Automata-theoretic Model Checking and Situation Calculus

Yilan Gu
Dept. of computer Science
University of Toronto
Toronto, ON M5S 3G4

February 28, 2001

Abstract

In this paper, we studied the relationship between one kind of approach of model checking – automata-theoretic approach and one kind of powerful description language – situation calculus which can be used to model dynamic systomes. By developing basic action theory for any given B*ü*chi automaton, we merge automata-theoretic approach into situation calculus and GOLOG, we did a few observations of the benefit and reason for doing so.

Keywords: basic action theory, situation calculus, GOLOG, B*ü*chi automata, model checking, linear-time temporal logic

1 Introduction

During last decade, several action formalisms have been developed ([9, 3, 7]) to focilitate describing dynamic systems, such as planning problems, actions of robotics (cf. [10]), etc. Situation calculus is one of the outstanding powerful languages. The research on the applications of situation calculus and further implementations now is a very active area in artificial intelligence.

On the other hand, with the development of hardware and software, the systems (including concurrent systems) become more and more complicated. So, researchers become more and more concerned with how to ensure the correctness and safety of various kinds of the systems – the reason why we are interested in automated verifivation. There are mainly two kinds of verification methods: model checking and theorem proving. Automata-theoretic appoach is one of model-checking methods.

During the study of action theory and automated varification, although they are in different branches of computer scientce, it seems that these two branches of studies have lots of connection. We would like to try to develop some relationship between them, to allow to either use the results in verfications and formal methods to help study action and dynamic systems, or to use the approaches or tools developed in the reseach of artificial intelligence to exploit new approaches of automated verfication.

The article is arranged as follows: in section 2, we take a look at situation calculus and GOLOG; in section 3, we reviewed the approach of automata-theoretic model checking together with LTL; in section 4, we give out our main result – embedding Buchi automata into basic action theory and GOLOG, hence get the conclusion in section 5.

2 The Situation Calculus and GOLOG

2.1 The Language of The Situation Calculus

The situation calculus is a second order language specifically designed for representing dynamically changing world. It was first proposed by John McCarthy in 1963(cf. [6]). Since dealing with a different problem, we may use a slightly different description of the language of situation calculus described below.

The language of situation calculus $\mathcal{L}_{sc}$ that we adopt here is from [10]. It is a three sorted language with equality. The three disjoint sorts of $\mathcal{L}_{sc}$ are:

- *action:* a first-order term representing actions in dynamic world, such as `jump` (the action of jumping), $\mathtt{kick}(x)$ (kicking object x), and $\mathtt{put}(r, x, y)$ (robot r putting object x on top of object y), etc. The constant and function symbols for actions are completely application-dependent.

- *situation:* a first-order term which denotes possible world histories. A distinguished constant S_0 and function symbol do are used. S_0 denotes the *initial situation*, before any action has been performed; $do(a, s)$ denotes the situation that results from performing action a in situation s.

- *object:* a catch-all sort representing for everything else depending on the domain of application, such as `ball, Mary`, etc.

In fact, every situation corresponds to a finite sequence of actions; for example, the initial situation S_0 corresponds to empty sequence of actions, the situation $do(\mathtt{pickup}(x), do(\mathtt{drop}(y), do(\mathtt{pickup}(y), S_0)))$ corresponds to the action sequence $\mathtt{drop}(y)$,$\mathtt{pickup}(x)$ from the beginning, etc. We shall use binary relation $s \sqsubset s'$ to represent that s is a proper sub-history of s', and $s \sqsubseteq s'$ is equivalent to $s = s' \vee s \sqsubset s'$.

Another important term in situation calculus is $fluent$. $fluents$ are predicates and functions whose values may vary from situation to situation, used to describe what holds in a situation. By convention, the last argument of a fluent is a situation. For example, the fluent $\mathtt{Holding}(r, x, s)$ might stand for the relation of robot r holding object x in situation s.

The logical symbols of the language are $\neg, \wedge, \exists$. Other connectives and the universal quantifier are the usual abbreviations.

