A Situation Calculus Based Approach for Model Checking

Yilan Gu Iluju Kiringa
University of Toronto University of Ottawa

Presented by: Yilan Gu. April, 2006
1. Motivations
2. Preliminaries
3. Translating Concurrent Systems into Basic Action Theories
4. Representing Model Checking properties in the Situation Calculus
5. Conclusions and Future Work
Motivation

- Two ways of reasoning about properties of reactive program
  1. Operational approach
     (model checking, in particular)
  2. Deductive approach
     (the situation calculus, in particular)
- Merge both approaches into one framework
- Show that the situation calculus has more expressive power
- Intend to explore a different way of doing model checking
An Example: RW Concurrent System

- Two processes: Reader (\# 1), Writer (\# 2)
- Three states: Non-Trying $N_i$, Trying $T_i$, Critical $C_i$
- Transitions: 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: Reader-Writer transition system
Kripke structures and Behaviors

Definition 1 (Kripke structure)

\( K = (P, W, R, w_0, L) \) 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, and
- \( L : W \to 2^P \) maps each \( w \in W \) to set \( \{ p \in P | \models_w p \} \).

RW system represented as a Kripke structure:

- \( P = \{ N_1, N_2, T_1, T_2, C_1, C_2 \} \);
- \( W = \{ w_0, \ldots, w_7 \} \);
- \( R = \{ (w_0, w_1), (w_0, w_2), (w_1, w_3), (w_1, w_4), (w_2, w_4), (w_2, w_5), (w_3, w_0), (w_3, w_6), (w_4, w_7), (w_5, w_0), (w_5, w_7), (w_6, w_2), (w_7, w_1) \} \);

Initial state \( w_0 \);
- \( L(w_0) = \{ N_1, N_2 \} \), \( L(w_1) = \{ T_1, N_2 \} \),
- \( L(w_2) = \{ N_1, T_2 \} \), \( L(w_3) = \{ C_1, N_2 \} \),
- \( L(w_4) = \{ T_1, T_2 \} \), \( L(w_5) = \{ N_1, C_2 \} \),
- \( L(w_6) = \{ C_1, T_2 \} \), \( L(w_7) = \{ T_1, C_2 \} \).
**Definition 2** (Behavior)

Let $K = (P, W, R, w_0, L)$ be a Kripke structure. Then a behavior $\sigma$ of $K$ is a function from $N$ to $W$ such that:

- $N = \{0, 1, \ldots, n\}$ for some natural number $n$,
- 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 of natural numbers, then $\sigma$ is an *infinite* behavior.

**Definition 3** (Computational Tree)

Suppose $K = (P, W, R, w_0, L)$ is a Kripke structure. Then the (infinite) computational tree $CT_K$ of $K$ is the set $\{\sigma_1, \sigma_2, \ldots\}$ of all (infinite) behaviors of $K$.

□ One may unwind a Kripke structure into an (infinite) computational tree that is rooted in $w_0$ (see Figure 2 top diagram).
The computational tree:

The canonical structure:

**Figure 2:** Computation tree and canonical structure of the RW system
The Situation Calculus

- Actions: \( \text{pickup}(x) \)
- Situations: \( S_0, \text{do}(a, s) \)
- Objects: \( \text{Tom}, \text{etc} \)
- Fluents: \( \text{ontable}(x, s) \)

The basic action theory (BAT) \( \mathcal{D} \):

- Action precondition axioms:
  \[
Poss(\text{pickup}(x), s) \equiv \neg \exists y.\text{holding}(y, s)
  \]
- Successor state axioms:
  \[
  \text{ontable}(x, \text{do}(a, s)) \equiv a = \text{putdown}(x) \lor
  \]
  \[
  \text{ontable}(x, s) \land a \neq \text{pickup}(x)
  \]
- Initial database:
  \[
  \neg \exists y.\text{holding}(y, S_0), \text{ontable}(\text{Box}, S_0)
  \]

Golog program – sequences of complex actions

\[
\textbf{proc} \ \text{execActions}
\]
\[
\textbf{while} \ \text{true} \ (\pi a)[\text{Poss}(a)?; a] \ \textbf{endWhile}
\]
\[
\textbf{endProc}
\]

\( \square \) A decidable fragment of the SitCalc \( \mathcal{L}_0^0 \):

1) Action functions with no arguments;
2) Fluents have only one argument of sort situation.
Translating Kripke Structures into BATs

- Define the *canonical structure* for any given BAT: a subtree obtained from the tree of situations by pruning away non-executable paths according to the given BAT.

