Verification of Real Time Systems - CS5270
5th lecture

Hugh Anderson

National University of Singapore
School of Computing

February, 2007

A warning...

Outline

1. Administration
   - Assignment 1
   - The road map...
2. State Transition Systems
   - State transition system overview
   - Parallel composition of TS
3. Timed transition systems
   - Timed transition systems overview
   - Parallel composition of TTS
   - Overview of reduction of TTS

Assignment 1

Assignment number 1: Correction

- Hand in next week (Feb **15**) - during lecture

The immediate road map

After completing scheduling, next 2/3 weeks have three topics:

- **TS: State transition systems**
  - some definitions
  - parallel composition
- **TTS: Timed transition systems**
  - formal definition
  - parallel composition

  - Reduction of a TTS (which has possibly infinite states and actions) to a finite TS by quotienting? (takes time)
- **Efficiency in TTS**
  - Regions
  - zones

State transition systems and Automata

What is a state transition system?

- It is an abstract machine used in the study of computation.
- The machine consists of a set of states and transitions between states
- Differs from finite state automata in that state transition systems do not have *accepting* states, and also may have a set of states that is not necessarily finite, or even countable.
- i.e. TS + accepting_states=automata...

Definition seen before

A state transition system ...

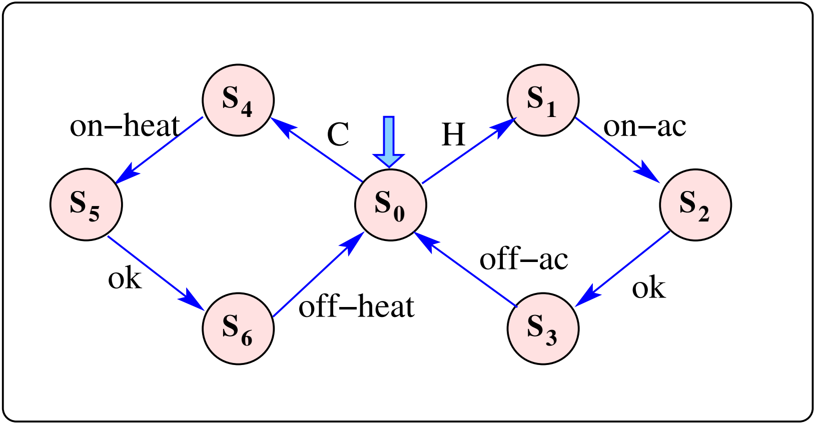
A state transition system TS is a 4-tuple $(S, \mathrm{Act}, \Longrightarrow, S_{\mathrm{in}})$, where

1. S is a set of **states**
2. Act is a set of **actions**
3. $\Longrightarrow : S \times \mathrm{Act} \times S$ is the **transition relation**
4. $S_{\mathrm{in}} \subseteq S$ is the set of **initial states**

Note that S and Act are often *finite* sets, there is often only a single S_{in} state, and the transition relation is often *deterministic* (to be defined soon).

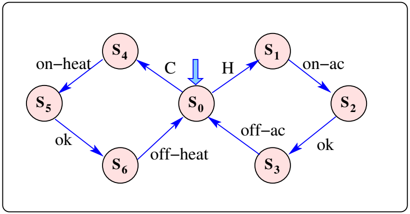
A state transition system

Temperature regulator example



State transition system

Formally...



- $S =$
- $\mathrm{Act} =$
- $\Rightarrow =$
- $S_{\mathrm{in}} =$

Temperature regulator

Controls the heater and air-con unit

The states have been labelled in diagram and we can use the labels to identify valid and invalid traces:

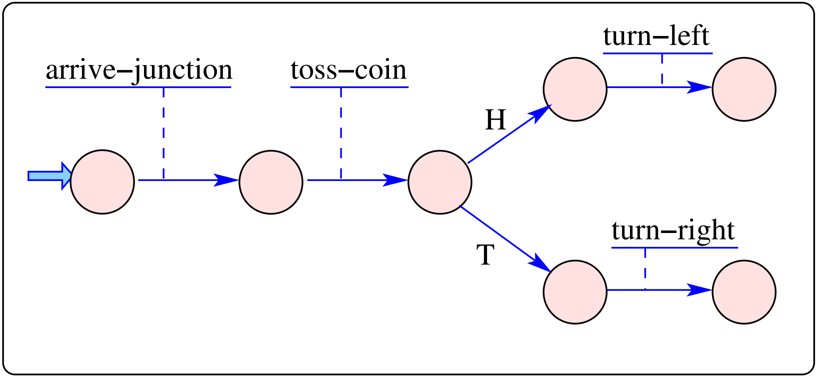
```
TRACE: s4 on-heat s5 ok s6 off-heat s0 ...
NON-TRACE: s5 off-heat s6 off-heat s0 ...
```