Finally, a distinguished predicate $Poss(a, s)$ is used to state that the action a can be performed in situation s. For example, $Poss(\mathtt{pickup}(r, x), S_0)$ says that the robot is able to pick up object x in the initial situation.

This completes the specification of the language $\mathcal{L}_{sc}$. For later convenience, an abbreviation is introduced as follows:

Abbreviation 2.1

- $do([], s) \stackrel{def}{=} s$;

- $do([a_1, a_2, \cdots, a_n], s) \stackrel{def}{=} do(a_n, do(\cdots, do(a_1, s) \cdots))$.

Definition 2.2 A log K is a list of actions $[a_1, a_2, \dots]$.

Notice that there is a one-to-one correspondence between a log beginning at initial time and situation, whenever this log is finite or infinite.

Remark 2.3 Later we will also need the following functional fluent defined recursively:

- $len(S_0) = 0$;
- $len(do(a, s)) = len(s) + 1$ for any action a.

2.2 The Basic Action Theory

A *basic action theory* is a set of axioms represented in situation calculus to model the actions and their effects in a given dynamic system $\mathcal{D}$ together with *functional fluent consistency property.* Hereby we just give out a summary; the detailed explanation could be found in [8].

The set $\mathcal{D} = \mathcal{D}_f \bigcup \mathcal{D}_{ap} \bigcup \mathcal{D}_{ss} \bigcup \mathcal{D}_{una} \bigcup \mathcal{D}_{S_0}$ consists of following axioms:

- Fundamental axioms, denoted as $\mathcal{D}_f$.
  There are four axioms included in $\mathcal{D}_f$:

  $$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2$$
  $$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s)$$
  $$\neg s \sqsubset S_0$$
  $$s \sqsubset do(a, s') \equiv s \sqsubseteq s'$$

  Since the fundamental axioms is so mechanical, we will not write them out during description, but assume them to be true for any basic action theory.

- Action precondition axioms, denoted as $\mathcal{D}_{ap}$.
  For each action function(could be 0-ary) A, there is one axiom of the form:

  $$Poss(A(x_1, \cdots x_n), s) \equiv \Pi_A(x_1, \cdots x_n, s),$$

  where $\Pi_A(x_1, \cdots x_n, s)$ is a formula uniform in s. This axiom characterizes the preconditions of performing action A in the current situation s.

- Successor state axiom, denoted as $\mathcal{D}_{ss}$.
  A successor state axiom for an $(n+1)$-ary relational fluent F is a sentence of $\mathcal{L}_{sc}$ of the form:

  $$F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F((x_1, \ldots, x_n, a, s),$$

  where $\Phi_F((x_1, \ldots, x_n, a, s)$ is a formula uniform is s.
  A successor state axiom for an $(n+1)$-ary functional fluent F is a sentence of $\mathcal{L}_{sc}$ of the form:

  $$f(x_1, \ldots, x_n, do(a, s)) = y \equiv \phi_f((x_1, \ldots, x_n, y, a, s),$$

  where $\Phi_F((x_1, \ldots, x_n, y, a, s)$ is a formula uniform is s.

  The successor state axiom for fluent F completely characterizes the value of fluent F in the successor resulting from performing action a in situation s.

- Unique name axioms, denoted as $\mathcal{D}_{una}$.
  For any n-ary action A, $A(x_1, \cdots, x_n) = A(y_1, \cdots, y_n) \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n$; for any distinct actions A and B, $A(\vec{x}) \neq B(\vec{y})$.

  Since the unique name axioms is so mechanical, we will not write them out during description, but assume them to be true for any basic action theory.

- Initial database, denoted as $\mathcal{D}_{S_0}$.
  It is a set of first order sentences in which S_0 is the only situation sort (i.e., uniform in S_0). No sentence of $\mathcal{D}_{S_0}$ quantifies over situations, or mentions $Poss$, $<$ or the function symbol do. Notice that the initial database may contain sentences mentioning no situation term at all, for example, unique names for individuals, or "timeless" facts like $dog(x) \supset mammal(x)$.

