovskaia 1999) that shows decidability for a similar fragment of the situation calculus, but where there are no situation independent predicates. The proof there is now augmented by showing that adding finitely many ground situation independent predicates does not change the nature of the automata constructed for $\mathcal{D}$ and ϕ. □

GOLOG

GOLOG (Levesque *et al.* 1997) is a situation calculus-based programming language for defining complex actions in terms of a set of primitive actions axiomatized in the situation calculus according to Subsection "Axiomatizing a Domain Theory". It has control structures found in most Algol-like languages, augmented by some nonstandard structures: *Sequence* (α ; β: Do action α, followed by action β); *Test actions* (p?: Test the truth value of expression p in the current situation); *Nondeterministic action choice* ($\alpha \mid \beta$: Do α or β); *Nondeterministic choice of arguments* ($(\pi\ x)\alpha$: Nondeterministically pick a value for x, and for that value of x, do action α); Conditionals (*if-then-else*) and *while* loops; and *Procedures, including recursion.*

The following is a GOLOG procedure that executes an sequence of n randomly picked actions which are possible:

proc $execActions(n)$
$n = 0? \mid$
$n > 0?\ ;\ (\pi\ a)[Poss(a)?\ ;\ a]; execActions(n-1)$
endProc .

The semantics of GOLOG programs is defined by macro-expansion, using a ternary relation Do (Levesque *et al.* 1997); $Do(\mathtt{P}, s, s')$ is an *abbreviation* for a situation calculus formula which intuitively means that s' is one of the situations reached by evaluating the GOLOG program P, beginning in situation s. In the reactive system setting, any binding for s' represents the system computation that results from executing P, beginning in the system state defined by the computation s.

Concurrent Systems

Concurrent reactive systems are semantically characterized by transition systems (Wolper 1998; Clarke, Grumberg, & Peled 1999). Usually, the latter are modeled by *Kripke structures* (Clarke, Grumberg, & Peled 1999) which we now introduce.

Definition 1 (Kripke structure) *A finite Kripke structure is a quintuple $\mathbf{K} = (P, W, R, w_0, L)$ where*

- *P is a finite set of atomic propositions;*
- *W is a finite set of states;*
- *$R \subseteq W \times W$ is a total (transition) relation;*
- *w_0 is an initial state;*
- *$L : W \rightarrow 2^P$ maps each $w \in W$ to the set $\{p \in P \mid \models_w p\}$.*

Definition 2 (Behavior) *Suppose $\mathbf{K} = (P, W, R, w_0, L)$ is a Kripke structure. Then a behavior σ of $\mathbf{K}$ is a function from N, a subset of the natural numbers, to W such that:*

- *$N = \{0, 1, .., n\}$ for some natural number $n \in \mathbb{N}$, or N is the set of natural numbers;*
- *$\sigma(0) = w_0$;*
- *$\forall i \geq 0\ \ (\sigma(i), \sigma(i+1)) \in R$.*

If N equals the set $\mathbb{N}$ of natural numbers, then σ is called an infinite behavior.

Example 1 *Consider a concurrent system – denoted by RW – consisting of a Reader process, numbered 1, and a Writer process, numbered 2 (Emerson & Trefler 1999). We define RW as follows. Each of these processes can be in three states: Non-Trying, Trying, and Critical Section. These are thus subscripted accordingly: N_i, T_i, and C_i refer the Non-Trying, Trying, and Critical Section states of Process i. Process 1 may enter its critical section only when Process 2 is in its Non-trying section, and Process 2 may enter its critical section only when Process 1 is in its Non-Trying or Trying states. Figure 1 is a Kripke structure representing all the reachable states of the RW system. Formally, we have the following:*
$P = \{N_1, N_2, T_1, T_2, C_1, C_2\}$; $W = \{w_0, \cdots, w_7\}$;
$R = \{(w_0, w_1), (w_0, w_2), (w_1, w_3), (w_1, w_4), (w_2, w_4), (w_2, w_5), (w_3, w_0), (w_3, w_6), (w_4, w_7), (w_5, w_0), (w_5, w_7), (w_6, w_2), (w_7, w_1)\}$;
the initial state w_0; and $L(w_0) = \{N_1, N_2\}, L(w_1) = \{T_1, N_2\}, L(w_2) = \{N_1, T_2\}, L(w_3) = \{C_1, N_2\}, L(w_4) = \{T_1, T_2\}, L(w_5) = \{N_1, C_2\}, L(w_6) = \{C_1, T_2\}, L(w_7) = \{T_1, C_2\}$.