In this system the transition relation is deterministic, i.e. if $s_1 \stackrel{a}{\Longrightarrow} s_2$ and $s_1 \stackrel{a}{\Longrightarrow} s_3$ then $s_2 = s_3$.

Non-determinism is useful for getting succinct specifications. When you abstract out elements of a program, this may give rise to non-determinism.

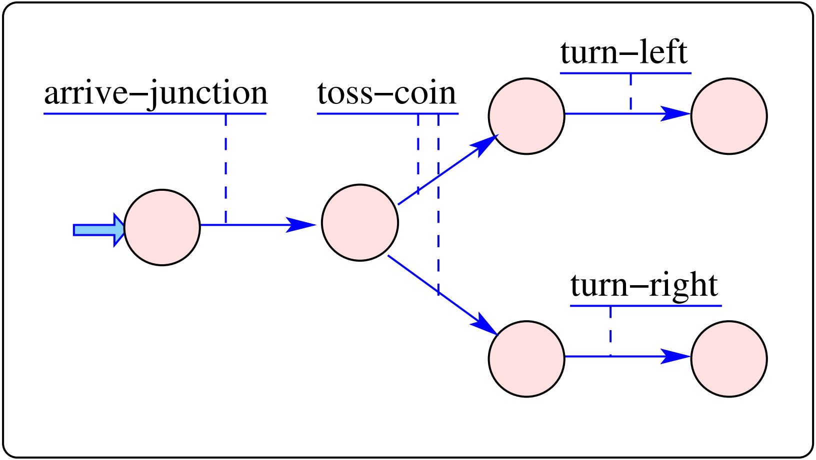
Deterministic system

Arrive at a road junction, toss a coin, turn left or right:



Non-deterministic system

Less states, is non-deterministic, may still be sufficient



Definitions for state transition systems

Paths and computations

Path: A **path** is an allowable sequence of states.

Run: Path starting from initial state is termed a **run**.

In a transition system, $\theta = s_0\ s_1\ s_2\ s_3\ \ldots\ s_n$ (written $s_0 \stackrel{*}{\Longrightarrow} s_n$) is a run, with a complete trace of

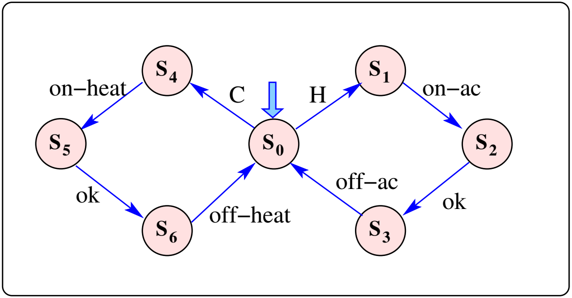
$s_0\ a_1\ s_1\ a_2\ s_2\ a_3\ s_3\ \ldots\ s_{n-1}\ a_n\ s_n$.

Computation: The sequence of actions $a_1\ a_2\ a_3\ \ldots\ a_n$ is termed a **computation**.

Every run θ induces a computation σ, and given a specific run θ, the corresponding computation σ is not unique. However, if the system is deterministic, for every computation σ, there is a unique run θ.

Examples for definitions

Paths, runs, computations



Path: $S_1\ S_2\ S_3$

Run: $S_0\ S_1\ S_2\ S_3$

Computation: C on-heat ok

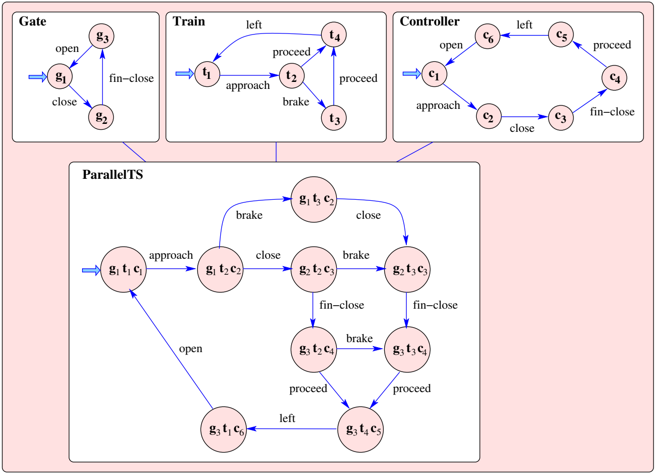
System behaviours and properties

Behaviours and properties...