The *functional fluent consistency property* is as follows:
Suppose f is a functional fluent whose successor state axiom in $\mathcal{D}_{ss}$ is

$$f(\vec{x}, do(a,s)) = y \equiv \phi_f((\vec{x}, y, a, s),$$

then

$$\mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models (\forall \vec{x}).(\exists y)\phi_f((\vec{x}, y, a, s) \wedge [(\forall y, y').\phi_f((\vec{x}, y, a, s) \wedge \phi_f((\vec{x}, y', a, s)] \supset y = y'.$$

The reason of requiring this consistency property is that it provides a sufficient condition for preventing a source of inconsistency in f's successor state axiom.

2.3 Complex actions, Procedures and GOLOG

So far, in our treatment of the situation calculus, we only dealt with primitive actions, with effects and preconditions independent of each other. In this section we will introduce a kind of compositional treatment of the frame problem for complex actions, i.e., the actions that have other actions as components. This results in a novel kind of programming language – GOLOG (cf. [10]).

To handle complex actions, it is sufficient to show that for each complex action A we care about, there is a formula in the situation calculus, which we call $Do(A, s, s')$, says that action A, when started in situation s, can terminate legally in situation s'.

Definition 2.4 The Do formula is defined recursively as follows:

- For any primitive action A,

$$Do(A, s, s') \stackrel{def}{=} Poss(A, s) \wedge s' = do(A, s);$$

- For the sequential composition of complex action A and B,

$$Do([A; B]s, s') \stackrel{def}{=} (\exists s'').Do(A, s, s'') \wedge Do(B, s'', s');$$

- For a conditional involving a test ϕ which is a formula of the situation calculus with a free variable s of the form $[if\ \phi\ then\ A\ else\ B]$,

$$Do([if\ \phi\ then\ A\ else\ B], s, s') \stackrel{def}{=} [\phi(s) \wedge Do(A, s, s')] \vee [\neg\phi(s) \wedge Do(B, s, s')];$$

- For a test action, $[\phi?]$, determine if a condition ϕ currently holds,

$$Do([\phi?], s, s') \stackrel{def}{=} \phi(s) \wedge s' = s;$$

- For a non-deterministic branch to action A or action B, $[A|B]$,

$$Do([A|B], s, s') \stackrel{def}{=} Do(A, s, s') \vee Do(B, s, s');$$

- For a non-deterministic choice of a value for variable x, $[\pi x.A]$,

$$Do([\pi x.A], s, s') \stackrel{def}{=} (\exists x).Do(A, s, s');$$

- For an iteration of the form, $[while\ \phi\ do\ A]$,

$$Do([while\ \phi\ do\ A], s, s') \stackrel{def}{=} (\forall P).\{(\forall s_1)[\neg\phi(s_1) \supset P(s_1, s_1)] \wedge$$

$$(\forall s_1, s_2, s_3)[\phi(s_1) \wedge Do(A, s_1, s_2) \wedge P(s_2, s_3) \supset P(s_1, s_3)]\} \supset P(s, s').$$

Then, the definition of complex actions could be extended to include procedures:

Suppose $P(t_1, t_2, \ldots, t_n)$ is a procedure call, taking pair of situation arguments, then

$$Do(P(t_1, \ldots, t_n), s, s') \stackrel{def}{=} P(t_1[s], \ldots, t_n[s], s, s')$$

means that executing the procedure P on actual parameters $t_1, \ldots, t_n$ causes a transition from situation s to s'.

Because of the definition of complex actions together with procedures, we then can deal with nondeterministic, conditional, or concurrent operations(detailed explanation and examples are in [10]).

Definition 2.5 A *GOLOG program* is of form:
$\texttt{proc}\ P_1(\vec{v_1})\delta_1\ \texttt{endproc};\cdots;\ \texttt{proc}\ P_n(\vec{v_n})\delta_n\ \texttt{endproc};\ \delta_0$

where P_i is declaration of procedures with formal parameter $\vec{v_i}$ and procedure body δ_i for each $i(1 \le i \le n)$, δ_0 is the main program body. $\delta_0, \ldots, \delta_n$ are complex actions extended by actions for procedure calls.