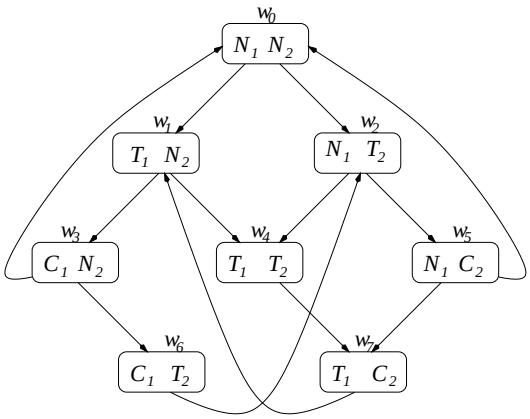

Figure 1: Reader-Writer transition system

One may unwind a Kripke structure into an infinite tree that is rooted in w_0. Such trees are called *computational trees*.

Definition 3 (Computational Tree) *Suppose* $\mathbf{K} = (P, W, R, w_0, L)$ *is a Kripke structure. Then the (infinite) computational tree* $CT_{\mathbf{K}}$ *of* $\mathbf{K}$ *is the set* $\{\sigma_1, \sigma_2, \ldots\}$ *of all (infinite) behaviors of* $\mathbf{K}$*; that is,*

- *for each* $i = 1, 2, \ldots$, σ_i *is a function from* $\mathbb{N}$ *to* W*;*
- $\sigma_i(0) = w_0$ *for all* $i = 1, 2, \ldots$*;*
- $\forall j > 0, i \geq 0 \;\; (\sigma_j(i), \sigma_j(i+1)) \in R$.

The top tree in Figure 2 shows the infinite computational tree of the RW system depicted in Figure 1. Thus any path starting in the root of the computational tree represents a behavior of the Kripke structure.

Translating Concurrent Systems into BATs

Recall that the foundational axioms guarantee the infinite tree structure of the situation calculus. We shall translate Kripke structures to the situation calculus by relating the idea of computational tree to the situation calculus tree of situations. More precisely, the Kripke structure will be translated into a BAT such that the computational tree of the Kripke structure is represented as a subtree obtained from the tree of situations by pruning away paths that are not executable.

In order to relate Kripke structures to BATs, we need to define a situation tree like model for a BAT $\mathcal{D}$; that model is precisely a tree of situations constrained appropriately using the successor state axioms and action precondition axioms of $\mathcal{D}$.

Definition 4 (Canonical Structure)[2] *Suppose* $\mathcal{D} = \mathcal{D}_f \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{S_0}$ *is a BAT. Then a structure* $\mathfrak{M}$ *is a canonical structure for* $\mathcal{D}$ *iff it is a pair* (D, L_D)*, where*

- $D \subseteq {\mathcal{U}_{\mathcal{A}}}^*$ *is the domain of* $\mathfrak{M}$*, satisfying the following property: if an action sequence* α *is in* D *then any prefix* α' *of* α *(i.e.,* $\alpha = \alpha'\alpha''$ *for some* $\alpha'' \in {\mathcal{U}_{\mathcal{A}}}^*$*) is in* D*;*
- L_D *is a labeling function* $D \rightarrow V^n$ *such that* $L_D([\,]) = \vec{v}_0$ *and* $L_D(\alpha) = \langle c_1(\alpha), \ldots, c_n(\alpha)\rangle \in V^n$.

  Here, $\vec{v}_0$ *is the initial vector of fluent values, and* c_i *is the characteristic function of fluent* F_i.

Notice that in the definition above, the pair (D, L_D) is in fact a situation tree constrained using the BAT $\mathcal{D}$. The domain D is the set of nodes of the tree; and for any sequence α and $1 \leq j \leq k$, αA_j is the j-th son of the node α labeled by value vector $L_D(\alpha)$. Figure 2 shows a canonical structure for the RW system depicted in Figure 1. The labeling $tr_{i,j}$ of the edges denotes an action corresponding to the transition $(S_i, S_j) \in R$. Details of the BAT underlying this tree will be clearer in Section "An Example". It suffices here to mention that $\langle N_1, N_2, T_1, T_2, C_1, C_2\rangle$ is the vector of fluents describing the properties of the system. The vector $\vec{v}_0$ of initial values for the root of the situation tree is $\langle 1, 1, 0, 0, 0, 0\rangle$, and the vectors labeling the other nodes of the tree depend on the characteristic functions of each fluent.

