Automata-Theoretic Model Checking of Reactive Systems

Radu Iosif
Verimag/CNRS (Grenoble, France)

Thanks to Tom Henzinger (IST, Austria), Barbara Jobstmann (CNRS, Grenoble) and Doron Peled (Bar-Ilan University, Israel)
Ensuring Correctness of Hw/Sw Systems

• Uses logic to specify correctness properties, e.g.:
  – the program never crashes
  – the program always terminates
  – every request to the server is eventually answered
  – the output of the tree balancing function is a tree, provided the input is also a tree ...

• Given a logical specification, we can do either:
  – VERIFICATION: prove that a given system satisfies the specification
  – SYNTHESIS: build a system that satisfies the specification
Approaches to Verification

- **THEOREM PROVING**: reduce the verification problem to the satisfiability of a logical formula (entailment) and invoke an off-the-shelf theorem prover to solve the latter
  - Floyd-Hoare checking of pre-, post-conditions and invariants
  - Certification and Proof-Carrying Code

- **MODEL CHECKING**: enumerate the states of the system and check that the transition system satisfies the property
  - explicit-state model checking (SPIN)
  - symbolic model checking (SMV)

- **COMBINED METHODS**:
  - static analysis (ASTREE)
  - predicate abstraction (SLAM, BLAST)
MODEL CHECKING REAL-LIFE SYSTEMS

• MODEL EXTRACTION:
  – give precise semantics (meaning) to what the system does and how it does it
  – the result is a (possibly infinite) directed graph in which the nodes denote states and the edges denote transitions
  – the model is an abstraction of the original system, i.e. it has more behaviors

• MODEL VERIFICATION:
  1. check whether the model satisfies a given property
  2. if no error was found, stop and report OK
  3. otherwise, check if the error is feasible in the original system
     – if yes, report ERROR
     – otherwise, refine the abstraction, by excluding the spurious behavior and goto 1
Safety vs. Liveness

- **Safety**: *something bad never happens*

  A counterexample is an *finite* execution leading to something bad happening

  *Example*: the program does not dereference any null pointers

- **Liveness**: *something good eventually happens*

  A counterexample is an *infinite* execution on which nothing good happens

  *Example*: the function terminates on any given input
Modeling Systems
**Systems Dichotomies**

- Deterministic/Non-deterministic
- Sequential/Concurrent
  - synchronous/asynchronous communication between processes
- Hardware/Software/Embedded
  - Hw is always finite-state (boolean data)
  - Sw is considered infinite (integers, recursive data structures, etc.)
- Transformational/Reactive
  - a **transformational** system takes input, computes output and stops
  - a **reactive** system interacts continuously with the environment
Problems in Systems Modeling

• Representing states
  – local/global components

• granularity of actions
  – what are the atomic transitions?

• Representing concurrency
  – one transition at a time
  – coinciding transitions
Modeling States

- $V = \{x_0, x_1, x_2 \ldots\}$ is a set of variables ranging over some domain (bool,int,...)

- $\varphi(x_0, x_1, \ldots)$ is a parameterized assertion over $V$ e.g.,
  $x_0 < 10$, $x_1 \leq x_2 + x_3, \ldots$

- A state is an assignment of values to the variables e.g.,
  $s(x_0) = 2$, $s(x_1) = 3$, $s(x_2) = 5, \ldots$

- $s \models \varphi$ iff $\varphi$ is true under $s$
Atomic Transitions

• An atomic transition is a small piece of code such that no smaller piece of code is observable

• Question: is $x ← x + 1$ observable?

• Answer1: yes, if $x$ is a register and the transition is executed using an inc machine command

• Answer2: yes, if $x$ is variable local to a process, which is not visible to other processes

```c
int a = 0;
```

P1: load R1, a   P2: load R2, a
    inc R1     inc R2
    store R1, a store R2, a
Modeling Atomic Transitions

- Each transition $G \rightarrow A$ has two parts:
  - the guard $G$: the enabling condition
  - the action $A$: a multiple assignment
  - the guard and action are executed in one atomic step

- **Example**: $x > y \rightarrow x' = y \land y' = x$

- **Frame rule**: if a variable $v'$ does not appear in $A$ then implicitly $v' = v$
Initial Conditions

- $V = \{x_0, x_1, x_2, \ldots\}$ are program variables
- The initial condition is an assumption $\psi(x_0, x_1, \ldots)$
- The program can start in any state $s$ such that $s \models \psi$
- Example: $x = 0, x > 0, \ldots$
Sequential Systems