The semantics of a program is

$$Do(\{\texttt{proc}\ P_1(\vec{v_1})\delta_1\ \texttt{endproc};\cdots;\ \texttt{proc}\ P_n(\vec{v_n})\delta_n\ \texttt{endproc};\ \delta_0\}, s, s') \stackrel{def}{=}$$
$$(\forall P_1, \ldots, P_n).[\textstyle\bigwedge_{i=1}^{n}(\forall s_1, s_2, \vec{v_i}).Do(\delta, s_1, s_2) \supset P_i(\vec{v_i}, s_1, s_2)] \supset Do(\delta_0, s, s'),$$

i.e. when $P_1, \ldots, P_n$ are the smallest binary relations on situations that are closed under the evaluation of their procedure bodies $\delta_1, \ldots, \delta_n$, then any transition (s, s') obtained by evaluating the main program δ_0 is a GOLOG transition for the evaluation for the program.

The program and the complex actions defined above can be viewed as a programming language whose semantics is refined via macro-expension into sentences of the situation calculus. We call this language GOLOG. GOLOG appears to offer significant advantages over current tolls for applications in dynamic domains like the high-level programming of robots and software agents, process control, discrete event simulation, complex database transactions, etc. (cf. [4, 1])

2.4 A Simple Example

To see how the situation calculus and GOLOG would work, let's look at a simple example.

Robotics domain involves three primitive actions: $pickup(x)$ (picking up a block x), $putonfloor(x)$ (putting a block x on the floor), and $putontable(x)$ (putting a block x on the table); and three fluents: $Holding(x, s)$ (the robot is holding a block x at situation s), $Onfloor(x, s)$ (a block x is on the floor at situation s), and $OnTable(x)$ (a block x is on the table at situation s).

The precondition axioms are the following:

- $Poss(pickup(x), s) \equiv (\forall z).\neg Holding(z, s)$;
- $Poss(putonfloor(x), s) \equiv Holding(x, s)$;
- $Poss(putontable(x), s) \equiv Holding(x, s)$.

The successor state axioms are the following:

- $Holding(x, do(a, s)) \equiv a = pickup(x) \vee Holding(x, s) \wedge a \neq putonfloor(x) \wedge a \neq putontable(x)$;
- $OnFloor(x, do(a, s)) \equiv a = putonfloor(x) \vee Onfloor(x, s) \wedge a \neq pickup(x)$;
- $OnTable(x, do(a, s)) \equiv a = putontable(x) \vee OnTable(x, s) \wedge a \neq pickup(x)$.

We might also have the following facts about the initial situation:

- $\neg Holding(x, S_0)$;
- $OnTable(x, S_0) \equiv (x = b_1) \vee (x = b_2)$.

So initially the robot is not holding anything, and b_1 and b_2 are the only blocks on the table. Finally, we can consider two complex actions, removing a block, and clearing the table:

- `proc` $RemoveBlock(x)$: $[pickup(x); putonfloor(x)]$ `endproc`;
- `proc` $ClearTable$: *while* $(\exists x).Ontable(x)$ *do* $\pi x[OnTable(x)?; RemoveBlock(x)]$.

This completes the specification of the example.

3 Automata-theoretic Model Checking

With the inevitable growth in scale and functionality of hardware and software systems, the likelihood of subtle errors become greater. Some of these errors may cause catastrophic loss of time, money, even human life. A major goal of software engineering is to enable developers to construct systems that operate reliably despite the growth of complexity. One way of achieving this goal is by using *formal methods*, which are mathematically-based languages, techniques, and tools for specifying and verifying such systems. That's why we are so interested in automated verification.

