System Specification, Verification and Synthesis (SSVS) – CS 4830/7485, Fall 2019

4: Formal System Modeling: Transition Systems

Stavros Tripakis

Northeastern University
Khoury College of Computer Sciences

Outline

- Transition systems
- Fundamental concepts: reachable states, deadlocks, pre/post, invariants, inductive invariants
- Labeled transition systems and Kripke structures
- Transition systems as the underlying semantics of many formalisms
- Modeling transition systems in Spin/Promela

TRANSITION SYSTEMS

Transition Systems

- The fundamental mathematical model in modern system theory
- A very basic model, capturing the essence of systems
- Reminder: **System** = state + dynamics
- **Transition system** = states + transitions

Transition Systems

- The fundamental mathematical model in modern system theory
- A very basic model, capturing the essence of systems
- Reminder: **System** = state + dynamics
- **Transition system** = states + transitions (+ labels)

Transition Systems: version 1 (without labels)

A tuple:

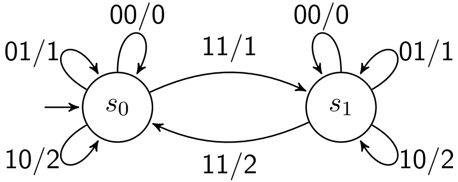
$$(S, S_0, R)$$

- S: set of states (perhaps infinite)
- $S_0 \subseteq S$: set of initial states
- R: transition relation

$$R \subseteq S \times S$$

Example: from an FSM to a transition system

Let's build the transition system for this Mealy machine:



Main idea:

- In an FSM, the next state depends on the input.
- For different inputs, different transitions $\Rightarrow$ non-determinism!

Transition Systems: crucial features

- Sets of states/transitions **possibly infinite**:
  - This allows to model everything! Finite-state systems, infinite-state systems, discrete systems, continuous systems, hybrid systems, ...
- **Non-determinism**:
  - This allows to avoid having to model everything!
  - E.g., omit modeling the inputs (abstraction).

Example: transition system for a digital circuit

Figure from [Baier and Katoen, 2008]:

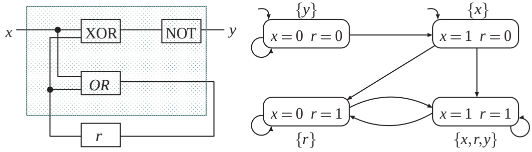


Figure 2.2: Transition system representation of a simple hardware circuit.

How would the transition system look if we didn't include the input x in the state?

Example: transition system for a program

```
var int x;
x := 0
while (x<100) {
  x := x+1;
}
```

Example: transition system for a program

```
var int x;
x := 0
while (x<100) {
  x := x+1;
}
```

What if the loop condition was `true` instead?

Example: transition system for a program

Figure from [Clarke et al., 2018]:

```
Var n:Integer initially n = 10
l0 : while (n > 0) {
l1 :     n = n - 1;
l2 : }
l3 :
```

What if the program was this one?

```
var int x;
x := read_input();
while (x>1) {
  if (x % 2 = 0)
    x := x/2;
  else
    x := 3*x+1;
}
```

What if the program was this one?

```
var int x;
x := read_input();
while (x>1) {
  if (x % 2 = 0)
    x := x/2;
  else
    x := 3*x+1;
}
```

Collatz conjecture: the above program terminates for every `x`.
Open problem in mathematics.
Nevertheless, the transition system is well-defined.

What if we had a real program?