- \( V = \{x_0, x_1, x_2, \ldots\} \)
- \( P = \langle V, T, I \rangle \), where
  - \( T \) is a set of transitions \( G \rightarrow A \) involving \( V \)
  - \( I \) is an initial condition over \( I \)
- **Example:**
  \[
  P = \langle \{x, y\}, \{\text{True} \rightarrow x' = x + y, y > 0 \rightarrow y' = y - 1\}, x = 0 \land y > 0 \rangle
  \]
- **State space:**
  
  \[
  \begin{array}{cccc}
  (0, 3) & (3, 2) & (5, 1) & (6, 0) \\
  (0, 4) & (4, 3) & (7, 2) & (9, 1) & (10, 0) \\
  (0, 5) & (5, 4) & (9, 3) & (12, 2) & (14, 1) & (15, 0) \\
  \end{array}
  \]
  
  \[
  \ldots
  \]
Concurrent Systems: the interleaving model

- \( S = \langle P_1, P_2, \ldots, P_n \rangle \), where \( P_i = \langle V_i, T_i, I_i \rangle \), \( i = 1, \ldots, n \)
- \( V = \bigcap_{i=1}^{n} V_i \) are called \textit{global} variables
- \( L_i = V_i \setminus V \) are called \textit{local} variables
- An \textit{execution} is a possibly infinite sequence of states \( s_0, s_1, s_2, \ldots \) such that:
  - \( s_0 \models I_1 \land \ldots \land I_n \)
  - for each \( i = 0, 1, 2, \ldots \) there exists \( j \in \{1, \ldots, n\} \) and \( G \rightarrow A \in T_j \) such that \( s_i \models G \) and \( s_i, s_{i+1} \models A \) (\( s_i \) is the valuation of unprimed and \( s_{i+1} \) the valuation of primed variables)
  - i.e., exactly one process is executed at the time
  - the \textit{frame rule} applies to that specific process
Mutual Exclusion Example

- $P_i = \langle \{m, x, l_i\}, \{t_1^i, t_2^i, t_3^i\}, m = 0 \land x = 0 \land l_i = 0 \rangle$, for $i = 1, 2$ where
  
  $t_1^i : l_i = 0 \land m = 0 \rightarrow l_i' = 1 \land m' = 1$
  
  $t_2^i : l_i = 1 \land m = 1 \rightarrow l_i' = 2 \land m' = 0 \land x' = x + 1$
  
  $t_3^i : l_i = 2 \rightarrow l_i' = 0$

- A possible execution:

  $$(m, x, l_1, l_2) : (0, 0, 0, 0) \xrightarrow{1} (1, 0, 1, 0) \xrightarrow{1} (0, 1, 2, 0) \xrightarrow{2} (1, 1, 2, 1) \xrightarrow{2} (0, 2, 2, 2)$$
Mutual Exclusion Example

- **No deadlock**: in every state there is at least one enabled transition.
- **Mutex**: there is at most one process in the critical section at any time.
- **No starvation**: if a process attempts to enter the critical section, then eventually it will enter.
- **Future problem**: the state space is infinite!
Fairness

Global assumptions on the process scheduler:

- **Weak process fairness**: if some process is enabled continuously from some state, then it will be executed

- **Weak transition fairness**: if some transition is enabled continuously from some state, then it will be executed

- **Strong process fairness**: if some process is enabled infinitely often, then it will be executed

- **Strong transition fairness**: if some transition is enabled infinitely often, then it will be executed
Fairness Example

\[ x = 0 \land y = 0 \land z = 0 \land l_1 = 0 \land l_2 = 0 \]

P1::= 0: x'=1  

P2::= 0: while y=0 do
    1: z'=z+1
    []
    2: if x=1 then y'=1

*Does* \( P_1 \) *terminate? Does* \( P_2 \) *terminate?*

- **No fairness**: nothing guaranteed
- **Weak fairness**: \( P_1 \) terminates
- **Strong process fairness**: \( P_1 \) terminates
- **Strong transition fairness**: both \( P_1 \) and \( P_2 \) terminate
Linear Temporal Logic
Reasoning about infinite sequences of states

- Linear Temporal Logic is interpreted on infinite sequences of states
- Each state in the sequence gives an interpretation to the atomic propositions
- Temporal operators indicate in which states a formula should be interpreted

Example 1  Consider the sequence of states:

\[ \{p, q\} \{\neg p, \neg q\} (\{\neg p, q\} \{p, q\})^\omega \]

Starting from position 2, \( q \) holds forever. \( \Box \)
Kripke Structures

Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a finite alphabet of atomic propositions.