Two well-established approaches to verification are model checking and theorem proving. We here are interested in one important approach in model checking – automata-theoretic approach together with linear-time temporal logic (LTL). This method is a theoretical base for an important verification tool – SPIN. In this approach, the specification is given as an automaton; then the system, also modeled as an automaton, is compared to the specification to determine whether or not its behavior conforms to that of the specification.

First of all, let's have a review on Büchi automata and linear-time temporal logic.

Definition 3.1 A `Büchi automaton` is a tuple $\mathcal{A} = (\Sigma, Q, \rho, q_0, F)$, where

- Σ is a finite alphabet;
- Q is a finite set of states;
- $\rho : Q \times \Sigma :\rightarrow 2^Q$ is a nondeterministic transition function;

- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is a set of final states.

For any given Büchi Automaton $\mathcal{A} = (\Sigma, Q, \rho, q_0, F)$, a **run** of $\mathcal{A}$ over infinite word $\alpha = a_0 a_1 \ldots$ from Σ^ω is a sequence $\delta_0, \delta_1, \ldots$ where $\delta_0 = q_0$ and $\delta_i \in Q$ for every i, $\rho(\delta_i, a_i) = \delta_{i+1}$. A run is called `successful` if some state of F occurs infinitely often in it.

- $\mathcal{A}$ accepts α if there is a successful run of α.
- $L(\mathcal{A}) = \{\alpha | \mathcal{A}\ accepts\ \alpha\}$.
- If $L = L(\mathcal{A})$ for some Büchi Automaton $\mathcal{A}$, L is said to be Büchi-recognizable.

Note that Büchi automata are closed under implementation and intersection. Moreover, the emptiness problem for Büchi automata is decidable.

The specifications is represented in a formal way – formulas in form of linear-time temporal logic (LTL).

Definition 3.2

- If p is an atomic propositional formula ($Prop$ denotes the set of all propositional formulas), it is a formula in LTL;
- If p and q are LTL formulas, so are $p \wedge q$, $p \vee q$, $\neg p$, $p\mathcal{U}q$, $\circ p$ (next), $\diamond p$ (eventually) ,$\Box p$ (always).

The interpretation for these formulas are over computations $\pi : \omega \to 2^{Prop}$ which assign truth values to the elements of $Prop$ at each time instant:

- $\pi, i \models p$ for $p \in Prop$ iff $p \in \pi(i)$;
- $\pi, i \models \circ p$ iff $\pi, i+1 \models p$;
- $\pi, i \models p\mathcal{U}q$ iff for some $j \geq i$, $\pi, j \models q$ and for all $k, i \leq k < j$, $\pi, k \models p$ (strong until);
- $\pi, i \models \Box p$ iff for all $j \leq i$, $\pi, j \models p$;
- $\pi, i \models \Box p$ iff there exists $j \leq i$, $\pi, j \models p$.

Linear-time temporal logic is good for expressing safety and liveness properties. It was shown by the work of Wolper, Vardi and Sistla in 1983 that given an LTL formula ϕ, one can build a Buchi automaton $\phi = (\Sigma, Q, \rho, s_0, F)$, s.t. $L(\mathcal{A}_\phi)$ is exactly the set of computations satisfying the formula ϕ(cf. [12]).

Vardi and Wolper (cf. [13]) showed how the temporal-logic model checking problem could be recast in terms of automata, thus related the approach of temporal model checking (developed by Clarke and Emerson [2]) with the approach of automata-theoretic based approach. Given a finite-state program $P = (W, s_0, R, V)$ where W is finite set of states, $s_0 \in W$ is the initial states, $R \subseteq W^2$ s total accessibility relation and $V : W \to 2^{Prop}$ which assigns truth values to propositions in $Prop$ for each state in W, we can represent it as a Buchi automaton $\mathcal{A}_P = (2^{Prop}, W, \rho, s_0, W)$ where $s' \in \rho(s, a)$ iff $(s, s') \in R$ and $a = V(s)$. Verifying that the program P satisfies the specification ϕ

is equivalent to checking that language $L' = L(\mathcal{A}_P) \bigcap \overline{(L(\mathcal{A}_\phi))}$ is empty. In fact we can construct a Büchi automaton, which recongize language L', from $\mathcal{A}_P$ and $\mathcal{A}_\phi$ in polynomial time.

