Refinement of Trace Abstraction for Real-Time Programs

September 9, 2017

Franck Cassez[2], Peter G. Jensen[1,2] and Kim G. Larsen[1]
pgj@cs.aau.dk

Department of Computer Science, Aalborg University

Department of Computing, Macquarie University

AALBORG UNIVERSITY
DENMARK

MACQUARIE
University

Setting of Talk

Modelchecking

- Generic framework for Timed Systems
- Verification of Reachability/Safety Properties
- Synthesis of Reachable/Safe parameter sets

Trace Abstraction Refinement
Overview

- Consider system as two parts
  - Control Flow Graph (CFG)
  - “Semantics” instructions as constraint-systems
- Check system one (abstract) trace at a time
  - The CFG is our coarsest abstraction
  - Refine CFG

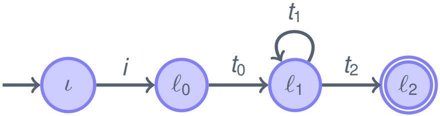
Conditions

- System has to be in CFG/Semantics form
- We need methods for;
  - encoding of trace as constraint-system (*Enc*),
  - checking satisfiability of constraint-system (Z3),
  - generalizing unsatisfiable traces, and
  - refining abstraction.

Real-Time Programs
Motivation

- Plethora of formalisms
  - Time(d) (Arc) Petri-Net,
  - Timed Automata,
  - Hybrid Automata,
  - Timed Process Algebras,
  - . . . ,
- Trace Abstraction Refinement origins from program-verification,
- Decouple control-flow and semantics

Real-Time Programs
Example



Edge	Guard	Update	Rate
i	`true`	`x:=y:=z:=0`	`dy/dt=1`
t_0	`true`	`z:=0`	`dy/dt=0`
t_1	`x==1`	`x:=0`	`dy/dt=0`
t_2	`x-y>=1 and z<1`	-	`dy/dt=0`

Notice
Because we are only concerned with **Reachability**, invariants can be seen as guards.

Real-Time Programs

Preliminaries

Let V be a set of real-valued variables

- $\nu : V \to \mathbb{R}$ is a valuation,
  - the set of valuations is $[V \to \mathbb{R}]$
- $\beta(V)$ is a set of constraints on V,
  - $\nu \models \varphi$ when $\varphi(\nu) = \mathit{True}$ for $\varphi \in \beta(V)$
- $\mathcal{U}(V)$ be the set of updates on the variables in V,
  - $\mu \subseteq [V \to \mathbb{R}] \times [V \to \mathbb{R}]$for $\mu \in \mathcal{U}(V)$,
- $\mathcal{R}(V) \subseteq \mathbb{Q}^V$ be the set of rates

Let $\mathcal{I} = \beta(V) \times \mathcal{U}(V) \times \mathcal{R}(V)$ denote the set of instructions.

Real-Time Programs
Semantics

Let $\nu : V \to \mathbb{R}$ and $\nu' : V \to \mathbb{R}$ be two valuations over the variables. For each pair $(\alpha, \delta) \in \mathcal{I} \times \mathbb{R}_{\geq 0}$ we define the following transition relation:

$$\nu \xrightarrow{\alpha,\delta} \nu' \iff \begin{cases} 1. & \nu \models \gamma_\alpha \text{(guard is satisfied in } \nu\text{)}, \\ 2. & \exists \nu'' \text{ s.t. } (\nu, \nu'') \in \mu_\alpha \text{ (discrete update) and} \\ 3. & \nu' = \nu'' + \delta \times \rho_\alpha \text{(continuous update).} \end{cases}$$

Real-Time Programs
Semantics

The semantics of $\alpha \in \mathcal{I}$ is a mapping $[\![\alpha]\!] : [V \to \mathbb{R}] \to [V \to \mathbb{R}]$ that can be extended to sets of valuations as follows:

$$\nu \in [V \to \mathbb{R}], \quad [\![\alpha]\!](\nu) = \{\nu' \mid \exists \delta \geq 0, \nu \xrightarrow{\alpha,\delta} \nu'\}$$
$$K \subseteq [V \to \mathbb{R}], \quad [\![\alpha]\!](K) = \bigcup_{\nu \in K} [\![\alpha]\!](\nu).$$

We inductively define the *post operator Post* as follows:

$$Post(K, \epsilon) = K$$
$$Post(K, \alpha.w) = Post([\![\alpha]\!](K), w)$$

Real-Time Programs
Formal

A Real-Time Program is a pair $P = (A_P, [\![\cdot]\!])$ where

- $A_P = (Q, \iota, I, \Delta, F)$ is a finite automaton defining the control-flow graph (CFG) and
  - Q is the set of states,
  - $\iota \in Q$ is the initial state,
  - $\mathcal{I}$ is a set of labels (instructions),
  - $\Delta \subseteq Q \times I \times Q$ is the transition-relation, and
  - F is a set of accepting states.
- $[\![\cdot]\!]$ gives semantics to each instruction.