- The behavior of a transition system is:
  - Its set of runs.
  - Its set of computations.
- Does the behavior of TS have the desired property?
  - Does every computation (run) of the transition system have the desired property?
  - *In no computation, C is immediately followed by On-Ac...*

Construct a parallel composition

The basic idea...



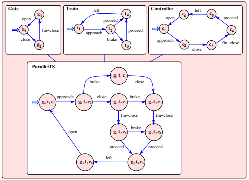
Parallel composition

How to construct the parallel composition of a finite set of TS

- Take cartesian product of states of each transition system $\mathcal{S}_{\text{Gate}} \times \mathcal{S}_{\text{Train}} \times \mathcal{S}_{\text{Controller}}$, and
- derive any allowable transitions for each of these states, performing common actions together.
  - Example: start from the new starting state $(g_1 t_1 c_1)$ synthesized from the starting states (g_1, t_1 and c_1), and
  - construct all possible future states by taking any actions common to the transition systems.
  - This process continues, at each stage constructing any new future state(s), until we have exhausted all possible actions.

Parallel composition

Using the TGC system:



- For the ParallelTS system, the cartesian product of all the states gives 72 potential new states (a state space explosion), although only 9 of these are actually used.
- For example, the action available at $(g_1 t_1 c_1)$ is `approach`, common to both the Train and the Controller. When we take this action, the next state is $(g_1 t_2 c_2)$.

Parallel composition

The parallel composition may be difficult...

- $TTS = TS_1 \parallel TS_2 \parallel \ldots \parallel TS_n$: TS is presented implicitly!
- Fix a communication convention between the TS, and present $TS_1, \ldots, TS_n$
- We wish to analyze TS and often implement TS.
- But constructing TS first explicitly is often hopeless.
  - if $|TS_i| = 10$, and $n = 6$ - then what is the worst case $|TS| =???$
- STATE SPACE EXPLOSION!!

TTS overview

TTS=TS+ClockVars

- Timed transition systems are transition systems with *clock variables* which are used to record the passage of time.
- Clock variables operate like hardware timers, can be reset to 0 during a transition, and can be read.
- Transitions are *guarded* (or constrained) by the current values of the relevant clock variables, which evolve in real-time until reset to 0.
- To capture all this, transitions are annotated with 3 items: the action, a set of clocks to reset, and a guard predicate over the clock variables:

TTS overview

Some examples...

- Turn OFF AC if the termperature is OK or if 5 time units have elapsed since turing it ON...
- Turn ON AC within 3 time units of receiving the HOT signal...

Formal definition

A timed transition system TTS is...

- a 6-tuple $(S, S_{\mathrm{in}}, \mathrm{Act}, X, I, \rightarrow)$:
- S, $S_{\mathrm{in}} \subseteq S$ and Act are as defined before
- X is a finite set of **clock variables**
- $I : S \rightarrow \Phi(X)$ assigns a clock invariant to each state. The clock constraints are limited to constraints of the form

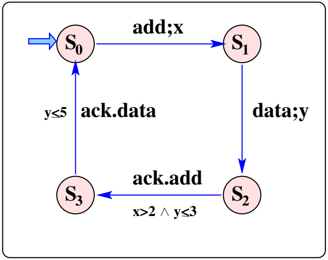
$$\Phi(X) = x \leq c \mid x \geq c \mid x < c \mid x > c \mid \phi_1 \wedge \phi_2$$

  where $c \in \mathbb{Q}$.
- $\rightarrow : S \times \mathrm{Act} \times 2^X \times \Phi(X) \times S$ is the **transition relation**, and 2^X is the set of subsets of X (the powerset of X)[a].

[a] The notation reflects the size of the powerset.