~/private/tools/open-kronos/Matrix

```c
#include "matrix.h"
#include "matrix_hash.h"

MATRIX  guarded_matrix,
        temp_matrix,
        reset_matrix,
        ExtrapoMatrix;

MATRIX  EMPTY_MATRIX;

MATRIX  create_matrix(int dimension)
{
  MATRIX marx;
  int i, j;

  if (dimension <= 0)
    return(NULL);
  marx = (BOUND_TYPE **) malloc(sizeof(BOUND_TYPE *) * dimension);
  for (i=0; i<dimension; i++) {
    marx[i] = (BOUND_TYPE *) malloc(sizeof(BOUND_TYPE) * dimension);
    for (j=0; j<dimension; j++)
      marx[i][j] = 0;
  }
  matrix_dimension(marx) = dimension;
  return(marx);
}

void  destroy_matrix(MATRIX marx)
{
  int i;
  if (is_not_trivial_matrix(marx)) {
    for (i = 0; i < matrix_dimension(marx); i++)
      free(marx[i]);
    free(marx);
  }
}

MATRIX  create_init_extrapolation_matrix()
{
  MATRIX marx;
  int i, j;

  marx = create_matrix(N_OF_CLOCKS);
  for (i=1; i<N_OF_CLOCKS; i++) {
    marx[i][i] = i;
    marx[0][i] = ZERO_LE;
"matrix.c" 756L, 20293C                                   1,0-1         Top
```

Transition systems: some fundamental concepts

Transition system: (S, S_0, R), also called a **state space** (sometimes state space refers only to S).

- We often write $s \to s'$ instead of $(s, s') \in R$.
  Note that the notation $s \to s'$ leaves R implicit.
- s' is a **successor** of s, and s a **predecessor** of s'.
- A (finite or infinite) sequence of states/transitions:
  $s_1 \to s_2 \to s_3 \to \cdots$ is called a **trace**.
- A state s is **reachable** if there exists a trace
  $s_0 \to s_1 \to s_2 \to \cdots \to s$, such that $s_0 \in S_0$.
  Otherwise s is called **unreachable**.

Transition systems: some fundamental concepts

Transition system: (S, S_0, R), also called a **state space** (sometimes state space refers only to S).

- We often write $s \to s'$ instead of $(s, s') \in R$.
  Note that the notation $s \to s'$ leaves R implicit.
- s' is a **successor** of s, and s a **predecessor** of s'.
- A (finite or infinite) sequence of states/transitions:
  $s_1 \to s_2 \to s_3 \to \cdots$ is called a **trace**.
- A state s is **reachable** if there exists a trace
  $s_0 \to s_1 \to s_2 \to \cdots \to s$, such that $s_0 \in S_0$.
  Otherwise s is called **unreachable**.
- Note: every initial state is reachable by definition. Why?

Transition systems: some fundamental concepts

Transition system: (S, S_0, R), also called a **state space** (sometimes state space refers only to S).

- We often write $s \to s'$ instead of $(s, s') \in R$.
  Note that the notation $s \to s'$ leaves R implicit.
- s' is a **successor** of s, and s a **predecessor** of s'.
- A (finite or infinite) sequence of states/transitions:
  $s_1 \to s_2 \to s_3 \to \cdots$ is called a **trace**.
- A state s is **reachable** if there exists a trace
  $s_0 \to s_1 \to s_2 \to \cdots \to s$, such that $s_0 \in S_0$.
  Otherwise s is called **unreachable**.
- Note: every initial state is reachable by definition. Why?
- Quiz: are all states in S reachable?

Transition systems: some fundamental concepts

Transition system: (S, S_0, R).

- A state s is a **deadlock** if $\nexists s' \in S : s \to s'$ (i.e., if s has no successor). Deadlocks are sometimes called **trap states** or just **traps**.
- A **cycle** is a trace $s_1 \to s_2 \to \cdots \to s_n$ such that $s_n = s_1$.
- A **lasso** is a trace $s_1 \to s_2 \to \cdots \to s_n \to s_{n+1} \to \cdots \to s_m$ such that $s_n \to s_{n+1} \to \cdots \to s_m$ is a cycle.

Transition systems: some fundamental concepts

Transition system: (S, S_0, R).

- The transition system is called **deterministic** if every state has at most one successor:

$$\forall s, s', s'' \in S : (s \to s' \land s \to s'') \Rightarrow s' = s''$$

- Otherwise it is called **non-deterministic**.