Traces
Feasibility

Timed Word

A *timed word* (over alphabet $\mathcal{I}$) is a finite sequence $\sigma = (\alpha_0, \delta_0).(\alpha_1, \delta_1). \cdots .(\alpha_n, \delta_n)$ such that for each $0 \leq i \leq n$, $\delta_i \in \mathbb{R}_{\geq 0}$ and $\alpha_i \in \mathcal{I}$.

The timed word σ is *feasible* if and only if there exists a set of valuations $\{\nu_0, \ldots, \nu_{n+1}\} \subseteq [V \to \mathbb{R}]$ such that:

$$\nu_0 \xrightarrow{\alpha_0, \delta_0} \nu_1 \xrightarrow{\alpha_1, \delta_1} \nu_2 \cdots \nu_n \xrightarrow{\alpha_n, \delta_n} \nu_{n+1}.$$

Traces
Feasibility cont'd

Let $Unt(\sigma) = \alpha_0.\alpha_1.\cdots.\alpha_n$ be the *untimed* version of σ.

Lemma

An untimed word $w \in \mathcal{I}^*$ is feasible iff $Post(True, w) \neq False$.

Checking Feasibility

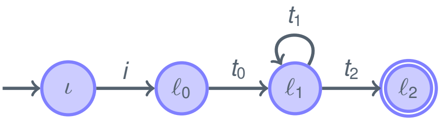
Assume $Enc(w) \in \beta(V^{\mathbb{N}})$ then w is feasible iff there exists ν s.t. $\nu \models Enc(w)$.

Traces
Complexity

- If the trace can be encoded in a decidable theory, checking the trace is decidable.
- Linear Hybrid Automata traces can be encoded in Linear Real Arithmetic (LRA).
- SAT of LRA is decidable – essentially Linear Programming.
- Even if theory is not decidable, we can be lucky.
- Off-the-shelf solvers such as Z3.

Real-Time Programs

Example



Edge	Guard	Update	Rate
i	`true`	`x:=y:=z:=0`	`dy/dt=1`
t_0	`true`	`z:=0`	`dy/dt=0`
t_1	`x==1`	`x:=0`	`dy/dt=0`
t_2	`x-y>=1 and z<1`	`-`	`dy/dt=0`

$Enc(i.t_0.t_2) =$

$$x_0 = y_0 = z_0 = \delta_0 \wedge \delta_0 \geq 0$$

$$x_1 = x_0 + \delta_1 \wedge y_1 = y_0 \wedge z_1 = \delta_1 \wedge \delta_1 \geq 0$$

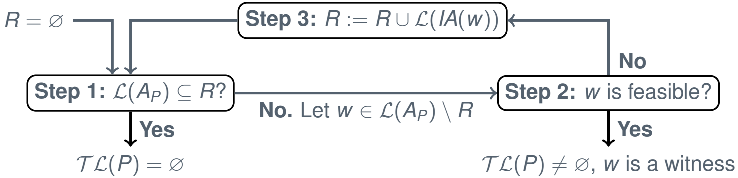
$$x_1 - y_1 \geq 1 \wedge z_1 < 1 \wedge x_2 = x_1 + \delta_2 \wedge y_2 = y_1 \wedge z_2 = z_1 + \delta_2 \wedge \delta_2 \geq 0$$

Trace Abstraction Refinement
Overview

Conditions

- System has to be in CFG/Semantics form ✓
- We need methods for;
  - encoding of trace as constraint-system (*Enc*), ✓
  - checking satisfiability of constraint-system (Z3), ✓
  - generalizing unsatisfiable traces, and
  - refining abstraction.

