Verification of timed systems

2 November 2005

– discrete and continuous time
– extensions of untimed model checking algorithms
– quantitative temporal logics
– timed automata

Timed sytems

= systems whose functional correctness depends on satisfying temporal constraints
– safety-critical systems (aviation, military)
– high-speed asynchronous circuits
– process control and fabrication systems
– communication protocols
– consumer electronics (increasingly so, incl. automotive control)
– timed synchronization protocols (e.g. in distributed systems)

Modeling

Any real system executes in physical time
⇒ untimed models studied so far are merely an *abstraction*
e.g. temporal logic expresses *qualitative*, not *quantitative* properties
Still: most formalisms start from an untimed description on top of which the time dimension is added later
– *discrete time*: all events happen at multiples of a time quantum
  models: e.g., automata with an integer duration for each transition
– *continuous time*: events happen at arbitrary moments on a time scale of reals
  models: timed automata, timed Petri nets, languages with timing constructs

Few formalisms are created specifically with time as first-class feature
e.g., *duration calculus* [Zhou, Hoare, Ravn '91], with operators:
– $\lfloor f \rfloor$: *duration* for which f holds (integral over time)
– concatenation of two time intervals
sample property: gas leak less than 30 seconds in any one-hour interval

Discrete and continuous time

What is the difference in expressiveness and efficiency ?
(How does a continuous-time system compare to its *discretized* model?)

[Henzinger, Manna, Pnueli '92]: discuss *timed transition systems*
(automata with lower/upper bounds on transitions)
– discretization preserves *qualitative* and some *quantitative* properties
e.g., invariance (**G**p) and response (p ⇒ **F**q) with time limits
– for other properties, *weaker* discrete-time versions can be derived

[Asarin, Maler, Pnueli '98] discuss combinational circuits, with limited time delays on the output of every gate:
– for acylic circuits, there is a discretization quantum which preserves qualitative properties (ordering of events)
  e.g., $1/n$ for a circuit with n signals
– there are *cyclic* circuits whose qualitative behavior is not preserved by any discretization (e.g., ring of 3 inverters)

Classical theories for real-time systems

One of main problems: *schedulability analysis*
Given a set of processes and their parameters (periods, deadline) is there a schedule that satisfies the deadlines ?

Rate-monotonic scheduling [Lehoczky, Liu, Layland]
– assigns priorities in increasing order of periods
(provably optimal)
– satisfiability test based on total CPU utilization (%)

– Advantages: simple method, optimal, fast analysis
– Disadvantages: restrictive model (periodic processes + some extensions); incomplete method, not applicable to high loads ($> ln2 \simeq 70\%$)
We discus more general approaches.

Quantitative analysis of temporal properties

RTCTL (real-time CTL) can express quantitative temporal properties
(e.g., p does not appear earlier than 5 time units)
but not a more detailed analysis (what is the maximum delay of p)

⇒ We define algorithms that can calculate such parameters
and have an efficient *symbolic* implementation (with BDDs)

– length of shortest and longest path between two sets of states
(expressed by predicates that characterize them)
e.g., longest execution time (*schedulability*)
– minimal/maximal number of occurrences of a property on a path
e.g., how many times the process is in the *wait* state

[Courcoubetis & Yannakakis; Campos, Clarke et al.]

Shortest path between two sets of states

Breath-first search from $start$ until $final$ first reached or no new states explored

In each iteration: Q = states reached in i steps R = set of all reached states, grows until fixpoint

procedure min(start, final)
for $(i \leftarrow 0, R \leftarrow Q \leftarrow start; Q \cap final = \emptyset; i{+}{+})$ **do**
  $Q \leftarrow$ Succ(Q); $R' \leftarrow R \cup Q$; // Q = frontier
  if $(R' = R)$ **then**
    return ∞; // can't reach $final$
  $R \leftarrow R'$; // R = all reached states
return i;