Until now we've reviewed the automata-theoretic approach of model checking. We like Büchi automata because they are clear and direct for the model-builder, also have powerful ability of description (for example, drawing a diagram).

Now we are going to relate automata and LTL with situation calculus and GOLOG.

4 Embedding Automata-Theoretic Model Checking into Situation Calculus and GOLOG

Now given any Büchi automaton $\mathcal{A} = (\Sigma, Q, \rho, q_0, F)$, we are going to imbed this automaton into situation calculus and GOLOG.

Theorem 4.1 Given any Büchi automaton $\mathcal{A} = (\Sigma, Q, \rho, q_0, F)$, there exists a corresponding basic action theory together with its GOLOG program, denoted as KB, and a goal G, there is 1-1 onto correspondence between infinite word w over Σ and log K_w which can be generated by KB, such that $w \in L(\mathcal{A})$ if and only if $KB \models G(K_w)$.

Proof: For any given Buchi automaton $\mathcal{A} = (\Sigma, Q, \rho, q_0, F)$, Suppose $\sigma = \{t_1, t_2, \ldots, t_n\}$, $Q = \{q_0, q_1, \ldots, q_m\}$. Let's construct the corresponding basic action theory $\mathcal{D}$ as follows:

- **Actions:**
  For every $t_i \in \Sigma$, introduce a_i be an action in KB;

- **Fluent:**
  For every $q_i \in Q$, introduce $q'_i(s)$ be a relationalfluent in KB;
  $len(s)$ is a functional fluent defined in section 2;
  When $F = \emptyset$, we also need to introduce a extra special fluent $Loop(s)$;

- **Fundamental axioms:**
  ...omitted

- **unique name axioms:**
  ...omitted

- **Initial database:**
  $r_{i,j}(0 \leq i, j \leq m)$ is a non-fluent predicate on actions such that for every $k(1 \leq k \leq n)$, $r_{i,j}(a_k)$ is $true$ if and only if $\rho(q_i, t_k) = q_j$;
  $q'_0(S_0)$; $\neg q'_i(S_0)$ for $1 \leq i \leq m$.

- **Action precondition axioms:**
  For every action a, we have axiom

$$Poss(a, s) \equiv \bigvee_{i=0}^{m} (\bigvee_{j=0}^{m} r_{i,j}(a) \wedge q_i(s));$$

- **Successor state axiom:**
  For every fluent q'_i, we have successor state axiom:

$q_i'(do(a,s)) \equiv \bigvee_{k=1}^{n}(a = a_k \wedge (\bigvee_{t=0}^{m} r_{t,i}(a_k))) \vee (q_i'(s) \wedge (\bigwedge_{k=1}^{n}(a \neq a_k \vee (\bigwedge_{t=0}^{m} \neg r_{t,i}(a_k)))))$;
$len(do(a,s)) = n+1 \equiv len(s) = n$
If there is fluent $Loop(s)$, then its successor state axiom:
$Loop(do(a,s)) \equiv Loop(s)$.

- **procedure:**

  proc *generateActions*

  while *true*

  $(\pi a)[Poss(a)?; a]$;

  endwhile

  endproc.

  here *generateActions* is a procedure that infinitely generates sequence of actions at random.

We also introduce another abbriviation $accept(s) \equiv \bigvee_{q_i \in F} q_i'(s)$ (special case: when $F = \emptyset$, let $accept(s) \equiv Loop(s)$), then give out the goal $G(K_w)$ as follows:

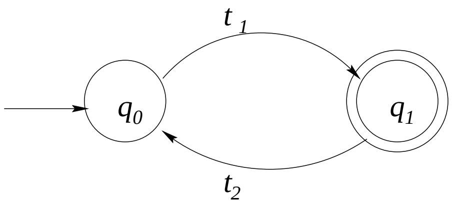
$$(\exists s).Do(K_w, S_0, s) \wedge \exists n \forall s'(len(s') \geq n \wedge s' \sqsubseteq s \supset \exists s''(s'' \sqsubseteq s' \wedge accept(s''))).$$

For any infinite word w over Σ, there is a corresponding log K_w such that $w \in L(\mathcal{A})$ if and only if $KB \models G(K_w)$.

