Compositionality for Timed Models

– a partial survey –

Marius Minea

Institute e-Austria Timişoara, Romania

Compositional model checking for timed automata

[Larsen, Petterson, Wang 1995]

Idea: moving part of the system into the specification formula:

$$A||B \models \varphi \quad \text{iff} \quad A \models \varphi/B$$

where φ = formula in a logic for safety and bounded liveness

and φ/B = *quotient* of formula wrt automaton B

– quotienting distributes over most operators in formula

– quotienting defined individually for each location l of automaton B; the overall quotient is that for the initial location l_0 of B

– can lead to *explosion in size of formula*, especially by quotienting over quantification of time and action transitions, e.g.

$$(\forall\varphi)/l = \forall(I(l) \Rightarrow \varphi/l$$

quotient for actions $[a]\varphi$ involves all actions which synchronize to a

In practice: needs lots of *heuristics* for simplifying/eliminating clock constraints in resulting formula

Compositional frameworks for timed systems

[Sifakis, Bornot, Tripakis; Sifakis & Goessler]
Problem: separate specification of transitions (specifically guards) and time progress conditions (e.g. state invariants) may lead to deadlocks.
$\Rightarrow$ need compositional formalism which avoids deadlocks and timelocks

Urgency types for transitions; timed automata with deadlines
– *eager* $d = g$ will fire as soon as enabled
– *delayable* $d = g \downarrow$ will fire at latest at upper bound
– *lazy* $d = false$ can be postponed indefinitely

Flexible synchronization modes: define synchronization interval in terms of the individual enabling intervals of processes (AND, MIN, MAX)

Priority choice operators – modal operators which can be used to preserve deadlock freedom

Schedulers and their role: controllers that restrict system behavior; relationship to controller synthesis

Modularity for Timed and Hybrid Systems

[Alur, Henzinger 1997]
– modularity, liveness and control in reactive and real-time setting
– discuss the case of *open* systems
– extend formalism of *reactive modules* to real-time

– receptiveness condition becomes *nonzenoness* (diverging time)
– analyze it as game between system and environment (both symbolic and region-graph algorithm), extending timed I/O automata results
– circular assume-guarantee rule remains valid for receptive modules:

$$P_1||Q_2 \leq Q_1 \wedge Q_1||P_2 \leq Q_2 \Rightarrow P_1||P_2 \leq Q_1||Q_2$$

– use results for synthesis of receptive controllers

Simulation and Assume-Guarantee for TA

[Serdar Tasiran, PhD thesis, Berkeley, 1998]

1) Checking *timed refinement* (timed trace inclusion/timed simulation)

– gives algorithm using homomorphisms and reduction to checking of untimed homomorphism

– relies on region graph construction, can quickly become complex

2) Assume-guarantee reasoning for timed abstractions ($\leq_L$ and $\leq_S$)

– requires *non-blocking* timed automata: react to any input, and outputs change due to inputs only after non-zero delay

– with these restrictions, circular assume-guarantee applies:

if $A_1||B_2 \leq_L A_2$ and $A_2||B_1 \leq_L B_2$ then $A_1||B_1 \leq_L A_2||B_2$

– same rule with same conditions applies for timed simulation $\leq_S$

– witness simulation for composition: computed from simulation relations for components

Assume-Guarantee for Timing Diagrams

[Amla, Emerson, Namjoshi, Trefler 2001] "timing" in diagrams is not explicit, but implicit in a reference clock

– generic formalism for synchronous composition of processes with variables

– to deal with liveness: need *closure* $CL(P)$ of process P

– prior approach [Alur & Henzinger '96] breaks circularity by taking closure of specification in one assumption: $CL(Q_1)||P_2 \models Q_2$

– here: additional check; can still use liveness properties as assumptions

Assumptions for $P_1||P_2 \models S$:

– $P_1||Q_2 \models Q_1$ and $Q_1||P_2 \models Q_2$ and $Q_1||Q_2 \models$S($spec$)

– $P_1||CL(T) \models T + Q_1 + Q_2$ or $P_2||CL(T) \models T + Q_1 + Q_2$

Timing diagrams are formalizations of those used in circuit descriptions (with clock waveforms, sequential and concurrent dependencies)

– could timing constraints be added ?