Longest path between two sets of states

Determines maximal length of a path until $final$ first reached starting from $start$.
Assumes: system reduced to reachable states
We search longest path from $start$ which does not reach $final$, by backwards traversal from states that are not in $final$
R = initial points of paths that can stay outside $final$ for i steps; decreases until fixpoint

procedure max(start, final)
for $(i \leftarrow 0, R = S \setminus final; R \cap start \neq \emptyset; i{+}{+})$ **do**
  $R' \leftarrow$ Pred$(R) \setminus final$;
  if $(R' = R)$ **then**
    return ∞; // exists path not reaching $final$
  $R = R'$;
return i;

Temporal logics with explicit time

Temporal logics discussed so far (LTL, CTL) have temporal modalities (next state, future), but w/o reference to explicit time

$\Rightarrow$ Need additional features to express real-time properties (e.g., time-bounded response)

Large variety of logics with explicit time, depending on various choices:
– linear or branching-time
– discrete or continuous time
– with timed operators or explicit time variables

Depending on choices $\Rightarrow$ differences in expressivity, decidability, algorithmic complexity

The temporal logic RTCTL

Simplifies expressing temporal properties in the discrete case by augmenting temporal operators with time intervals

Consider a path $\pi = s_0 s_1 \cdots$. We define:
– $\pi \models f \mathbf{U}_{[a,b]} g \Leftrightarrow \exists i \,.\, a \leq i \leq b \wedge s_i \models g \wedge \forall j < i \,.\, s_j \models f$
– $\pi \models \mathbf{G}_{[a,b]} f \Leftrightarrow \forall i \,.\, a \leq i \leq b \Rightarrow s_i \models f$
– $\pi \models \mathbf{F}_{[a,b]} f \Leftrightarrow \exists i \,.\, a \leq i \leq b \wedge s_i \models f$

Example: $\mathbf{AG}(p \rightarrow \mathbf{AF}_{[0,3]} q)$:
p always followed by q within at most 3 time units.
For $a = 0, b = \infty$, we obtain CTL semantics

Algorithms: recursive, by modification of fixpoint algorithms:
– $\mathbf{E}[f \mathbf{U}_{[a,b]} g] = f \wedge \mathbf{EX}\,\mathbf{E}[f \mathbf{U}_{[a-1,b-1]} g]$ $(a, b > 0)$
– $\mathbf{E}[f \mathbf{U}_{[0,b]} g] = g \vee (f \wedge \mathbf{EX}\,\mathbf{E}[f \mathbf{U}_{[0,b-1]} g])$ $(b > 0)$
– $\mathbf{E}[f \mathbf{U}_{[0,0]} g] = g$

TPTL: Timed Propositional Temporal Logic

[Alur, Henzinger 1989]
– extension of propositional fragment of LTL (only path formulas)
– linear time, discrete (interpreted over state sequences)
– uses explicit variables for time, but with restrictions:
each variable is bound to time in a certain state (by a quantifier)

Example:: "each p is followed by a q within at most 10 time units"
$$\Box x.(p \rightarrow \Diamond y.(q \wedge y \leq x + 10))$$

TPTL: comparison with other formalisms

Example: response in at most 10 time units to a continuous request:
$$\Box x.(p \rightarrow p \mathbf{U} y.(q \wedge y \leq x + 10))$$

Comparison with first-order temporal logic

Variable now represents current time in each state
$$\Box((p \wedge now = x) \rightarrow p \mathbf{U}(q \wedge now = y \wedge y \leq x + 10))$$
then we fill in the appropriate quantifiers (error prone)
$$\Box \forall x.((p \wedge now = x) \rightarrow p \mathbf{U} \exists y.(q \wedge now = y \wedge y \leq x + 10))$$

Comparison with time-bounded operators

– TPTL can express such operators, e.g., $\Diamond_{\leq 5} \phi$ expressed as:
$$x. \Diamond\, y.(y \leq x + 5 \wedge \phi)$$
– operators with limits compare timepoints of two successive events
TPTL can compare times of two arbitrary events
$$\Box x.(p \rightarrow \Diamond(q \wedge \Diamond y.(r \wedge y \leq x + 5)))$$

TCTL: Timed CTL

[Alur, Courcoubetis, Dill, '90]
– branching time, continuous time, time-bounded operators
– interpreted on continuous-time computation trees
a *path* is a function $\rho : \mathbb{R} \to S$ from real numbers (time) to states