Now, we show how to effectively construct a basic action theory from a given Kripke structure.

Theorem 2 *Suppose* $\mathbf{K} = (P, W, R, w_0, L)$ *is a Kripke structure. Then one can effectively construct a BAT* $\mathcal{D}_{\mathbf{K}}$ *whose canonical structure* $\mathfrak{M}$ *is obtained from the computational tree* $CT_{\mathbf{K}}$ *of* $\mathbf{K}$ *such that*

$$\mathbf{K} \;\; has \;\; CT_{\mathbf{K}} \;\; iff \;\; \models_{\mathfrak{M}} \mathcal{D}_{\mathbf{K}}.$$

Proof:
Let $\mathcal{D}_{\mathbf{K}} = \mathcal{D}_f \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{S_0}$ be the following BAT.

Fluents: for each $p \in P$, introduce a fluent $p(s)$; for each state $w_i \in W$, introduce a fluent $state_i(s)$.

Actions: for each transition $(w_i, w_j) \in R$ where $w_i, w_j \in W$, introduce an action $tr_{i,j}$.

Initial database ($\mathcal{D}_{S_0}$): Whenever $p \in L(w_0)$, introduce the axiom $p(S_0)$, otherwise introduce $\neg p(S_0)$; introduce axiom $state_0(S_0)$ and, for all $w_i \neq w_0$, introduce axioms $\neg state_i(S_0)$. We also need to introduce finitely many non-fluent predicates $tr(i, j)$ where $0 \leq i, j < |W|$, such that $trans(i, j)$ is true if and only if $(w_i, w_j) \in R$.

Action precondition axioms ($\mathcal{D}_{ap}$): for each transition $tr_{i,j}$, we have the axiom

$$Poss(tr_{i,j}, s) \equiv trans(i, j) \wedge state_i(s).$$

Successor state axioms ($\mathcal{D}_{ss}$): For every i, $0 \leq i < |W|$,

[2] This definition is similar to the notion of k-ary n-labeled situation tree associated with a BAT defined in (Ternovskaia 1999).

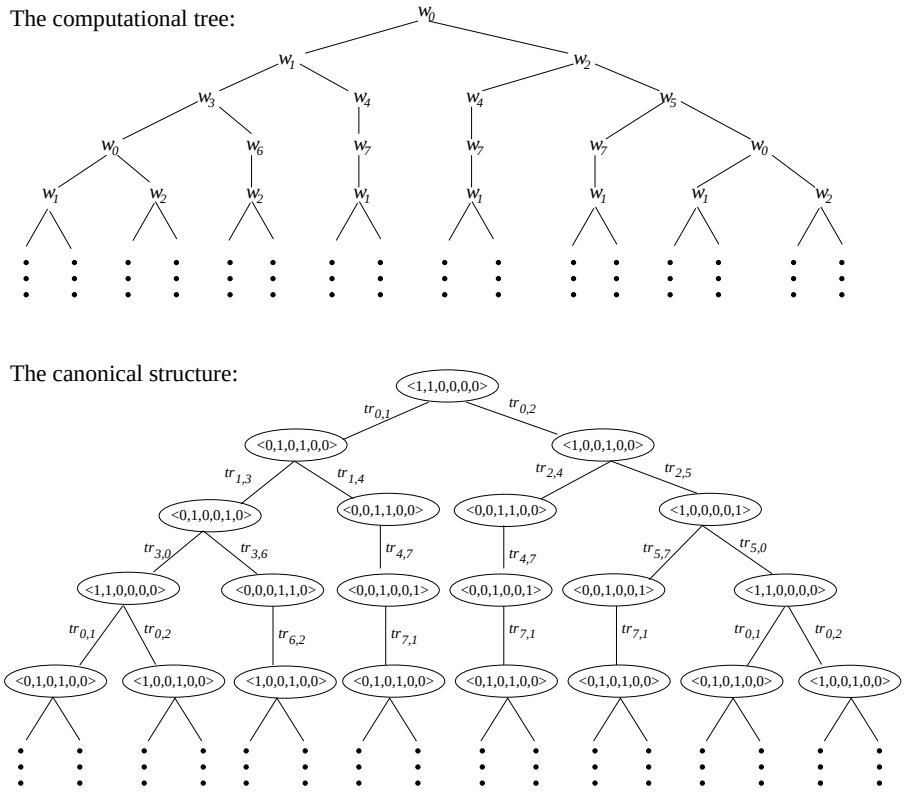

Figure 2: Computation tree and canonical structure of the RW system

$$state_i(do(a,s)) \equiv \bigvee_{j=1}^{|W|} a = tr_{j,i} \vee state_i(s) \wedge \bigwedge_{j=1}^{|W|} a \neq tr_{i,j}.$$