Timed Interfaces

[de Alfaro, Henzinger, Stoelinga 2002]
– specify both *assumptions* (about timing of inputs) as well as *guarantees* (about timing of outputs)
– semantics is *optimistic*: an interface is *well-formed* if there is at least *some* environment that satisfies its input assumptions
– similarly, interfaces are *compatible* iff composition is *well-formed*, i.e., there exists a common environment in which they work

Issues in composition:
– control: error states (outputs are not acceptable inputs for the other)
– timing: time errors (one component cannot let time pass)

Game-theoretic view: interface compatibility checking using algorithms for solving timed games

Specific case:
– Timed interface automata with *input* and *output* invariants

Timed I/O Automata

[Kaynar & Lynch, 2003/2004]
Timed I/O Automata have:
– set X of internal variables, defining set Q of states;
– internal (H), input (I) and output (O) actions
– discrete transitions and timed trajectories

Requirements:
– *input action enabling*: $\forall x \in Q\,; \forall a \in I\ \exists x' \in Q\ .\ x \xrightarrow{a} x'$
– *time passage enabling*: in every state, time can either reach infinity or there is a trajectory which is (right-)closed and has a controllable action ($H \cup O$) enabled in its last state

Two TIOA are *comparable* if they have the same external actions.
Two TIOA are *composable* if they have disjoint internal variables and outputs, and hidden actions of one are not actions of the other.

Implementation relation $\leq$ is *trace inclusion*.

Assume-Guarantee for Timed I/O Automata

1) $A_1||B_2 \leq A_2||B_2$ and $A_2||B_1 \leq A_2||B_2$ imply $A_1||B_1 \leq A_2||B_2$ if:
– traces of A_2 and B_2 are closed under limits (*safety* properties)
– traces of A_2 and B_2 are closed under time extension
(do not impose stronger time passage constraints than $A_1||B_1$)

2) Conditions on A_2 and B_2 can be relaxed by introducing variant contexts A_3 and B_3, closed under limits and time-extension. Then:
$A_2||B_3 \leq A_3||B_3$ and $A_3||B_2 \leq A_3||B_3$ and
$A_1||B_3 \leq A_2||B_3$ and $A_3||B_1 \leq A_3||B_2$ imply $A_1||B_1 \leq A_2||B_2$

Reasoning can be extended to *liveness* (with more complex conditions)

Comparison and issues

Composability of components

– typically (timed {automata, diagrams, I/O automata}) is a separate precondition to any assume-guarantee rule

– timed interfaces: optimistic view of composability ($\exists$ *context*)

– work on establishing a framework for composability [Sifakis et al.]

Q: what (reasonable) restrictions result in simple composability check?

Safety and Liveness

– most assume-guarantee results concerned with safety

– liveness in a timed context – for timed I/O automata

Q: how to extend liveness results for other models ?

Comparison and issues

Completeness of assume-guarantee methods

– reasoning is usually incomplete for liveness; sometimes for safety

– [Namjoshi, Trefler 2000] give complete rule in untimed setting

– [Maier 2003]: assume-guarantee cannot be both sound and complete

(for a different setting) $\Rightarrow$ what is the relation ?

Automation of assume-guarantee checking

Q: for given goal $P_1||Q_1| = S$, how to split $S = P_2||Q_2$?

Q: if helper assertions/contexts are needed, how to generate them ?

– some answers (w/o explicit timing) in [Namjoshi, Trefler 2000]

– related to generating abstractions for timed systems

(lazy predicate abstraction [Sorea 2004])