- An example of canonical structures: see the 2nd diagram in Figure 2.

- Translate any given Kripke structure into a BAT so that the computational tree of the Kripke structure is represented as a canonical structure.

**Theorem** Suppose $K = (P, W, R, w_0, L)$ is a Kripke structure. Then one can effectively construct a BAT $D_K$ of language $L^0_0$ whose canonical structure $M$ is obtained from the computational tree $CT_K$ of $K$ such that

$$K \text{ has } CT_K \text{ iff } \models_M D_K.$$
• General translation approach (e.g., RW system)

**Actions:** Introduce $tr_{i,j}$ for each $(w_i, w_j) \in R$

E.g., $tr_{0,1}, tr_{0,2}, tr_{1,3}$, etc.

**Fluents:** Introduce $p(s)$ for each $p \in P$, and $state_i(s)$ for each $w_i \in W$

E.g., $state_i(s) \ (i = 0..7)$, $T_j(s), N_j(s), C_j(s) \ (j = 1..2)$.

$\mathcal{D}_K = \mathcal{D}_\Sigma \cup \mathcal{D}_{una} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{S_0}$

$\mathcal{D}_{S_0}$: Introduce axiom $state_0(S_0)$,

axioms $\neg state_i(S_0)$ for all $i \neq 0$,

and a fact $tr(i, j)$ for each $(w_i, w_j) \in R$

$\mathcal{D}_{ap}$: For each $tr_{i,j}$,

Poss$(tr_{i,j}, s) \equiv trans(i, j) \land state_{w_i}(s)$.

$\mathcal{D}_{ss}$: For each $state_i \ (0 \leq i < |W|)$, and $p \in P$,

$state_i(do(a, s)) \equiv \bigvee_{j=1}^{|W|} a = tr_{j,i} \lor$

$state_i(s) \land \bigwedge_{j=1}^{|W|} a \neq tr_{i,j}$;

$p(s) \equiv \bigvee_{w_i \in \{w \mid \models_p\}} state_i(s)$.

E.g., $state_0(do(a, s)) \equiv a = tr_{3,0} \lor a = tr_{5,0} \lor$

$state_0(s) \land a \neq tr_{0,1} \land a \neq tr_{0,2}$

$T_1(s) \equiv state_1(s) \lor state_4(s) \lor state_7(s)$
Model Checking: CTL Formulas

- Computational Tree Logic (CTL) formulas: expresses a branching time logic by extending linear-time temporal logic with behavior quantifiers. For example, in the RW system,

\[ EG(N_2 \supset EX N_2) \] (there is a behavior s.t. at all of its future states it holds that if process 2 is at non-trying section at current state then it is still at non-trying section at some of its next state),

\[ AG(N_2 \supset EF C_2) \] (for all behaviors and at all of their future states it holds that if process 2 is at non-trying section at current state then process 2 will be at critical section at some of its future state),

\[ EG(\neg C_1 \land \neg C_2) \] (there is a behavior s.t. at all of its future states it holds that neither process 1 nor process 2 are at critical section at current state),

\[ EF(C_1 \land C_2) \] (there is a behavior s.t. at some of its future states it holds that both process 1 and process 2 are at critical section at current state),

etc.
• Represent CTL semantically using the SitCalc:

\[ p[s] = df \bigvee_{w_i \in \{w \models p\}} state_i(s), \text{ where } p \text{ is an atomic,} \]
\[ (\neg \phi)[s] = df \neg \phi[s], \quad (\phi_1 \land \phi_2)[s] = df \phi_1[s] \land \phi_2[s], \]
\[ EX\phi[s] = df (\exists s').\text{succ}(s, s') \land \phi[s'], \]
\[ A(\psi_1 U \psi_2)[s] = df (\forall s').\text{succ}^*(s, s') \land \psi_2[s'] \supset \]
\[ (\forall s'').s \subseteq s'' \sqsubseteq s' \supset \psi_1[s''], \]
\[ E(\psi_1 U \psi_2)[s] = df (\exists s').\text{succ}^*(s, s') \land \psi_2[s'] \land \]
\[ (\forall s'').s \subseteq s'' \sqsubseteq s' \supset \psi_1[s'']. \]

Here, \( \text{succ}(s, s') \) is defined as follows:
\[ \text{succ}(s, s') = df (\exists a).Poss(a, s) \land s' = do(a, s), \]
and \( \text{succ}^* \) denotes the transitive closure of \( \text{succ} \).