Transition systems: some fundamental concepts

Transition system: (S, S_0, R).

- The transition system is called **deterministic** if every state has at most one successor:

$$\forall s, s', s'' \in S : (s \to s' \wedge s \to s'') \Rightarrow s' = s''$$

- Otherwise it is called **non-deterministic**.

- **Quiz**: can deadlocks introduce non-determinism?

Transition systems: some fundamental concepts

Transition system: (S, S_0, R).
Let $X \subseteq S$ be a set of states.

- One-step **predecessors**:

$$\mathsf{pre}(X) = \{s \in S \mid \exists s' \in X : s \to s'\}$$

- One-step **successors**:

$$\mathsf{post}(X) = \{s \in S \mid \exists s' \in X : s' \to s\}$$

Transition systems: some fundamental concepts

Transition system: (S, S_0, R).
Let $X \subseteq S$ be a set of states.

- X is an **invariant** if every reachable state is in X
- i.e., an invariant is a superset of the set of reachable states
- X is **inductive** if post$(X) \subseteq X$
- i.e., X is inductive if when starting at any state in X, we cannot leave X
- X is an **inductive invariant** if X is both an invariant and inductive

Transition systems with labels

- Kripke Structures: labels on states
- Labeled Transition Systems: labels on transitions

Kripke Structures

A Kripke structure is a tuple:

$$(\mathsf{AP}, S, S_0, L, R)$$

- AP: set of atomic propositions (modeling state properties)
- S: set of states (perhaps infinite)
- $S_0 \subseteq S$: set of initial states
- L: labeling function on states

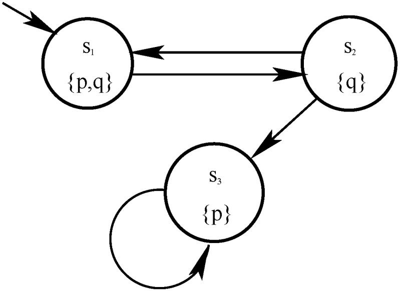
  $$L : S \to 2^{\mathsf{AP}}$$

  2^{AP}: the **powerset** (set of all subsets) of AP.
  For $p \in \mathsf{AP}$ and $s \in S$: "s has property p" iff $p \in L(s)$.
- R: transition relation

  $$R \subseteq S \times S$$

Example: Kripke Structure

In a Kripke structure the labels are on the states. Each state is labeled with a **set of atomic propositions** (those facts that are true when system is in that state).



In the example above, AP $= \{p, q\}$. Atomic proposition p holds at states s_1 and s_3, while atomic proposition q holds at states s_1 and s_2.

Labeled Transition Systems

An LTS is a tuple:

$$(\Sigma, S, S_0, R)$$

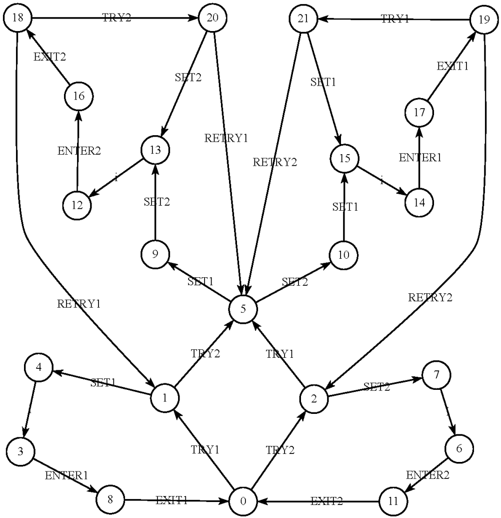
- Σ: set of labels (modeling events, actions, ...)
- S: set of states (perhaps infinite)
- $S_0 \subseteq S$: set of initial states
- R: transition relation

$$R \subseteq S \times (\Sigma \cup \{\epsilon\}) \times S$$

ϵ (sometimes τ): internal, unobservable action (used in composition, simulation/bisimulation equivalences, ...).