Syntax: $\phi ::= p \mid false \mid \phi_1 \to \phi_2 \mid \exists \phi_1 \mathbf{U}_{\sim c} \phi_2 \mid \forall \phi_1 \mathbf{U}_{\sim c} \phi_2$
(cu $\sim$ unul din $<, >, \leq, \geq, =$), $p \in AP$, $c \in \mathbb{N}$

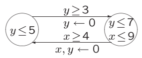
Semantics:
$s \models \exists \phi_1 \mathbf{U}_{\sim c} \phi_2 \Leftrightarrow \exists \rho, \exists t \sim c$ such that $\rho(t) \models \phi_2$
and for all $0 \leq t' < t$, $\rho(t') \models \phi_1$

Satisfiability: undecidable (!)

Model checking (with timed automata as model): decidable:
based on constructing a finite equivalence relation between paths

Timed automata

One of most widely used formalisms for continous-time systems
[Alur & Dill '90]
= finite-state machine augmented with set of continous-time *clocks*
(real-valued, advance synchronously)
– states/transitions labeled with *clock constraints*
– a clock can be reset on executing a transition
⇒ measures time passed since an event



Definition of timed automata

Set of clock constraints $\mathcal{B}(C)$ = conjunction of terms of form $x \prec c$, $c \prec x$, $x - y \prec c$, with $x, y \in C$, $\prec \in \{<, \leq\}$, $c \in \mathbb{Z}$

$A = (S, S_0, \Sigma, C, I, T)$, where
– S = finite set of locations (states, nodes)
– S_0 = set of initial locations
– Σ = alphabet of *labels* for transitions
– C = set of clock variables
– $I : S \to \mathcal{B}(C)$ associates to each state an *invariant*
(limiting the passage of time in that state)
– $T \subseteq S \times \Sigma \times \mathcal{B}(C) \times 2^C \times S$ = set of transitions
Transition $\langle s, a, g, R, s' \rangle$ from s to s' labeled by a is executed only if *guard* g is true, and *resets* clocks in $R \subseteq C$

Semantics of timed automata

Set of states: pair (s, v), where $s \in S$ = location
and $v : C \to \mathbb{R}$ is a clock assignment

– automaton can stay in a state as long as invariant is satisfied
(can leave state but cannot be forced unless invariant false)
– or can (but must not) execute instantaneous transitions when associated guard is true

Two types of transitions:
– action: $(s, v) \xrightarrow{a} (s', v')$ if there exists a transition $\langle s, a, g, R, s' \rangle \in T$, the guard $g(v)$ is true for assignment v, and v' is obtained from v by resetting clocks from R: $v' = v[R \leftarrow 0]$.
– passage of time: $(s, v) \xrightarrow{d} (s, v')$ if $v' = v + d$ ($v'(x) = v(x) + d$, $\forall x \in C$), şi $I(s)(v + \epsilon)$ true $\forall \epsilon \in [0, d]$ (invariant is preserved)

⇒ transition system with infinitely many states

Paths of the form $(s_0, v_0) \xrightarrow{d_1} (s_0, v_1) \xrightarrow{a_1} (s_1, v_1') \xrightarrow{d_2} (s_1, v_2) \xrightarrow{a_2} \ldots$

Parallel composition of timed automata

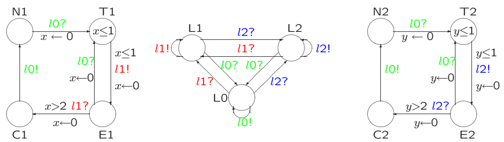
Execute synchronous transitions if labls match, separate transitions otherwise
Let $A_1 = (S_1, S_{01}, \Sigma_1, C_1, I_1, T_1)$ and $A_2 = (S_2, S_{02}, \Sigma_2, C_2, I_2, T_2)$, with $C_1 \cap C_2 = \emptyset$.
Define $A = A_1 || A_2 = (S_1 \times S_2, S_{01} \times S_{02}, \Sigma_1 \cup \Sigma_2, C_1 \cup C_2, I, T)$, where:
– $I((s_1, s_2)) = I_1(s_1) \wedge I_2(s_2)$
– if $\langle s_1, a, g_1, R_1, s_1' \rangle \in T_1$ and $\langle s_2, a, g_2, R_2, s_2' \rangle \in T_2$, cu $a \in \Sigma_1 \cap \Sigma_2$ then $\langle (s_1, s_2), a, g_1 \wedge g_2, R_1 \cup R_2, (s_1', s_2') \rangle \in T$ (synchronization)
– if $\langle s_1, a_1, g_1, R_1, s_1' \rangle \in T_1$, with $a \in \Sigma_1 \setminus \Sigma_2$, then $\forall s_2 \in S_2$, $\langle (s_1, s_2), a_1, g_1, R_1, (s_1', s_2) \rangle \in T$
– if $\langle s_2, a_2, g_2, R_2, s_2' \rangle \in T_2$, with $a \in \Sigma_2 \setminus \Sigma_1$, then $\forall s_1 \in S_1$, $\langle (s_2, s_1), a_2, g_2, R_2, (s_2', s_1) \rangle \in T$