For every $p \in P$, suppose $W_p = \{w_i | \models_{w_i} p\}$. Then,

$$p(s) \equiv \bigvee_{w_i \in W_p} state_i(s).$$

Hence we could easily get the successor state axiom for $p(s)$.

Now we show that $\mathbf{K}$ has $CT_{\mathbf{K}}$ iff $\models_{\mathfrak{M}} \mathcal{D}_{\mathbf{K}}$. Suppose $\mathbf{K}$ has $CT_{\mathbf{K}}$. Then the proof proceeds by constructing the canonical structure corresponding to $CT_{\mathbf{K}}$. It is easy to show that the BAT constructed above is satisfiable in this canonical structure.

Suppose $\models_{\mathfrak{M}} \mathcal{D}_{\mathbf{K}}$ and $D \subseteq \mathcal{U}_{\mathcal{A}}{}^*$. Then $L_D([\,]) = \vec{v}_0$ specifies the root of $CT_{\mathbf{K}}$ by telling exactly which fluent of the form $state_0$ is true and which other fluents are true in the state w_0 of the Kripke structure $\mathbf{K}$. Furthermore, $L_D(\omega A_j) = \langle c_1(w), \ldots, c_n(w)\rangle \in V^n$ determines the fluent values in situation ωA_j by telling which fluent of the form $state_i$ is true and which other fluents are true in the state w of $\mathbf{K}$ corresponding to the execution of the transitions encoded in ωA_j. Thus each path in $\mathfrak{M}$ starting in $[\,]$ yields a corresponding path in $CT_{\mathbf{K}}$. □

Model Checking

CTL

The temporal logics that are used for specifying properties in **MC** are subsets of the logic CTL* (Clarke & Sistla 1986) which expresses a branching time logic by extending linear time temporal logic with behavior quantifiers. The logic CTL is the smallest set of formulas inductively defined as follows.

Situation formulas

– `true` and `false` are atomic situation formulas, as well as are p and $\neg p$ for all $p \in P$.

– If ϕ and ψ are situation formulas, then $\phi \wedge \psi$ and $\phi \vee \psi$ are situation formulas.

– If ϕ is a behavior formula, then $A\phi$ ("ϕ holds for all behaviors") and $E\phi$ ("ϕ holds for some behavior") are situation formulas.

Behavior formulas

– If ϕ and ψ are situation formulas, then $X\phi$ ("next time ϕ"), and $\phi U \psi$ ("ϕ until ψ") are behavior formulas.

Moreover, $F\phi$ ("ϕ holds at some future state on a behavior") and $G\phi$ ("ϕ holds at all future states on a behavior") for behavior formula ϕ abbreviate `true`$U\phi$ and $\neg F \neg \phi$ respectively.

Semantics

Here, we semantically characterize CTL formulas by translating them to formulas of the decidable fragment of the situation calculus described in Subsection "The Situation Calculus". CTL formulas are interpreted over Kripke structures. We shall denote the suffix of the behavior $\sigma = S_0, s_1, s_2, \ldots$ that starts at situation s_j by σ^j. Given a Kripke structure $\mathbf{K} = (P, W, R, w_0, L)$, by Theorem 2 we first get a BAT $\mathcal{D}_{\mathbf{K}}$ corresponding to $\mathbf{K}$. We then introduce the notation $\phi[s]$ to denote the situation calculus formula obtained from a given expression ϕ by restoring the situation argument s in all the fluents occurring in ϕ. Finally, we view a CTL formula $(Op\ \phi)[s]$ as a macro defined in the situation calculus as follows:

$$p[s] =_{df} \bigvee_{w_i \in W_p} state_i(s), \text{ where } p \text{ is an atomic proposition and } W_p = \{w \mid \models_w p\},$$
$$(\neg\phi)[s] =_{df} \neg\, \phi[s]$$
$$(\phi_1 \wedge \phi_2)[s] =_{df} \phi_1[s] \wedge \phi_2[s],$$
$$EX\phi[s] =_{df} (\exists a).Poss(a, s) \wedge \phi[do(a, s)],$$
$$A(\psi_1 U \psi_2)[s] =_{df} (\forall s').succ^*(s, s') \wedge \psi_2[s'] \supset (\forall s'').s \sqsubseteq s'' \sqsubset s' \supset \psi_1[s''],$$
$$E(\psi_1 U \psi_2)[s] =_{df} (\exists s').succ^*(s, s') \wedge \psi_2[s'] \wedge (\forall s'').s \sqsubseteq s'' \sqsubset s' \supset \psi_1[s''].$$