Example: LTS

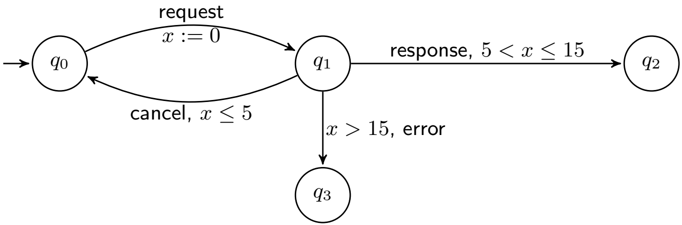
In a LTS the labels are on the transitions:



TRANSITION SYSTEMS: SEMANTIC FOUNDATIONS OF OTHER FORMALISMS

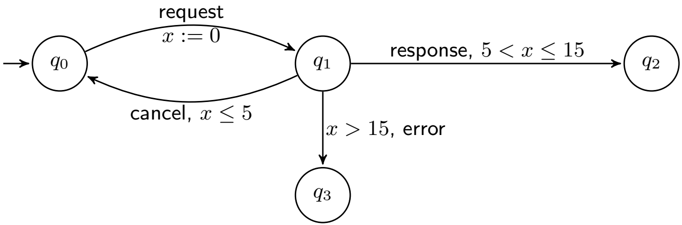
Transition systems of timed automata [Alur and Dill, 1994]

Timed automaton = finite automaton + clocks
Clocks = continuous (real) variables measuring time



Transition systems of timed automata [Alur and Dill, 1994]

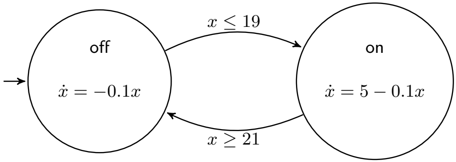
Timed automaton = finite automaton + clocks
Clocks = continuous (real) variables measuring time



Transition system of timed automaton:
States: $\{(q_1, x = 0), (q_1, x = 1), (q_1, x = 0.5), (q_1, x = 0.1), ..., (q_2, x = 0), ...\}$
Transitions: discrete (change q_i) or continuous (change x, time elapses)
State space, transition space: infinite and uncountable!

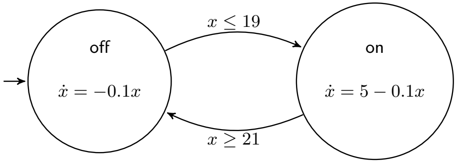
Transition systems of hybrid automata [Alur et al., 1995]

Hybrid automaton = finite automaton + continuous (real) variables
Continuous dynamics: differential equations
Example: thermostat:



Transition systems of hybrid automata [Alur et al., 1995]

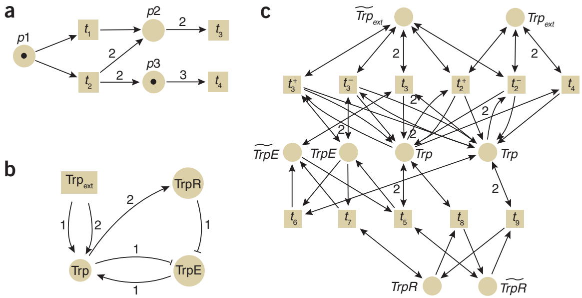
Hybrid automaton = finite automaton + continuous (real) variables
Continuous dynamics: differential equations
Example: thermostat:



Transition system of hybrid automaton:
State space: $\{\text{on, off}\} \times \mathbb{R}$
Transitions: discrete or continuous
Both state and transition spaces: infinite and uncountable!