More general: with synchronication function to match transition labels

Example: mutual exclusion

Fischer's mutual exclusion protocol
correctness based on respecting time constraints

Synchronization: pairs of transitions a? and a!

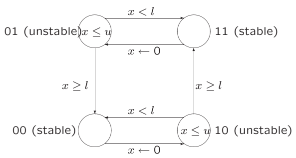


Can prove: correct if time constants (here 1 and 2) have this ordering.

Example: Modeling asynchronous circuits

Continuous-time model ⇒ more precise than discrete time; appropriate for modeling asynchronous and transient behavior

E.g. delay element: propagates input to output
– if input pulse not shorter than l
– with delay at most u



Finite representations for timed automata

State = pair (s, v) of location and clock assignment
⇒ state space is infinite (even uncountable)

But: we cannot observe behavior with arbitray precision
– constraints from automaton have integer time limits
– formulas of temporal logic also have integer constants

Questions:
– when are two states (s, v) and (s, v') with same location, but different clock assignments equivalent ?
– and is there a *finite* number of equivalence classes ?

Two approaches:
– *time regions* ⇒ region graph = finite automaton
– *time zones* ⇒ geometric constraints, symbolic exploration

Region graph: Motivation

When are two states (s, v) and (s, v') equivalent ?

1. if same transitions can be taken from both states
– conditions on transitions can have arbitrary integer time bounds
e.g. there may be transitions a with $x > 4$ and b with $x < 5$
$(s, x = 4.2)$ and $(s, x = 4.7)$ can both execute either a or b
$(s, x = 4.2)$ and $(s, x = 5.1)$ are *not* equivalent
⇒ must have same integer part for all clocks

$(s, x = 4)$ cannot execute a, but $(s, x = 4.1)$ can
⇒ fractional parts must both be nonzero or both zero

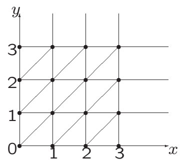
2. must execute transitions *in same order*
consider transitions a with $x \geq 2$ and b with $y \geq 3$.
from state $(s, x = 1.5, y = 2.7)$ can execute b before a
from state $(s, x = 1.4, y = 2.3)$ can execute a before b
⇒ states are not equivalent
⇒ clocks must have same ordering for fractional parts

Region graph. Definition

[Alur & Dill '90]: Define $v \simeq v'$ if:
– $\forall x \in C \,.\, \lfloor v(x) \rfloor = \lfloor v'(x) \rfloor \vee (\lfloor v(x) \rfloor \geq c_x \wedge \lfloor v'(x) \rfloor \geq c_x)$ where $c_x \in \mathbb{Z}$ is largest constant with which x is compared in automaton
(integer parts of clocks are either equal in both assignments or both exceed largest constant)
– $\forall x, y \in C$, $\lfloor v(x) \rfloor < c_x, \lfloor v(y) \rfloor < c_y$, $\{v(x)\} \leq \{v(y)\} \Leftrightarrow \{v'(x)\} \leq \{v'(y)\}$
(fractional parts of clocks have same order in both assignments)
– $\forall x \in C$ cu $\lfloor v(x) \rfloor \leq c$, $\{v(x)\} = 0 \Leftrightarrow \{v'(x)\} = 0$