In fact, it is a 1-1 onto correspondence between infinite word $w = t_{i_1} t_{i_2} \dots t_{i_n} \dots$ over Σ and log $K_w = [a_{i_1} a_{i_2} \dots a_{i_n} \dots]$ which clearly can be generated by *generateActions* according to the construction of above KB, and satisfying $w \in L(\mathcal{A})$ if and only if $KB \models G(K_w)$.
$\square$

Let's look at a simple example.

Example 4.2 Given a Buchi automaton $\mathcal{A} = \{\Sigma, Q, \rho, q_0, F\}$ where $\Sigma = \{t_1, t_2\}$, $Q = \{q_0, q_1\}$, $F = \{q_1\}$ and ρ is defined as $\rho(q_0, t_1) = \{q_1\}, \rho(q_0, t_2) = \emptyset, \rho(q_1, t_1) = \emptyset, \rho(q_1, t_2) = \{q_0\}$.



Now we could give out the corresponding KB and goal G in situation calculus as follows:

- **Actions:**
  For $t_1 \in \Sigma$, let *turnOn* be an action in KB;
  For $t_2 \in \Sigma$, let *turnOff* be an action in KB;

- **Fluent:**
  For $q_0 \in Q$, let $Dark(s)$ be a fluent in KB;
  For $q_1 \in Q$, let $Bright(s)$ be a fluent in KB;

- **Fundamental axioms:**
  ...omitted

- **Unique name axioms**
  ...omitted

- **Initial database**
  $r_{0,1}(turnOn); r_{1,0}(turnOff);$
  $Dark(S_0); \neg Bright(S_0);$

- **Action precondition axioms:**

  Substituting the corresponding actions and predicates, after simplifying by using the facts in initial database, we obtain:
  $Poss(turnOn, s) \equiv Dark(s);$
  $Poss(turnOff, s) \equiv Bright(s);$

- **Successor state axiom:**

  Substituting the corresponding actions and predicates, after simplifying by using the facts in initial database, we obtain:
  $Dark(do(a, s)) \equiv a = turnOff \vee (Dark(s) \wedge a \neq turnOff);$
  $Bright(do(a, s)) \equiv a = turnOn \vee (Dark(s) \wedge a \neq turnOn);$

- **procedure:**

  ```
  proc generateActions
      while true
          (πa)[Poss(a)?; a];
      endwhile
  endproc.
  ```

  Here *generateActions* is a procedure that generates an infinite sequence of actions at random.

Let $accept(s) \equiv Bright(s)$, then $G(x)$ is defined as $(\exists s).Do(x, S_0, s) \wedge \exists n \forall s'(len(s') \geq n \wedge s' \sqsubseteq s \supset \exists s''(s'' \sqsubseteq s' \wedge accept(s''))$.

As we've seen in the above section, for any given Bϋchi automaton model $\mathcal{A}$ and LTL specification ϕ, we can obtain a Bϋchi automaton $\mathcal{A}'$ in polynomial-time from $\mathcal{A}$ and $\mathcal{A}_\phi$ such that $L(\mathcal{A}') = L(\mathcal{A}) \bigcap \overline{(\mathcal{A}_\phi)}$. To check that automaton $\mathcal{A}$ satisfies property ϕ is equivalent to checking whether $L(\mathcal{A}') = \emptyset$. If we use the theory of actions, it is equivalent to developing a basic action theory together with GOLOG program system KB and a goal $G(x)$, such that there is no log x satisfying $KB \models G(x)$. Notice that for $Do(x, S_0, s')$, the log x and s' have 1-1 onto correspondence, i.e. x in fact, is not a free variable in G, therefore, if there is a situation s consistent in KB satisfying $Do(x, S_0, s) \wedge \exists n \forall s'(len(s') \geq n \wedge s' \sqsubseteq s \supset \exists s''(s'' \sqsubseteq s' \wedge accept(s''))$, there certainly is a x s.t. $KB \models G(x)$. So our target is reduced to checking if $KB \bigcup \{G\}$ is inconsistent. Hence we have following corollary.