TAR
Algorithm



Trace Abstraction Refinement Semi-Algorithm for Real-Time Programs

TAR
Generalization of Infeasibility

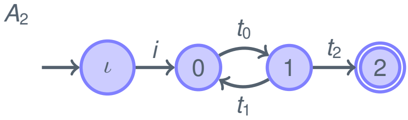
Consider an infeasible word w over the program $(A_P, [\![\cdot]\!])$ then we can

- we can encode w as a conjunction of constraint-systems $c = C_0 \wedge \cdots C_n$ where, for $0 \leq m \leq n$ we have C_m is the encoding of the effect of instruction i_m,
- check feasibility using a solver
- construct *Craig*-interpolants using an interpolanting solver (as Z3).

Craig Interpolant

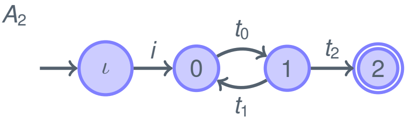
A Craig-interpolant is a sequence of *sufficient* conditions for showing unsatisfiability of a constraint-system.

TAR
Example



Edge	Guard	Update
i	`true`	`x:=y:=k:=0`
t_0	$x \geq 1$	—
t_1	`true`	`x:=0; k++`
t_2	$y < k$	—

TAR
Example



Edge	Guard	Update
i	`true`	`x:=y:=k:=0`
t_0	$x \geq 1$	—
t_1	`true`	`x:=0; k++`
t_2	$y < k$	—

Consider an **infeasible** word w_n for $n > 1$ of the form $i.t_0.(t_1.t_0)^n.t_2$, encoded as $c =$

$$x_0 = y_0 = k_0 = 0 \wedge \quad \delta_0 \geq 0 \wedge x_1 = x_0 + \delta_0 \wedge y_1 = y_0 + \delta_0 \bigwedge$$

$$x_1 \geq 1 \wedge \quad \delta_1 \geq 0 \wedge x_2 = x_1 + \delta_1 \wedge y_2 = y_1 + \delta_1 \bigwedge$$

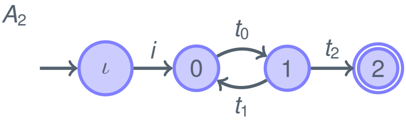
$$x_3 = 0 \wedge k_1 = k_0 + 1 \quad \delta_2 \geq 0 \wedge x_4 = x_3 + \delta_2 \wedge y_3 = y_2 + \delta_2 \bigwedge$$

$$x_4 \geq 1 \wedge \quad \delta_3 \geq 0 \wedge x_5 = x_4 + \delta_3 \wedge y_4 = y_3 + \delta_3 \bigwedge$$

$$\cdots \bigwedge y_n < k_m$$

TAR

Example



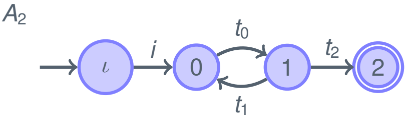
Edge	Guard	Update
i	`true`	`x:=y:=k:=0`
t_0	$x \geq 1$	—
t_1	`true`	`x:=0; k++`
t_2	$y < k$	—

Consider an **infeasible** word w_n for $n > 1$ of the form $i.t_0.(t_1.t_0)^n.t_2$ If we give c to Z3, we get the following interpolants (modulo indexes)

1. $I_0 = y \geq x \wedge k \leq 0$,
2. $I_1 = y \geq 1 \wedge k \leq 0$,
3. $I_2 = y \geq k + x$,
4. $I_3 = y \geq k + 1$,
5. $I_4 = y \geq k + x$,
6. $I_5 = y \geq k + 1$,
7. $\ldots$

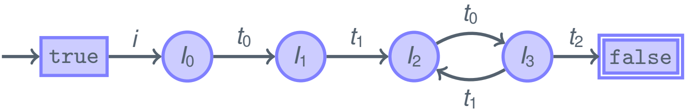
Notice that for $n > 4$ we have $I_n = I_{n+2}$.