Here, $succ^*(s, s')$ is defined as follows:

$$succ^*(s, s') =_{df} s \sqsubseteq s' \wedge executable(s'),$$

meaning that s' is a subsequent situation of s and s' is executable. Further operators are defined in terms of those above:

$$(\phi_1 \vee \phi_2)[s] =_{df} \neg(\neg\phi_1 \wedge \neg\phi_2)[s],$$
$$(\phi_1 \supset \phi_2)[s] =_{df} (\neg\phi_1 \vee \phi_2)[s],$$
$$AX\phi[s] =_{df} (\neg EX\neg\phi)[s],$$
$$EF\phi[s] =_{df} E(\mathtt{true} U \phi)[s],$$
$$AF\phi[s] =_{df} A(\mathtt{true} U \phi)[s],$$
$$EG\phi[s] =_{df} (\neg AF\neg\phi)[s],$$
$$AG\phi[s] =_{df} (\neg EF\neg\phi)[s].$$

Usually, the semantics of CTL is given in terms of situations (states) and behavior (paths) formulas (Clarke & Sistla 1986). We can introduce this distinction here by viewing the situations in the situation formulas as "snapshots" of the world and those in behavior formulas as "histories". For later convenience, given CTL formula ϕ we will always denote the corresponding semantic formula of ϕ at any situation s as $Q_\phi(s)$.

Checking Properties

Above, we have defined the semantics of CTL formulas in terms of a translation of these formulas into formulas of a decidable fragment of the situation calculus. Now, we define the model checking task in terms of a logical entailment. Given a Kripke structure $\mathbf{K} = (P, W, R, w_0, L)$ and a CTL formula ϕ, we first construct a BAT $\mathcal{D}_{\mathbf{K}}$ using the algorithm in the proof of Theorem 2. We then construct a situation calculus formula $Q_\phi(s)$ corresponding to ϕ using the method described in Subsection "Semantics". All these constructions are done in polynomial time and yield axioms and formulas that are polynomial in the size of both $\mathbf{K}$ and ϕ. Now model checking the system $\mathbf{K}$ against the property ϕ for initial state w_0 amounts to establishing the entailment

$$\mathcal{D}_{\mathbf{K}} \models Q_\phi(S_0).$$

In (De Giacomo, Ternovskaia, & Reiter 1997), dynamic properties of reactive systems are expressed by using the transition semantics and second order formulas expressing least and greatest fix-point properties. Here, following (Clarke, Grumberg, & Peled 1999), we specify properties by using transition relation R of the Kripke structure representing a reactive system and second order formulas expressing least and greatest fix-point properties. We use the following theorem from (Clarke, Grumberg, & Peled 1999) reformulated in the situation calculus to that end:

Theorem 3 *Suppose that K is a Kripke structure and that we identify each CTL formula ϕ with the set $\{s \mid K, s \models \phi\} \subseteq 2^{S_K}$. Then each of the basic CTL operators may be characterized as a least or greatest fix-point of an appropriate predicate transformer in the following way:*

$$EF\phi[s] \equiv \mu_Z.[\phi[s] \vee EX(Z)[s]],$$
$$AF\phi[s] \equiv \mu_Z.[\phi[s] \vee AX(Z)[s]],$$
$$EG\phi[s] \equiv \nu_Z.[\phi[s] \wedge EX(Z)[s]],$$
$$AG\phi[s] \equiv \nu_Z.[\phi[s] \wedge AX(Z)[s]],$$
$$A(\phi_1 U \phi_2)[s] \equiv \mu_Z.[\phi_2[s] \vee \phi_1[s] \wedge AX(Z)[s]],$$
$$E(\phi_1 U \phi_2)[s] \equiv \mu_Z.[\phi_2[s] \vee \phi_1[s] \wedge EX(Z)[s]].$$

An Example

Now, we effectively construct a BAT from the Kripke structure of Figure 1 representing the RW system.

