Algorithmic Verification

Comp4151
Lecture 11-A
Ansgar Fehnker

Overview

Model Checking Approaches

- Explicit State Model Checking
- Symbolic Model Checking
- Bounded Model Checking
- Automatic Abstraction Refinement

- Correctness of software, hardware and protocols
- Correctness for finite state systems

Overview

Time critical systems

Correctness of embedded and distributed systems
- correctness depends result of a computation
- and on the timing of events, computations, responses

Time critical systems
- railway crossing
- process control
- consumer electronics
- automotive and avionics
- wireless

Overview

Next two weeks

Model checking real-time systems

Themes
- Decidability
- Efficient implementations and data structures
- Application examples

Today
- Real or continuous time vs discrete time models
- Syntax and semantics of timed automata
- Example: Biphase Mark Protocol
Discrete Time vs. Real Time

Discrete time

Finite model of time
- time ranges over the natural numbers
- events can only occur at integer times
- time advances by multiples of a smallest time step
- use a special tick event to synchronize

Example

Intelligent light switch
- Press button twice quickly to switch to bright
- Press it once to switch to dimmed
- If light is on or dimmed, pressing button switches light off

Discrete time model
- tick-event just a simple addition to finite state models
- standard model checkers and temporal logic for verification
- delays are modelled explicitly
- synchronization either to strict or to loose

⇒ suitable to model synchronous systems that evolve in lock-step
⇒ inadequate for many asynchronous distributed systems.

Example: wireless networks
- local clocks with drift and jitter
- discrete time may introduce absent synchrony
- however, nodes do synchronize on time
- complete asynchronicity inadequate as well
Discrete Time vs. Real Time

Real (or continuous) time

- Infinite model of time
- Time ranges over the positive reals
- Events can take place at any point in (real) time
- Multiple events can take place at the same point in time
- Time may advance by any positive real amount
- No smallest time step
- Timers and delays modelled using real valued variables (clocks)

Example

Intelligent light switch

- Press button twice quickly to switch to bright
- Press it once to switch to dimmed
- If light is on or dimmed, pressing button switches light off

Discrete Time vs. Real Time

Continuous time model

- Clock variables natural to model timers and delays
- Delay is modelled implicitly
- Real-valued variables lead to infinite state system
- Model checking algorithms become more complicated

⇒ Suitable to model systems that synchronize on real time
⇒ Too much good for completely discrete systems.

Example: Bi-Phase mark protocol

- Protocol for sending bits as square wave
- Local clocks for sender and receiver
- Clock drift and non-deterministic delay
- To be discussed later in this lecture
Comp4151 Ansgar Fehnker

Real-Time vs Real-Time

The term *real-time* often refers to
- supervisory systems that update information at the same rate as they receive data
- operating systems that employ reliable and predictable scheduling to minimize the number of missed deadlines, tardiness, ...
- control systems that react to input within some small upper limit on the response time
- simulators that ensure that simulation-time proceeds at the same rate as the simulated time.

Real-time models with real-valued clock-variables

Outline

Today
- Real or continuous time vs discrete time models
- Syntax and semantics of timed automata
  - Syntax of timed automata
  - Invariants and guards
  - Semantics of timed automata
  - Executions and runs
  - Reachability
  - Timed Languages
  - Composition
- Example: Biphase Mark Protocol

Introduction to Timed Automata

The basics
- A finite control graph with locations and edges
- Instantaneous transitions along edges,
- Delays while in location

Introduction to Timed Automata

The basics
- Real-valued clocks, that increase at the same rate
- Constraints on clocks as guard on edges
- Clock resets to measure time between transitions
- Invariants in locations to enforce progress
- Labels for synchronization
Syntax of Timed Automata

Definitions

Clocks
A clock is a variable ranging over the positive reals $\mathbb{R}^+$.  

Constraints
Given a set of clocks $C$, let $\Psi(C)$ be defined by
$$\psi = \phi \land \psi \lor \lnot \psi \land (x \leq n) \lor (x < n) \lor (y \leq m) \lor (y < m)$$
where $x, y \in C$, $n \in \mathbb{N}$, $m \in \mathbb{Z}$.