TAR
Example



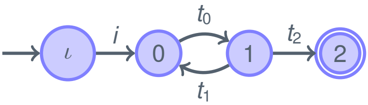
Edge	Guard	Update
i	`true`	`x:=y:=k:=0`
t_0	$x \geq 1$	—
t_1	`true`	`x:=0; k++`
t_2	$y < k$	—

Consider an **infeasible** word w_n for $n > 1$ of the form $i.t_0.(t_1.t_0)^n.t_2$ from this we can construct the *Interpolant* automaton $IA(w_n)$ accepting only infeasible words



TAR
Example

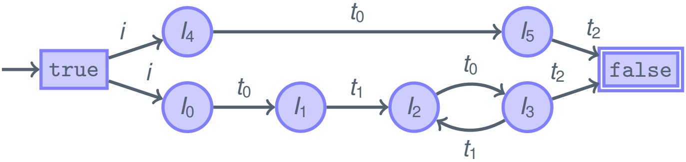
A_2



Edge	Guard	Update
i	`true`	`x:=y:=k:=0`
t_0	$x \geq 1$	—
t_1	`true`	`x:=0; k++`
t_2	$y < k$	—

Same construction for $w_0 = i.t_0.t_2$,

By doing a simple union we have $\mathcal{L}(IA(w_n)) \cup \mathcal{L}(IA(w_0)) \supseteq \mathcal{L}(A_2)$



TAR
Ups and Downs

Bad News

- Undecidable in general, and
- calling SMT-solvers are expensive.

Good News

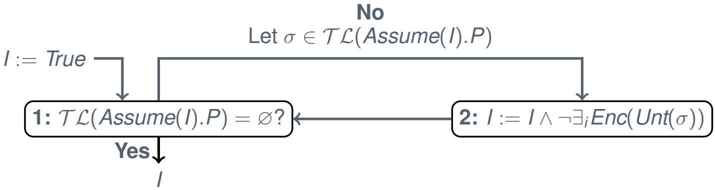
- Works for any encoding within a theory the solver supports,
  - Timed Automata, Stopwatch Automata, Time(d)(-Arc) Petri Nets, Hybrid Automata are all in the decidable theory of Linear Real Arithmetic (Linear Programs)
- Abstracts both continuous and discrete parts of the system, and
- Early termination - even on undecidable things.

TAR

Beyond Reachability

But Wait! There is more!

Existential quantification over Linear Real arithmetic falls withing the theory of linear real arithmetic via Fourier–Motzkin-elimination; hence we can do parameter-synthesis.



Parameter-set synthesized is the largest safe parameter-set.

Experiments
Stopwatch Automata

On the two example shown,

- UPPAAL over-approximates both, and returns unknown/error.
- PHAVER and IMITATOR only computes the first example, and never terminate.

Experiments
Robustness of Timed Automata

Test	Time	$\epsilon <$	Time	$\epsilon <$
	SYMROB		RTTAR	
csma_05	0.43	$1/3$	68.23	$1/3$
csma_06	2.44	$1/3$	227.15	$1/3$
csma_07	8.15	$1/3$	1031.72	$1/3$
fischer_04	0.16	$1/2$	45.24	$1/2$
fischer_05	0.65	$1/2$	249.45	$1/2$
fischer_06	3.71	$1/2$	1550.89	$1/2$
M3c	4.34	$250/3$	43.10	∞
M3	**N/A**	**N/A**	43.07	∞
a	27.90	$1/4$	15661.14	$1/2$

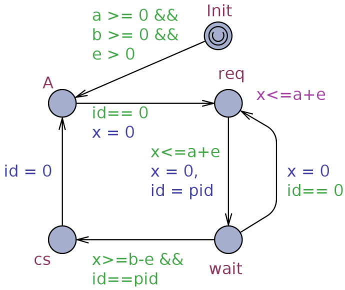
Results for robustness analysis comparing RTTAR with SYMROB. Time is given in seconds. N/A indicates that SYMROB was unable to compute the robustness for the given model.

Experiments
Parameter Synthesis

Test	IMITATOR	RTTAR
Sched2.50.0	201.95	1656.00
Sched2.100.0	225.07	656.26
A_1	DNF	0.1
fischer_2	DNF	0.23
fischer_4	DNF	40.13
fischer_2_robust	DNF	0.38
fischer_4_robust	DNF	118.11

Results for parameter-synthesis comparing RTTAR with IMITATOR. Time is given in seconds. DNF marks that the tool did not complete the computation within an hour.

Experiments
Showing Off



Proposition

For which a, b and ϵ can we guarantee that no two instances are in $\mathtt{cs}$ at the same time?

Answer

$\epsilon \leq 0 \lor a < 0 \lor b < 0 \lor b - a - 2\epsilon > 0$

Conclusion

- Very big hammer,
- slow but exact,
- room for improvement and novel techniques,
- good supplement to existing tools,
- extends family of models solvable.

Further Work

- Function-calls,
- Reductions,
- continuous dynamics,
- liveness-propperties.