Example TTS

A simple timed transition system...



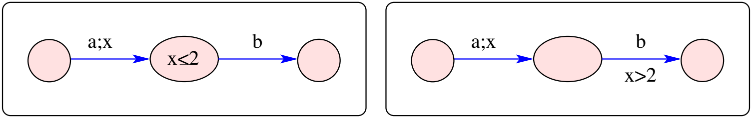
- Actions: $\mathrm{Act} = \{\mathrm{add}, \mathrm{data}, \mathrm{ack.add}, \mathrm{ack.data}\}$, clocks: $X = \{x, y\}$. When transition taken, clocks are reset to 0.

$(s_0, \mathrm{add}, \{x\}, \mathrm{True}, s_1)$, $(s_3, \mathrm{ack.data}, \emptyset, y \le 5, s_0)$: valid transitions.

Invariants and guards are related

Stay in state as long as state invariant not violated.

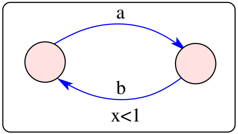
If time points violate the invariant, and no output is enabled, we have a *time* deadlock. If more than one output transition is enabled, the choice between the transitions is made non-deterministically



On left we have a state invariant asserting that x should be less than or equal to 2 time units in this state. On right we have a different guard asserting that the transition is enabled if x is more than 2 time units. .

Zeno computations

Consider this TTS:



- Look at the computation $(b, \frac{1}{2})\,(a, \frac{1}{2})\,(b, \frac{3}{4})\,(a, \frac{3}{4})\,(b, \frac{15}{16})\ldots$
- This could go on forever
- We must model our systems carefully.

Parallel compositon of TTS

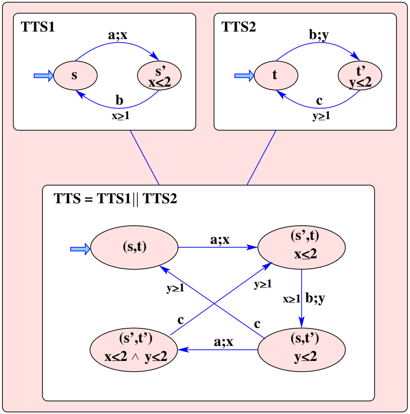
Use the following principles:

To compute:

- **Do common actions** together.
- **Take union** of all the clock variables.
- **Take conjunction** of all the guards (state invariants).

Example parallel composition

Two TTS:



Formally...

Formalized by using the following construction:

Given $\mathrm{TTS}_1 = (S_1, S_{0,1}, \mathrm{Act}_1, X_1, I_1, \rightarrow_1)$, $\mathrm{TTS}_2 = (S_2, S_{0,2}, \mathrm{Act}_2, X_2, I_2, \rightarrow_2)$ the product construction $\mathrm{TTS} = \mathrm{TTS}_1 \| \mathrm{TTS}_2 = (S, S_0, \mathrm{Act}, X, I, \rightarrow)$ is:

- $S = S_1 \times S_2$, $S_0 = S_{0,1} \times S_{0,2}$, $\mathrm{Act} = \mathrm{Act}_1 \cup \mathrm{Act}_2$, $X = X_1 \cup X_2$
- $I(s_1, s_2) = I_1(s_1) \wedge I_2(s_2)$
- Finally, $\rightarrow$ is the least subset of $S \times \mathrm{Act} \times \Phi(X) \times 2^X \times S$ (given $(s_1, a, \phi_1, Y_1, s_1') \in \rightarrow_1$ and $(s_2, b, \phi_2, Y_2, s_2') \in \rightarrow_2$) that satisfies:
  - Case 1: $a = b \in \mathrm{Act}_1 \cap \mathrm{Act}_2$ then $((s_1, s_2), a, \phi_1 \wedge \phi_2, Y_1 \cup Y_2, (s_1', s_2')) \in \rightarrow$
  - Case 2: $a \in \mathrm{Act}_1 - \mathrm{Act}_2$ then $((s_1, t), a, \phi_1, Y_1, (s_1', t)) \in \rightarrow$ for every $t \in S_2$
  - Case 3: $b \in \mathrm{Act}_2 - \mathrm{Act}_1$ then $((t, s_2), b, \phi_2, Y_2, (t, s_2')) \in \rightarrow$ for every $t \in S_1$

The process...

Three steps...