Actions: $tr_{0,1}, tr_{0,2}, tr_{1,3}, tr_{1,4}, tr_{2,4}, tr_{2,5}, tr_{3,0}, tr_{3,6}, tr_{4,7}, tr_{5,0}, tr_{6,2}, tr_{7,1}$.

Fluents: $T_1(s), T_2(s), N_1(s), N_2(s), C_1(s), C_2(s), state_0(s), state_1(s), state_2(s), state_3(s), state_4(s), state_5(s), state_6(s), state_7(s)$.

Initial database:

$$N_1(S_0) \wedge N_2(S_0) \wedge \neg T_1(S_0) \wedge \neg T_2(S_0) \wedge \neg C_1(S_0) \wedge \neg C_2(S_0) \wedge trans(0,1) \wedge trans(0,2) \wedge trans(1,3) \wedge trans(1,4) \wedge trans(2,4) \wedge trans(2,5) \wedge trans(3,0) \wedge trans(3,6) \wedge trans(4,7) \wedge trans(5,0) \wedge trans(5,7) \wedge trans(6,2) \wedge trans(7,1) \wedge state_0(S_0) \wedge \neg state_1(S_0) \wedge \neg state_2(S_0) \wedge \neg state_3(S_0) \wedge \neg state_4(S_0) \wedge \neg state_5(S_0) \wedge \neg state_6(S_0) \wedge \neg state_7(S_0).$$

Action precondition axioms:

$$Poss(tr_{0,1}, s) \equiv trans(0,1) \wedge state_0(s),$$
$$Poss(tr_{0,2}, s) \equiv trans(0,2) \wedge state_0(s),$$
$$Poss(tr_{1,3}, s) \equiv trans(1,3) \wedge state_1(s),$$
$$Poss(tr_{1,4}, s) \equiv trans(1,4) \wedge state_1(s),$$
$$Poss(tr_{2,4}, s) \equiv trans(2,4) \wedge state_2(s),$$
$$Poss(tr_{2,5}, s) \equiv trans(2,5) \wedge state_2(s),$$
$$Poss(tr_{3,0}, s) \equiv trans(3,0) \wedge state_3(s),$$
$$Poss(tr_{3,6}, s) \equiv trans(3,6) \wedge state_3(s),$$
$$Poss(tr_{4,7}, s) \equiv trans(4,7) \wedge state_4(s),$$
$$Poss(tr_{5,0}, s) \equiv trans(5,0) \wedge state_5(s),$$
$$Poss(tr_{6,2}, s) \equiv trans(6,2) \wedge state_6(s),$$
$$Poss(tr_{7,1}, s) \equiv trans(7,1) \wedge state_7(s).$$

Successor state axioms:

$$state_0(do(a,s)) \equiv a = tr_{3,0} \vee a = tr_{5,0} \vee state_0(s) \wedge a \neq tr_{0,1} \wedge a \neq tr_{0,2},$$
$$state_1(do(a,s)) \equiv a = tr_{0,1} \vee a = tr_{7,1} \vee state_1(s) \wedge a \neq tr_{1,3} \wedge a \neq tr_{1,4},$$
$$state_2(do(a,s)) \equiv a = tr_{0,2} \vee a = tr_{6,2} \vee state_2(s) \wedge a \neq tr_{2,4} \wedge a \neq tr_{2,5},$$
$$state_3(do(a,s)) \equiv a = tr_{1,3} \vee state_3(s) \wedge a \neq tr_{3,0} \wedge a \neq tr_{3,6},$$
$$state_4(do(a,s)) \equiv a = tr_{1,4} \vee a = tr_{2,4} \vee state_4(s) \wedge a \neq tr_{4,7},$$
$$state_5(do(a,s)) \equiv a = tr_{2,5} \vee state_5(s) \wedge a \neq tr_{5,0} \wedge a \neq tr_{5,7},$$
$$state_6(do(a,s)) \equiv a = tr_{3,6} \vee state_6(s) \wedge a \neq tr_{6,2},$$
$$state_7(do(a,s)) \equiv a = tr_{4,7} \wedge a = tr_{5,7} \vee state_7(s) \wedge a \neq tr_{7,1}.$$