⇒ *region* associated with state (s, v) = set of states (s, v') with $v \simeq v'$.
⇒ representation with finite number of equivalence classes

region graph (cont'd)



Ex.: region graph for two clocks and maximal constant $c = 3$

Regions are:
– 0-dimensional: points with integer coordinates, $x, y \in \{0, 1, 2, 3\}$
– one-dimensional: segments/diagonals; open-ended segments (≥ 3)
– two-dimensional: bounded (triangles) or not (rectangular stripes)

Fro mtwo states (points) in the same region:
– can execute same transitions
– by passage of time, same regions are traversed

Model checking for the region graph

[Alur, Courcoubetis, Dill '90]
For the timed automaton A, define finite-state automaton $R(A)$:
– states of $R(A)$ are *regions*
– there is a transition between r and r' if and only if
r' is the *successor* region of r with respect to time passage
there is an action transition $(s, v) \xrightarrow{a} (s', v')$ between two representatives (timed states) $(s, v) \in r$ and $(s', v') \in r'$

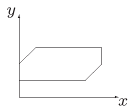
Can prove: TCTL model checking for a timed automaton reduces to CTL model checking for the region graph
(with additional clocks for time bounds on operators)

Size of region graph: bounded by $|C|! \cdot 2^{|C|} \prod_{x \in C} (2c_x + 2)$
– exponential in number of clocks
– exponential in value of maximal time constant

Finite representation with timed zones

Region graph: exponential in number of clocks
⇒ often costly to build and analyze
⇒ alternative representation with temporal inequalities

timed zone = condition from $\mathcal{B}(C)$
ex. $x \leq 5 \wedge 1 \leq y \leq 3 \wedge -2 \leq x - y \leq 3$
(represents a convex polyhedron in hyperspace $\mathbb{R}^{|C|}$)



⇒ a zone = a convex union of regions

Graph of timed zones

Consider zones which are maximal with respect to passage of time in a location (up to time limit imposed by invariant)
⇒ initial zones: $\langle s_0, I(s_0) \wedge \bigwedge_{i \neq j}(x_i = x_j)\rangle$ with $s_0 \in S_0$, $x_i, x_j \in C$

Successors of a zone ϕ by a combined action + time transition:
– conjunct with guard g of transition: $\phi \wedge g$
– reset clocks associated with transition $\phi[x \leftarrow 0] = \exists x \phi \wedge (x = 0)$
(existential quantification over $x \in R$, and then conjunction with $x = 0$)
– take into account passage of time: $\phi\Uparrow = \exists t > 0 \,.\, \phi(v - t)$
(eliminate inequalities $x \prec d$)
– impose the invariant of destination state (conjunct with $I(s')$)

Overall:

$$\phi' = (\phi \wedge g)[R \leftarrow 0] \Uparrow \wedge I(s')$$

Representing zones: difference bound matrices

A zone = conjunction of inequalities $x - y \prec c$, $x \prec c$ or $c \prec x$
⇒ can be represented as square matrix of size $|C| + 1$
(one line for each clock and one line for comparing with zero)
Matrix elements are integers from interval $[-c, c]$:
value d for element (x, y) $(x, y \in C)$ means $x - y \leq d$
(plus one bit to distinguish strict from non-strict inequality)
first line and column: for comparisons with 0

To obtain a finite number of zones: same observation as for regions (concerning maximal time constant)
$x \prec d$ pentru $d > c_{max}$ becomes $x \prec \infty$
$x \prec d$ pentru $d < -c_{max}$ becomes $x \prec -c_{max}$

Working with difference bound matrices

Example: $x \leq 5 \wedge 1 \leq y \wedge -2 \leq x - y \leq 3$

	0	x	y
0	≤ 0	≤ 0	≤ -1
x	≤ 5	≤ 0	≤ 3
y	$\leq \infty$	≤ 2	≤ 0

– conjunction between two zones: smallest of two matrix elements
+ relaxation to propagate clock constraints
(like shortest paths in graph)
– resetting a clock: copy line/column 0 into line/column for clock
– passage of time: set limits $x \prec c$ to $x \prec \infty$

Timed verifiers (e.g., UPPAAL, KRONOS) use this representation or optimized variants thereof