Refinement for Timing Properties

Action line: Abstraction and Compositionality for Timed Systems

Marius Minea

Institute e-Austria Timişoara, Romania

Issues to discuss within action line

Frameworks for compositionality

– various formalisms are used (timed automata, timed I/O automata, timed interfaces, etc.)

– focus on one particular formalism ?

– which is:

  – more expressive ?

  – easier to handle ?

  – more suitable for composition ?

Automation

– of generating an abstract version of a timed system
(with respect to some property to verify)

– of generating environments (contexts)
for use in assume-guarantee reasoning

Issues to discuss within action line (2)

Abstract vs. refinement

or: Synthesis or analysis ?

– focus on refining timing specification into designs ?

– or on generating timed abstractions from a low-level description for use in verification

This talk: discuss some approaches to checking timed refinement

Counterexample-based abstraction refinement

– used successfully in model checking (untimed systems, software)

1. construct (initially very coarse) abstract model
2. model check with respect to specification

3a. if correct, done (abstraction is conservative)

3b. if counterexample, find correspondent in concrete system

4a. if counterexample real (feasible), done (error found)

4b. if not feasible, refine abstraction

(eliminate spurious counterexample ,continue loop)

Timing Verification by Successive Approximation

[Alur, Itai, Kurshan, Yannakakis – Information and Computation, 1995]

Model: – parallel composition of ω-automata

$$P = P_1 || P_2 || \ldots || P_n$$

– actual model M obtained from P by adding delay constraints D

Specification: property T, also as (timed) ω-automaton

Verification problem: (timed) language inclusion:

$$\mathcal{L}(M) \subseteq \mathcal{L}(T)$$

Details

– each process P_i has set of *delays* Δ_i
– each delay $\delta \in \Delta_i$ is defined by lower and upper bounds, $\alpha(\delta)$, $\beta(\delta)$
– each event in the alphabet Σ may be associated with the beginning or end of a delay

Restriction:
in any sequence of events considered, delays may not overlap
(events in between the beginning and end of a delay
may not themselves start or end another delay)

Approach

Want to prove: $\mathcal{L}_D(P) \subseteq \mathcal{L}(T)$
timing-consistent sublanguage of P included in T

brute-force approach: force constraints in D by *region automaton*
– is exponential (unavoidable, since problem PSPACE-complete)

Proposed solution: *try using simpler approximation of constraint D*

Counterexample-based refinement

Starting from counterexample to $\mathcal{L}(P) \subseteq \mathcal{L}(T)$
1) check timing consistency of counterexample.
Two cases:

– finite counterexample – check quadratic (in number of processes)
(standard negative cost cycle algorithm in matrix)

– infinite counterexample $\sigma'\sigma^\omega$
cubic shortest path algorithm in periodic weighted digraph

2) select *small* (optimal?) delay constraint D' that:

– is implied by system delay constraint D

– makes the detected counterexample timing-inconsistent

This delay constraint is used as abstraction.

Usage in practice

Examples and case studies:

– tested on train-gate controller and versions of mutual exclusion

Implementation (in COSPAN)

– checking delay constraint based on region-graph construction

$\Rightarrow$ could possibly be improved by zone automaton ?

Lazy Approximation for Dense Real-Time Systems

[M. Sorea, FORMATS/FTRTFT 2004]
also Ph.D. thesis (2004):
"Verification or Real-Time Systems through Lazy Approximations"

Model: timed automaton
Specification: TCTL (dense-time CTL with bounds on operators)
Abstraction: zone-based, using *predicates* for relations between clocks
– abstract state = location + clock predicates
– can refine incrementally, introducing new clock predicates

Lazy approximation (cont'd)

Key issue: approximation is no longer the same kind of system:

- initial model: timed automaton
- abstract model: finite-state (zone) automaton

Other aspects:

– considers event-recording logic
  counterpart of event-clock automata [Alur, Fix, Henzinger '99]
  decidable, with effective tableau construction
  $\Rightarrow$ model checking problem reduces to logic implication

– not substantiated with significant case studies
  worth investigating performance on realistic examples

Comparison and potential directions

[Alur et al.]
– delay abstractions *global* and *explicit* (lower/upper bounds)
– abstract system is of the same type (automaton + delays) as original
– uses region graph construction

[Sorea]
– delay abstractions are local
  (only some states/zones) split by predicates (*lazy*)
– abstract system no longer timed (timing implicit in zones)
– uses zone automaton construction

Worth investigating (?)
– produce simplified delay constraints explicitly, as in [Alur et al.]
– use zone automaton for verification
– evaluate on case studies