Abbreviations:

$$T_1(s) \equiv state_1(s) \vee state_4(s) \vee state_7(s),$$
$$T_2(s) \equiv state_2(s) \vee state_4(s) \vee state_6(s),$$
$$N_1(s) \equiv state_0(s) \vee state_2(s) \vee state_5(s),$$
$$N_2(s) \equiv state_0(s) \vee state_1(s) \vee state_3(s),$$
$$C_1(s) \equiv state_3(s) \vee state_6(s),$$
$$C_2(s) \equiv state_5(s) \vee state_7(s).$$

Simulation

To generate finite sequences of actions of the given concurrent system $\mathcal{D}_{\mathbf{K}}$ and check if property $Q_\phi(S_0)$ is satisfied , we solve the following deduction task:

$$\mathcal{D}_{\mathbf{K}} \models (\exists s).Do(execActions(N), S_0, s) \wedge Q_\phi(S_0),$$

where N is a constant natural number and $execActions(N)$ is the GOLOG procedure defined in the section on GOLOG.

To generate non-terminating sequences of actions and check if $\mathcal{D}_{\mathbf{K}} \models Q_\phi(S_0)$, we solve the following deduction task:

$$\mathcal{D}_{\mathbf{K}} \models checkCTL(\phi),$$

where abbreviation $checkCTL(\phi)$ represents a sentence obtained by replacing all the predicate $succ^*(s, s')$ in $Q_\phi(S_0)$ by

$$\exists \delta. Trans^*(execActions, s, \delta, s'),$$

and *execActions* is the following GOLOG procedure that infinitely generates transitions of the system at random.

proc *execActions*
  while *true* $(\pi\ a)[Poss(a)?;\ a]$; **endWhile**
endProc .

$Trans^*(execActions, s, \delta, s')$ represents the execution of (non-terminating) GOLOG program *execActions* starting from situation s and getting to situation s' with program δ remained. The detailed semantics of $Trans^*$ is given in (De Giacomo, Ternovskaia, & Reiter 1997). For example, to check CTL formula whether $A(\psi_1 U \psi_2)$ holds for concurrent system $\mathbf{K}$, we are to solve whether

$$\mathcal{D}_{\mathbf{K}} \models (\forall s).(\exists \delta).Trans^*(execActions, S_0, \delta, s) \wedge \psi_2[s] \supset (\forall s').(s' \sqsubset s \supset \psi_1[s']).$$

Sample Properties

Some simple properties of the RW system expressed in CTL are: $EG(N_2 \supset EX\ N_2)$, $AG(N_2 \supset EF\ C_2)$, $EG(\neg C_1 \wedge \neg C_2)$, $EF(C_1 \wedge C_2)$, etc. By restoring the situation argument s, these properties can be viewed as macros defined in the situation calculus, which still match the intuitive semantics of the CTL formulas. For instance,

$$\begin{aligned}
&(EG(N_2 \supset EX\ N_2))[s] \\
&\quad \equiv (\neg AF(N_2 \wedge \neg EX\ N_2))[s] \\
&\quad \equiv \neg A(\mathtt{true} U(N_2 \wedge \neg EX\ N_2))[s] \\
&\quad \equiv \neg(\forall s').succ^*(s, s') \wedge (N_2 \wedge \neg EX\ N_2)[s'] \\
&\qquad\qquad \supset (\forall s'').s \sqsubseteq s'' \sqsubset s' \supset true[s''] \\
&\quad \equiv (\exists s').succ^*(s, s') \wedge N_2(s') \supset (EX\ N_2)[s] \\
&\quad \equiv (\exists s').succ^*(s, s') \wedge N_2(s') \supset \\
&\qquad\qquad (\exists s'').succ(s', s'') \wedge N_2(s'').
\end{aligned}$$

$$\begin{aligned}
&(AG(N_2 \supset EF\ C_2))[s] \\
&\quad \equiv (\neg EF(N_2 \wedge \neg EF\ C_2))[s] \\
&\quad \equiv (\neg E(trueU(N_2 \wedge \neg EF\ C_2))[s] \\
&\quad \equiv \neg(\exists s').succ^*(s, s') \wedge N_2(s') \wedge (\neg EF\ C_2)[s'] \\
&\quad \equiv (\forall s').succ^*(s, s') \wedge N_2(s') \supset (E(\mathtt{true} UC_2))[s'] \\
&\quad \equiv (\forall s').succ^*(s, s') \wedge N_2(s') \supset \\
&\qquad\qquad (\exists s'').succ^*(s', s'') \wedge C_2(s'').
\end{aligned}$$

Discussion

Ours can be considered as a symbolic model checking approach without BDDs, similar to the approach described in (Biere *et al.* 1999). In fact we show how to reduce model checking to entailment in a decidable subset of the situation calculus. An early work heading in this direction is reported in (Rajan, Shankar, & Srivas 1995); unfortunately, lack of technical detail does not allow a comparison with our approach.