- Restriction to simple and diagonal constraints to ensure decidability
- Constraints of the form $x+y<n$ would make model checking TAs undecidable
- [AD94] defines TAs without diagonal constraints
- TAs without diagonal constraints are called diagonal-free
- Each TA with diagonal constraints is bisimilar to diagonal-free TA

Timed Automata

Given a set $C$ of clocks, a timed automaton is a tuple
$$(\text{Loc}, I_0, \Sigma, E, \text{Inv})$$
where
- $\text{Loc}$ is a finite set of locations
- $I_0$ is the initial location
- $\Sigma$ is a set of labels
- $E$ is a set of edges
- $\text{Inv} : \text{Loc} \rightarrow \Psi(C)$ a mapping from locations to invariants
Guards and Invariants

- Guards enable progress, invariants enforce progress

Invariants may lead to deadlocks

Semantics

Definition

The operational semantics of a timed automaton \((\mathsf{Loc}, \mathsf{I}_0, \Sigma, E, \mathsf{Inv})\) is given as a (timed) transition system with

- set of states \(S = \{ (l, v) \mid l \in \mathsf{Loc}, \ v \models \mathsf{Inv}(l) \} \)
- initial state \(s_0 = (l_0, 0)\)

(to be continued)
Semantics

Definition (cont)

- transition relation \( R \subseteq S \times \Sigma \cup R_{\geq 0} \times S \) that contains the following:
  - discrete transitions \( (l,v) \xrightarrow{\sigma} (l',v') \)
    if there exist \((l,g,\sigma,l') \in E \) s.t. \( v = g \), and \( v[r:=0] = v' \)
  - delay transitions \( (l,v) \xrightarrow{d} (l,v + d) \)
    for \( d \in \mathbb{R}_{\geq 0} \)
    if for all \( 0 \leq d' \leq d \) holds \( v + d' = v + d \)

Runs and Executions

Run (or execution)

A finite or infinite sequence of transitions
\[
(l_0,v_0) \xrightarrow{a_0} (l_1,v_1) \xrightarrow{a_1} (l_2,v_2) \xrightarrow{a_2} (l_3,v_3) \ldots
\]
with initial state \((l_0,v_0)\) and \( a_i \in \Sigma \cup R_{\geq 0} \)

Dense time

Transitions may occur at any point in real time
\[
(l_0,v_0) \xrightarrow{\sqrt{2}} (l_1,v_1) \xrightarrow{\sqrt{2}} (l_2,v_2) \ldots
\]

Super dense time

Multiple transitions may occur at any point in real time
\[
(l_0,v_0) \xrightarrow{a_0} (l_1,v_1) \xrightarrow{a_0} (l_1,v_1) \xrightarrow{a_0} (l_1,v_1) \ldots
\]

Zeno

Time divergent

An infinite run is time-divergent if it has an infinite number of delays \( d_i \) such that
\[
\lim_{i \to \infty} \sum_{i=0}^{\infty} d_i = \infty
\]
Counterexample
\[
(l_0,v_0) \xrightarrow{1/2} (l_0,v_1) \xrightarrow{1/4} (l_0,v_2) \xrightarrow{1/8} (l_0,v_3) \ldots
\]
Non-Zenoness

A timed automaton is non-zeno, if each finite run can be extended into a time-divergent run

Reachability

Reachability

A state \((l,v)\) is reachable if \((l_0,v_0) \xrightarrow{\ldots} (l,v)\)
A location \( l \) is reachable if there exist a \( v \) such that \((l,v)\) is reachable.

Time additivity

Two successive delays can be combined
\[
(l_0,v_0) \xrightarrow{d_0} (l_0,v_1) \xrightarrow{d_1} (l_0,v_2) \iff (l_0,v_0) \xrightarrow{d_0+d_1} (l_0,v_2)
\]
for \( d_0, d_1 \in R_{\geq 0} \).
Finite runs can be rewritten as alternating sequence of transitions and delays
Timed Languages

Definitions

- A **timed action** is a pair \((\sigma, t)\) with \(\sigma \in \Sigma, t \in \mathbb{R}_{\geq 0}\).
- A **timed trace** of timed automaton \(A\) is a finite or infinite sequence \((\sigma_1, t_1), (\sigma_2, t_2), (\sigma_3, t_3), \ldots\) with \(t_0 \leq t_1 \leq t_2 \leq \ldots\). There exist a run \((l_0, v_0), d_1(l_1, v_1), d_2(l_2, v_2), \ldots\) and \(t_{i+1} = t_i + d_i\) for \(i > 0\), and \(t_0 = 0\).
- The **timed language** of \(A\) is the set of all timed traces of \(A\).
- The **untimed language** is the restriction of the timed language to \(\Sigma\).

Composition

Asynchronous and distributed systems modelled by **parallel composition** of timed automata.
- Timed automata (typically) **closed** under parallel composition.
- Several competing definitions of parallel composition.
  - Synchronization on **common action labels** [Alur].

Timed \(I/O\) automata [Lynch et al].
Uppaal’s **handshake** synchronization.
Uppaal’s **broadcast** channels.