Transition system of a biological system

Modeling biological networks with **Petri nets**:



(a) A simple, standard Petri net. The circles denote places, whereas the boxes denote transitions. The distribution of tokens (black dots) in the places at a given time defines a marking. Transitions change the marking by removing a token from each incoming arrow and adding a token to each outgoing arrow. (b) Simplified logical regulatory graph for the biosynthesis of tryptophan in E. coli. Each node of the regulatory graph represents an active component: tryptophan (Trp), the active enzyme (TrpE) and the active repressor (TrpR). The node marked by a rectangle accounts for the import of Trp from external medium. All nodes are binary (that is, can take the value 0 or 1), except Trp, which is represented by a ternary variable (taking the values 0, 1, 2). Arrows represent activation and bars denote inhibition. (c) Petri net of the Trp regulatory network. Each of the four components of b is represented by two complementary places and all the different situations that lead to a change of the state of the system are modeled by one of the nine transitions.
Figure and text taken from [Fisher and Henzinger, 2007]. For more info on Petri nets, see [Petri and Reisig, 2008].

Challenge – Extra credit!

Can you think of a system that **cannot** be modeled as a transition system?

Spin

The model-checker Spin

- Widespread explicit-state (enumerative) model checker
- Created by Gerard J. Holzmann at Bell Labs in the 1980s
- Open source, numerous extensions, continuously evolving
- Two books [Holzmann, 1991, Holzmann, 2003], online course
- ACM System Software Award 2001
- Asynchronous systems (mostly)

Modeling discrete transition systems in Spin

```
// a small example spin model
// Peterson's solution to the mutual exclusion problem (1981)

bool turn, flag[2];          // the shared variables, booleans
byte ncrit;                  // nr of procs in critical section

active [2] proctype user()  // two processes
{
    assert(_pid == 0 || _pid == 1);
again:
    flag[_pid] = 1;
    turn = _pid;
    (flag[1 - _pid] == 0 || turn == 1 - _pid);

    ncrit++;
    assert(ncrit == 1);     // critical section
    ncrit--;

    flag[_pid] = 0;
    goto again
}
// analysis:
// $ spin -run peterson.pml
```

Summary

- Transition systems = our first formal model for systems!
  - FSMs can be seen as special cases
  - We will see later that transition systems cannot capture everything we want (e.g., composition, fairness), but OK for now
- Many kinds of TSs: finite/infinite, discrete/continuous, labeled in various ways, ...
- Serve as the semantic foundation of many higher-level formalisms

Notes

- State machines, transition systems, nuXmv models, Spin models, ...: these are not the same like your typical Java or Python programs!
- Why?

Notes

- State machines, transition systems, nuXmv models, Spin models, ...: these are not the same like your typical Java or Python programs!
- Why?
- The former are declarative and formal models of systems: they have formal semantics, and they represent sets or trees of behaviors.
- So what can I do with a model like the one above?
  - Right now: **simulation**! E.g.:
    - Load a model (in nuXmv, Spin, ...)
    - Set the initial state
    - Print the current state
    - Print the set of successor states
    - Choose a successor and move on step forward
    - Repeat
  - You are "unfolding" paths in the transition system of the model: like **debugging** your program with a debugger!
  - Do this also by hand (paper and pencil)! You might get this in homeworks, exams, etc.
  - Later, we will also do: model-checking (verification), synthesis, ...

Bibliography I

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. (1995).
The algorithmic analysis of hybrid systems.
Theoretical Computer Science, 138:3–34.

Alur, R. and Dill, D. (1994).
A theory of timed automata.
Theoretical Computer Science, 126:183–235.

Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
MIT Press.

Clarke, E., Grumberg, O., and Peled, D. (2000).
Model Checking.
MIT Press.

Clarke, E. M., Henzinger, T. A., Veith, H., and Bloem, R. (2018).
Handbook of Model Checking.
Springer.

Fisher, J. and Henzinger, T. A. (2007).
Executable cell biology.
Nature Biotechnology, 25(11).

Bibliography II

Holzmann, G. (1991).
Design and Validation of Computer Protocols.
Prentice Hall.

Holzmann, G. (2003).
The Spin Model Checker.
Addison-Wesley.

Petri, C. A. and Reisig, W. (2008).
Petri net.
Scholarpedia, 3(4).