Notice that (Reiter 2001) gives an implementation technique for BATs such as the one of Section "An Example". This technique justifies a straightforward translation of the BATs to a form suitable for a Prolog implementation. This technique could be applied here. This would amount to implementing a predicate, e.g. $checkCTL(\phi)$, where ϕ is a CTL formula, for checking whether the BAT entails $Q_\phi(S_0)$. The same technique could be used to simulate the system modeled by the BAT. Since CTL properties involve the predicate $\sqsubset$, any interpreter for checking these properties will necessarily be non-Markovian (Gabaldon 2002) meaning that effects of actions are explained by taking into account all past situations. All this however remains to be accounted for.

A perceived advantage of our framework is the richness of the situation calculus which is more expressive than branching time temporal logic (Pinto 1994). A systematic study of fragments richer than the one considered in this paper remains to be undertaken. It remains also to see how our framework can be turned into a practical tool using well-known automata theoretic semantics for the situation calculus in the style of (Vardi & Wolper 1986).

The fragment ${\mathcal{L}_0}^0$ is powerful enough to express relatively realistic systems. In general however, we can define fragments ${\mathcal{L}_j}^i$, with increasing indexes i and j. What is the exact expressive power of ${\mathcal{L}_0}^0$? What do we gain in expressive power with the fragments ${\mathcal{L}_j}^i,\ i = 1, 2, \cdots,\ j = 1, 2, \cdots$? All these questions are worth pursuing.

References

Biere, A.; Cimatti, A.; Clarke, E.; and Zhu, Y. 1999. Symbolic model checking without bdds. In *Proceedings of TACAS/ETAPS'99*, 193–207. Berlin: Springer Verlag.

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

Clarke, E.; Grumberg, O.; and Peled, D. 1999. *Model Checking*. Cambridge, MA: MIT Press.

De Giacomo, G.; Ternovskaia, E.; and Reiter, R. 1997. Non-terminating processes in the situation calculus. *AAAI'97 Workshop*.

Emerson, E., and Trefler, R. 1999. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In Pierre, L., and Kropf, T., eds., *Correct Hardware Design and Verification Methods. Proceedings of the 10th IFIP WG*, 142–156. Springer Verlag. LNC 1703.

Gabaldon, A. 2002. Non-markovian control in the situation calculus. In *Eighteenth national conference on Artificial intelligence*, 519–524. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Harel, D.; Tiuryn, J.; and Kozen, D. 2000. *Dynamic Logic*. Cambridge, MA, USA: MIT Press.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12(10):576–580.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. *J. of Logic Programming, Special Issue on Actions* 31(1-3):59–83.

McCarthy, J. 1963. Situations, actions and causal laws. Technical report, Stanford University.

Pinto, J. 1994. *Temporal Reasoning in the Situation Calculus*. Ph.D. Dissertation, Department of Computer Science, University of Toronto, Toronto.

Pirri, F., and Reiter, R. 1999. Some contributions to the metatheory of the situation calculus. *Journal of the ACM* 46(3):325–364.

Plotkin, G. 1981. A structural approach to operational semantics, tr-daimi-fn 19. Technical report, Comp. Science Dpt., Aarhus University.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on foundations of Computer Science*, 46–57. IEEE Computer Society.

Pnueli, A. 1981. A temporal logic of concurrent programs. *Theoretical Computer Science* 13:45–60.

Rajan, S.; Shankar, N.; and Srivas, M. 1995. An integration of model checking with automated proof checking. In Wolper, P., ed., *Proceedings of the 7th International Conference on Computer Aided Verification*, 84–97. Springer Verlag. LNC 939.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. Cambridge: MIT Press.

Ternovskaia, E. 1999. Automata theory for reasoning about actions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 153–158.

Vardi, M., and Wolper, P. 1986. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press.

Wolper, P. 1998. The algorithmic verification of reactive systems. 1998 francqui chair lectures given at the fundp (namur). Lecture Notes, http://www.montefiore.ulg.ac.be/~pw/cours/.