Composition of common labels [Alur]
- Set of locations \((F_1, F_2)\) is the product of locations \(F \in \text{Loc}_1, F \in \text{Loc}_2\).
- Invariants in \((F_1, F_2)\) are the conjunction of invariants in \(F_1\) and \(F_2\).
- Edges must synchronize on shared labels, guard is the conjunction of guards, reset sets the union of reset sets.
- Edges without shared labels may fire without synchronization.

![Timer Switch Example](image)
Handshake synchronization

- Uppaal defines a network of timed automata
- Synchronization via channels
- Handshake synchronization on pairs of \(!\) and \(?\) label
- Both guards have to be satisfied
- If multiple pairs possible choose non-deterministically

Broadcast synchronization

- Uppaal defines a network of timed automata
- Synchronization via channels
- Broadcast from \(!\) channel to all \(?\) channels
- Guards only on transition labelled \(!\)
- If multiple \(!\)-transitions enabled choose non-deterministically

Composition

- Asynchronous and distributed systems modelled by parallel composition of timed automata
- Timed automata (typically) closed under parallel composition
- Several competing definitions of parallel composition
  - Synchronization on common action labels [Alur]
  - Timed I/O automata [Lynch et al]
  - Uppaal’s handshake synchronization
  - Uppaal’s broadcast channels
  - Many others
- Synchronization via shared variables

Outline

- Today
  - Real or continuous time vs discrete time models
  - Syntax and semantics of timed automata
    - Syntax of timed automata
    - Invariants and guards
    - Semantics of timed automata
    - Executions and runs
    - Reachability
    - Timed Languages
    - Composition
  - Example: Bi-phase mark protocol
Example

Biphase Mark Protocol

- Convention for representing both a string of bits and clock edges in a square wave.
- Used, for instance, in:
  - Intel 82530 Serial Communications Controller
  - Ethernet
  - Manchester encoding
  - Optical communications
  - Satellite telemetry applications
- Model based on work in [Vaandrager and de Groot]

Example

Terminology

- Message encoded as square wave over as many cells as bits
- Cells are divided into mark subcell and code subcell
- Receiver should sample at the beginning of mark subcell, and somewhere within the code subcell

Example

Assumptions

- Sender and receiver have each its own clocks
- Clocks with drift and jitter
- The signal takes some time after a change in voltage at stabilize.
- Sampling within this period may produce any value.
- The receiver may sample non-deterministically at some point during clock cycle.

Example

A compositional model
Example

A compositional model

Example

The digital clocks

Example

The digital clocks

Example

The digital clocks

Constants

Constants for a typical configuration

- length cell: 32 clock cycles
- length mark subcell: 16 clock cycles
- sampling point: 23 clock cycles
- min length clock cycle: 81 time units
- max length clock cycle: 100 time units
- max length unstable edge: 81 time units
- max sample delay < 81 time units

4 clocks (sender, receiver, wire, sampler)

5 channels (put, get, edge, tick, tock)
Example

The digital clocks

- Input event: tick?, get?
- Output event: edge!
- Input variable: in
- Local variable: n
- No clock (!) guards on tick?
- Time may not advance while in an urgent location
- Equivalent to a reset a clock x on entry and invariant x<=0 in location
- Reduction on the number of used clocks.

Example

The encoder

Example

The wire

- Input event: edge?
- Local variable: v (voltage)
- Output variable: w (output voltage)
- Local events: fuzz! and stable!
- Voltage changes upon input edge!
- Output voltage may change (fuzz) during edglength time after edge!

Example

The digital clocks
Example

The sampler

- Input variable: w (voltage)
- Output variable: new
- Input event: tick?
- Local clock: s
- Local event: Sample!
- Samples variable w less than sampledelay time after tick?
- Point of sampling non-deterministically

Example

The decoder

- Input variable: new
- Local variable: old, m
- Output variable: out
- Input event: tick?
- When change in new is detected, wait for sample.
- The output is 0 if sampled value equals old, 1 otherwise.
- Copy new to old, wait for next edge

Example

The digital clocks

- Input variable: out
- Local variable: buf
- Output variable: in
- Input event: put?
- Output event: get!
- Send non-deterministically 0 or 1
- One place-buffer buf to send while waiting for feedback
Model Checking

Main problems

The state space is infinite: \( S = \{ (l,v) | l \in \text{Loc}, v \models \text{Inv}(l), v : C \rightarrow \mathbb{R} \geq 0 \} \)

The transition relation is infinite: \( R \subseteq S \times \Sigma \cup \mathbb{R} \geq 0 \times S \)

Alur and Dill have a solution to this problem.