Corollary 4.3 The automata-theoretic model-checking problem can be reduced to a problem in the situation calculus and GOLOG.

Until now we've seen the very closed relation between automata-theoretic model-checking and situation calculus.

5 Related Research and Future Work

In fact, the closed relationship between automated verification and action theoty has been noticed by several researchers. One distinguish work has been done by E. Ternovskaia(cf. [11]), which shown the decidability of a rather expressive fragment of the situation calculus by translating an arbitrary formula in this fragment into tree automaton. Another nice research now under processing by I. Kiringa is going to merge classical model-checking of Clarke and Emerson into situation calculus(cf. [5]).

In this report, we embedded the B*ü*chi automata into situation calculus and GOLOG, and therefore get the result that the automata-theoretic model checking can be simulated by situation calculus and GOLOG. On one hand, we could say that situation calculus and GOLOG in AI may could be extended to be another tool for automated verification. On the other hand, since automata and LTL provide a very clear and intuitive way for modeling (such as giving out diagrams in automata models) and specification, lots of dynamic systems, such as reactive systems, can be modeled as B*ü*chi automata, therefore helping us construct in a formal easier way their basic action theory, such as giving out precondition axioms and successor axioms, and GOLOG program.

Our future work may focus on studying the direct relation between linear-time temporal logic and situation calculus, and the complexity of the transaction.

References

[1] A.Bonner and M.Kifer. "Transaction Logic Programming", *Technical Report, Dept. of Computer Science, Univ. of Toronto*, 1993.

[2] E.M.Clarke,E.A.Emerson,A.P.Sistla, "Automatic Verification of Finite-State Consurrent Systems Using Temporal Logics Specifications A Practical Approach", *Proc. ACM Symp. on Principles of Programming Languages, Austim*, pp.117-126,1983.

[3] M.Gelfond and V.Lifschitz, "Representing Action and Change by Logic Programs", *J. of Logic Programming*, Vol.17, pp.301-322, 1993.

[4] G.de Giacomo, Y.Lespérance, and H.J. Levesque. "Reasoning About Concurrent Executing, Prioritized Interrupts, and Exogenous Actions In The Situation Calculus". *Proc. of the 15th Int. Joint Conf. on AI* pp.1221-1226 ,Nagova,Japan,1997.

[5] I.Kiringa, "Model Checking in the Situation Calculus", under processing, 2001.

[6] John McCarthy, "Situations, Actions, and Causal Laws", *Stanford University Artificial Intelligence Project*, 1963.

[7] R.Miller and M.Shanahan, "Narrative in the Situation Calculus", *J. of Logic and computation (special Issue on Actions and Processes)*, Vol.4(5), pp.513-530, 1994.

[8] F.Pirri and R. Reiter, "Some Contributions to the Meta-theory of the Situation Calculus", *J. of the ACM*, 1999.

[9] R. Reiter, "The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and A Completeness Result for Goal Regression", *Artificial Intelligence and Mathematical Theory of Computing: Papers in Honor of John McCarchy*, pp.359-280, 1991.

[10] R. Reiter, "Knowledge in Action", book draft, 2000.

[11] E. Ternovskaia, "Automata Theory for Reasoning about Actions", Proc. IJCAI'99,1999.

[12] P.Wolper, M.Y.Vardi and A.P. Sistla, "Reasoning about Infinite Computation Paths", *Proc. 24th IEEE Symp. on Foundations of Computer Science*, pp. 185-194, 1983.

[13] M.Y.Vardi and P.Wolper, "An Automata-theoretic Approach to Automatic Program Verification", *Proc.of Logic in Computer Science*, 1986.