Composing Families of Timed Automata

Guillermina Cledou   José Proença   Luis Barbosa

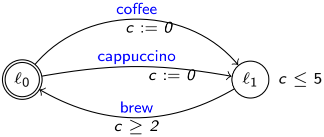
ARCA, Nov 2016
Universidade do Minho

Feature Timed Automata

- Extends Timed Automata (TA) to models families of TA
- Associates boolean expressions, called feature expressions, to transitions

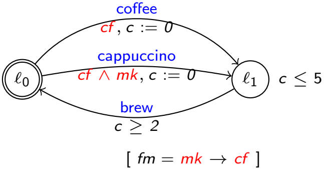
Feature Timed Automata

- Extends Timed Automata (TA) to models families of TA
- Associates boolean expressions, called feature expressions, to transitions



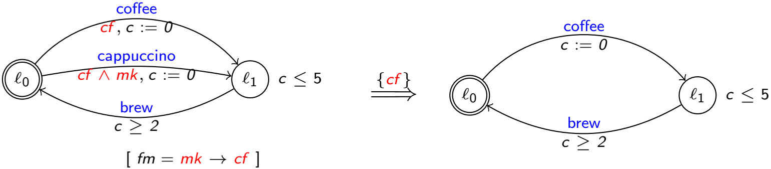
Feature Timed Automata

- Extends Timed Automata (TA) to models families of TA
- Associates boolean expressions, called feature expressions, to transitions

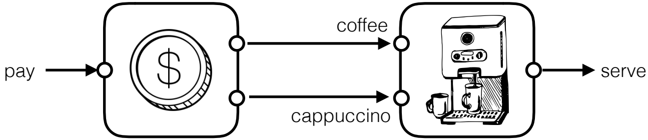


Feature Timed Automata

- Extends Timed Automata (TA) to models families of TA
- Associates boolean expressions, called feature expressions, to transitions

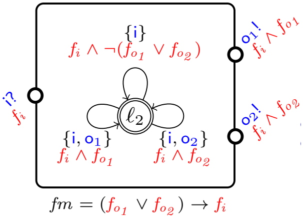


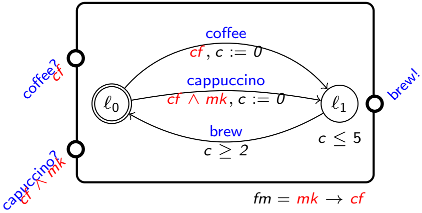
Towards Interface Feature Timed Automata



Interface Feature Timed Automta

- Extends FTA with *interfaces* and *multi-action transitions*.



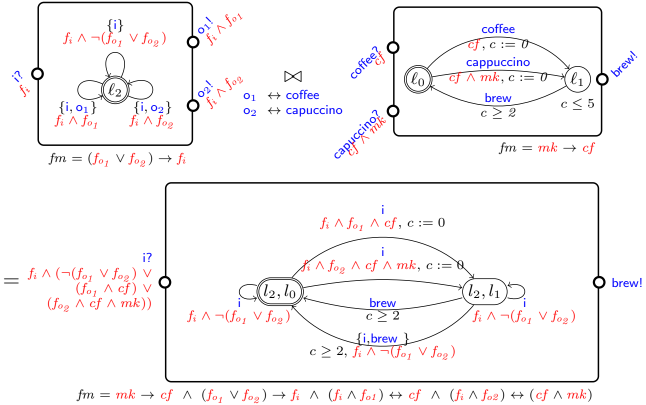


- ?,! denote inputs and outputs interfaces, respectively.
- each interface has associated an inferred *feature expression*.

Interface Feature Timed Automata

Operations over IFTA:

- Product: $\mathcal{A}1 \times \mathcal{A}2$
- Synchronization: $\Delta_{a,b}(\mathcal{A})$
- Composition $= \mathcal{A}_1 \bowtie_{a_1 \leftrightarrow b_1, \dots, a_n \leftrightarrow b_n} \mathcal{A}_2 = \Delta_{a_1,b_1} \dots \Delta_{a_n,b_n}(\mathcal{A}_1 \times \mathcal{A}_2)$