A Kripke structure is a tuple $K = \langle S, s_0, \rightarrow, L \rangle$ where:

- $S$ is a set of states,
- $s_0 \in S$ a designated initial state,
- $\rightarrow : S \times S$ is a transition relation,
- $L : S \rightarrow 2^\mathcal{P}$ is a labeling function.
Paths in Kripke Structures

A *path* in $K$ is an *infinite* sequence $\pi : s_0, s_1, s_2 \ldots$ such that, for all $i \geq 0$, we have $s_i \rightarrow s_{i+1}$.

By $\pi(i)$ we denote the $i$-th state on the path.

By $\pi_i$ we denote the *suffix* $s_i, s_{i+1}, s_{i+2} \ldots$.

\[
\text{inf}(\pi) = \{ s \in S \mid s \text{ appears infinitely often on } \pi \}
\]

If $S$ is *finite* and $\pi$ is *infinite*, then $\text{inf}(\pi) \neq \emptyset$. 
Linear Temporal Logic: Syntax

The alphabet of LTL is composed of:

- atomic proposition symbols \( p, q, r, \ldots \),
- boolean connectives \( \neg, \lor, \land, \rightarrow, \leftrightarrow \),
- temporal connectives \( 
\Box, \Diamond, U, R \).

The set of LTL formulae is defined inductively, as follows:

- any atomic proposition is a formula,
- if \( \varphi \) and \( \psi \) are formulae, then \( \neg \varphi \) and \( \varphi \bullet \psi \), for \( \bullet \in \{ \lor, \land, \rightarrow, \leftrightarrow \} \) are also formulae.
- if \( \varphi \) and \( \psi \) are formulae, then \( \Diamond \varphi, \Box \varphi, \Diamond \psi, \varphi U \psi \) and \( \varphi R \psi \) are formulae,
- nothing else is a formula.
Temporal Operators

- $\circ$ is read *at the next time* (in the next state)
- $\blacksquare$ is read *always in the future* (in all future states)
- $\diamond$ is read *eventually* (in some future state)
- $\mathcal{U}$ is read *until*
- $\mathcal{R}$ is read *releases*
Linear Temporal Logic: Semantics

\[
K, \pi \models p \iff p \in L(\pi(0))
\]
\[
K, \pi \models \neg \varphi \iff K, \pi \not\models \varphi
\]
\[
K, \pi \models \varphi \land \psi \iff K, \pi \models \varphi \text{ and } K, \pi \models \psi
\]
\[
K, \pi \models \Box \varphi \iff K, \pi_1 \models \varphi
\]
\[
K, \pi \models \varphi U \psi \iff \text{there exists } k \in \mathbb{N} \text{ such that } K, \pi_k \models \psi \text{ and } K, \pi_i \models \varphi \text{ for all } 0 \leq i < k
\]

Derived meanings:

\[
K, \pi \models \Diamond \varphi \iff K, \pi \models \top U \varphi
\]
\[
K, \pi \models \Box \varphi \iff K, \pi \models \neg \Diamond \neg \varphi
\]
\[
K, \pi \models \varphi R \psi \iff K, \pi \models \neg (\neg \varphi U \neg \psi)
\]
Examples

- $p$ holds throughout the execution of the system ($p$ is invariant) : $\Box p$
- whenever $p$ holds, $q$ is bound to hold in the future : $\Box (p \rightarrow \Diamond q)$
- $p$ holds infinitely often : $\Box \Diamond p$
- $p$ holds forever starting from a certain point in the future : $\Diamond \Box p$
- $\Box (p \rightarrow \Diamond (\neg q \cup r))$ holds in all sequences such that if $p$ is true in a state, then $q$ remains false from the next state and until the first state where $r$ is true, which must occur.
- $pRq : q$ is true unless this obligation is released by $p$ being true in a previous state.
Concurrent system specification in LTL

- Let $S = \langle P_1, \ldots, P_n \rangle$ be a concurrent system, where $P_i = \langle V_i, T_i, I_i \rangle$
- Absence of deadlock: $\Box \bigvee_{i=1}^{n} enabled(P_i)$
- Weak process fairness: $\bigwedge_{i=1}^{n} \Diamond \Box enabled(P_i) \rightarrow \Diamond execute(P_i)$
- Strong process fairness: $\bigwedge_{i=1}^{n} \Box \Diamond enabled(P_i) \rightarrow \Diamond execute(P_i)$
Conclusion of the first part

- Need a formal language (logic) to express queries about a system’s behavior: deadlock freedom, absence of starvation, fairness conditions, etc.

- The global behavior of a system is modeled as a possibly infinite directed graph, whose nodes are labeled with assertions

- System executions are possibly infinite paths through this graph

- Linear Temporal Logic is a powerful language to express properties of system behaviors