Further operators are defined in terms of those above:
\[ (\phi_1 \lor \phi_2)[s] = df \neg (\neg \phi_1 \land \neg \phi_2)[s], \]
\[ (\phi_1 \supset \phi_2)[s] = df (\neg \phi_1 \lor \phi_2)[s], \]
\[ AX\phi[s] = df (\neg EX\neg \phi)[s], \]
\[ EF\phi[s] = df E(true U \phi)[s], \]
\[ AF\phi[s] = df A(true U \phi)[s], \]
\[ EG\phi[s] = df (\neg AF\neg \phi)[s], \]
\[ AG\phi[s] = df (\neg EF\neg \phi)[s]. \]
• RW system: examples of CTL formula properties represented using the SitCalc:

\[(EG(N_2 \supset EX N_2))[s]\]
\[\equiv (\neg AF(N_2 \land \neg EX N_2))[s]\]
\[\equiv \neg A(trueU(N_2 \land \neg EX N_2))[s]\]
\[\equiv \neg (\forall s').suc^{*}(s, s') \land (N_2 \land \neg EX N_2)[s']\]
\[\quad \supset (\forall s'').s \sqsubseteq s'' \sqsubseteq s' \supset true[s'']\]
\[\equiv (\exists s').suc^{*}(s, s') \land N_2(s') \supset (EX N_2)[s]\]
\[\equiv (\exists s').suc^{*}(s, s') \land N_2(s') \supset (\exists s'').suc(s', s'') \land N_2(s'').\]

\[(AG(N_2 \supset EF C_2))[s]\]
\[\equiv (\neg EF(N_2 \land \neg EF C_2))[s]\]
\[\equiv (\neg E(trueU(N_2 \land \neg EF C_2))[s]\]
\[\equiv \neg (\exists s').suc^{*}(s, s') \land N_2(s') \land (\neg EF C_2)[s']\]
\[\equiv (\forall s').suc^{*}(s, s') \land N_2(s') \supset (E(trueUC_2))[s']\]
\[\equiv (\forall s').suc^{*}(s, s') \land N_2(s') \supset (\exists s'').suc^{*}(s', s'') \land C_2(s'').\]
Checking Properties and Simulation

• Checking properties: Consider a Kripke structure $\mathbf{K} = (P, W, R, w_0, L)$ and a CTL formula $\phi$.

  1. Construct a BAT $\mathcal{D}_K$;
  2. Construct a SitCalc formula $Q_\phi(s)$ corresponding to $\phi$.

Complexity (time and size): polynomial

$$(\mathbf{K}, w_0) \models \phi \iff \mathcal{D}_K \models Q_\phi(S_0).$$
• Simulation

1. Generate finite sequences of actions:

   proc execActions(n)
   
   n = 0? |
   
   n > 0? ; (π a)[Poss(a)? ; a];
   
   execActions(n – 1)
   
   endProc .

   \[ \mathcal{D} \models (\exists s).Do(execActions(N), S_0, s) \land Q_\phi(S_0) \]

2. Generate sequences of actions for non-terminating programs:

   Given \( Q_\phi(S_0) \), replace \( \text{succ}^*(s, s') \) with \( (\exists \delta)\text{Trans}^*(execActions, s, \delta, s') \) [GTR97] and obtain \( Q_\phi(S_0)' \). Then,

   \[ \mathcal{D} \models Q_\phi(S_0)' \]
Discussion

• A symbolic model checking approach without BDDs

• Reduce model checking to entailment in a decidable subset of the SitCalc

• Further implementation required

• Look for other possibly decidable fragments $\mathcal{L}_{i,j}$ other than $i = j = 0$