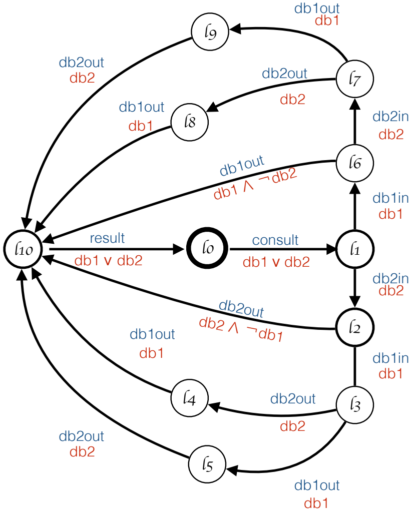


{i}
$f_i \wedge \neg(f_{o_1} \vee f_{o_2})$
i?
f_i
ℓ_2
{i, o1}
$f_i \wedge f_{o_1}$
{i, o2}
$f_i \wedge f_{o_2}$
o1!
$f_i \wedge f_{o_1}$
o2!
$f_i \wedge f_{o_2}$
$fm = (f_{o_1} \vee f_{o_2}) \rightarrow f_i$
$\bowtie$
o1 ↔ coffee
o2 ↔ capuccino
coffee?
cf
capuccino?
$cf \wedge mk$
coffee
$cf, c := 0$
cappuccino
$cf \wedge mk, c := 0$
ℓ_0
ℓ_1
brew
$c \geq 2$
$c \leq 5$
brew!
$fm = mk \rightarrow cf$
$=$
i?
$f_i \wedge (\neg(f_{o_1} \vee f_{o_2}) \vee$
$(f_{o_1} \wedge cf) \vee$
$(f_{o_2} \wedge cf \wedge mk))$
i
$f_i \wedge f_{o_1} \wedge cf, c := 0$
i
$f_i \wedge f_{o_2} \wedge cf \wedge mk, c := 0$
l_2, l_0
l_2, l_1
i
$f_i \wedge \neg(f_{o_1} \vee f_{o_2})$
brew
$c \geq 2$
i
$f_i \wedge \neg(f_{o_1} \vee f_{o_2})$
{i,brew }
$c \geq 2, f_i \wedge \neg(f_{o_1} \vee f_{o_2})$
brew!
$fm = mk \rightarrow cf \wedge (f_{o_1} \vee f_{o_2}) \rightarrow f_i \wedge (f_i \wedge f_{o1}) \leftrightarrow cf \wedge (f_i \wedge f_{o2}) \leftrightarrow (cf \wedge mk)$

Motivation - FTA not enough

Example: call two databases, DB1 and DB2, and wait for their results

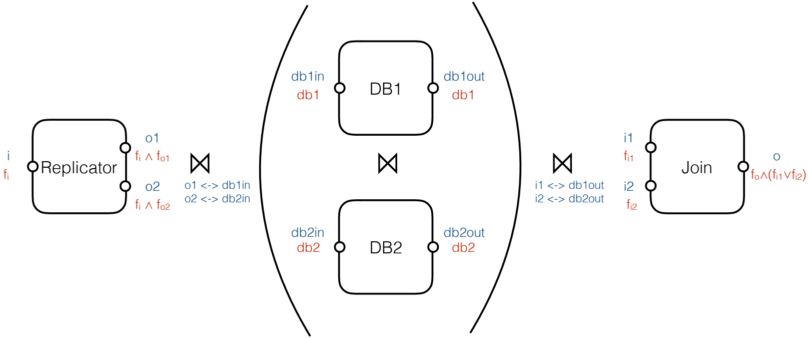
- using FTA and usual modeling approach



Motivation - FTA not enough

Example: call two databases, DB1 and DB2, and wait for their results

- using IFTA

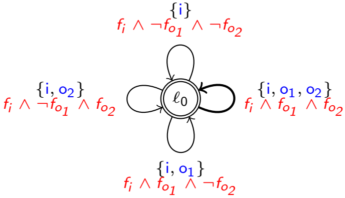


$$
\begin{gathered}
fm = (f_o \rightarrow (f_{i_1} \vee f_{i_2})) \wedge (f_o \rightarrow (f_{i_1} \vee f_{i_2})) \wedge \\
((f_{o_1} \wedge f_i) \leftrightarrow db1) \wedge ((f_{o_2} \wedge f_i) \leftrightarrow db2) \wedge \\
(db1 \leftrightarrow f_{i_1}) \wedge (db2 \leftrightarrow f_{i_2})
\end{gathered}
$$

Motivation - FTA not enough

Example: call two databases, DB1 and DB2, and wait for their results

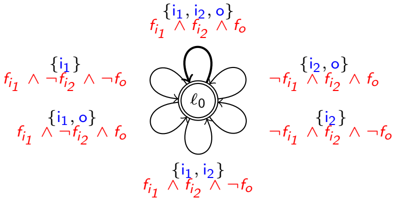
- using IFTA



$\{i\}$
$f_i \wedge \neg f_{o_1} \wedge \neg f_{o_2}$
$\{i, o_2\}$
$f_i \wedge \neg f_{o_1} \wedge f_{o_2}$
ℓ_0
$\{i, o_1, o_2\}$
$f_i \wedge f_{o_1} \wedge f_{o_2}$
$\{i, o_1\}$
$f_i \wedge f_{o_1} \wedge \neg f_{o_2}$

$fm = (f_{o_1} \vee f_{o_2}) \rightarrow f_i$

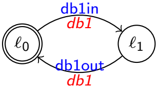
Replicator



$\{i_1, i_2, o\}$
$f_{i_1} \wedge f_{i_2} \wedge f_o$
$\{i_1\}$
$f_{i_1} \wedge \neg f_{i_2} \wedge \neg f_o$
$\{i_2, o\}$
$\neg f_{i_1} \wedge f_{i_2} \wedge f_o$
ℓ_0
$\{i_1, o\}$
$f_{i_1} \wedge \neg f_{i_2} \wedge f_o$
$\{i_2\}$
$\neg f_{i_1} \wedge f_{i_2} \wedge \neg f_o$
$\{i_1, i_2\}$
$f_{i_1} \wedge f_{i_2} \wedge \neg f_o$

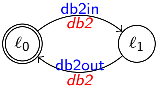
$fm = f_o \rightarrow (f_{i_1} \vee f_{i_2})$

Join



$fm = \top$

DB1



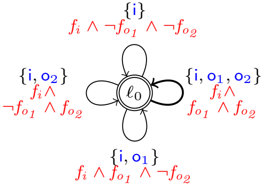
$fm = \top$

DB2

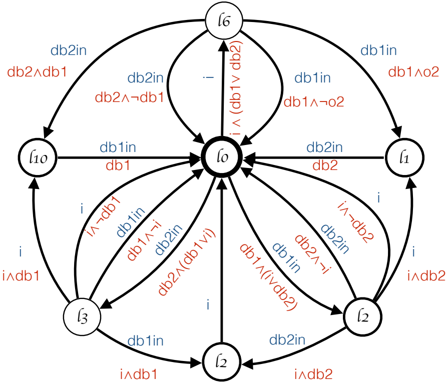
Motivation - FTA not enough

Example: call two databases, DB1 and DB2, and wait for their results

- using FTA and a modular approach


{i}
$f_i \wedge \neg f_{o_1} \wedge \neg f_{o_2}$
{i, o2}
$f_i \wedge$
$\neg f_{o_1} \wedge f_{o_2}$
ℓ_0
{i, o1, o2}
$f_i \wedge$
$f_{o_1} \wedge f_{o_2}$
{i, o1}
$f_i \wedge f_{o_1} \wedge \neg f_{o_2}$

$fm = (f_{o_1} \vee f_{o_2}) \rightarrow f_i$



Replicator

Implementation

Scala DSL:

- Specification of IFTA and networks of IFTA (NIFTA)
- Product, synchronization and composition over IFTA and NIFTA
- NIFTA to networks of FTA (FTA) (with committed states)
- NFTA to Uppaal network of TA
- Visualization in DOT and Vis.js (interactive)

Demo: `https://github.com/joseproenca/ifta`

Discussion

Advantages over FTA:

- Multi-action transitions simplify design
- Interfaces:
  - automatic reasoning about variability during composition
  - makes easier to see how the automata can interact with others
- Composition:
  - composes feature model
  - facilitates modular approach

Limitations:

- Uppaal doesn't work very well with sequence of committed states
- Size of IFTA composition can growth quickly

Discussion

Advantages over FTA:

- Multi-action transitions simplify design
- Interfaces:
  - automatic reasoning about variability during composition
  - makes easier to see how the automata can interact with others
- Composition:
  - composes feature model
  - facilitates modular approach

Limitations:

- Uppaal doesn't work very well with sequence of committed states
- Size of IFTA composition can growth quickly

Questions?