Branching bisimulation reduction of imperative process algebras

Rob van Glabbeek (Data61 & UNSW)

21 June 2018

Imperative process algebras

An imperative process algebra combines a process algebraic language with instructions from imperative programming languages, notably assignment.

Example:

$$
\begin{array}{rl}
P ::= & \delta \mid \varepsilon \mid a(\mathsf{exp}_1, \ldots, \mathsf{exp}_n) \mid P{\cdot}Q \mid P + Q \mid P \| Q \mid \partial_H(P) \mid \tau_I(P) \\
& X(\mathsf{exp}_1, \ldots, \mathsf{exp}_n) \qquad\qquad\qquad\qquad \text{where } \; X \stackrel{\text{def}}{=} P \\
& [\![\mathtt{var} := \mathsf{exp}]\!] \mid \\
& \textbf{while } \varphi \textbf{ do } P \textbf{ od} \mid \\
& \textbf{if } \varphi \textbf{ then } P \textbf{ else } Q \textbf{ fi}
\end{array}
$$

$$\mathsf{exp} ::= 0 \mid 1 \mid \mathsf{exp}_1 + \mathsf{exp}_2 \mid \mathsf{exp}_1 + \mathsf{exp}_2 \mid \mathtt{var} \qquad (\mathtt{var} \in \mathit{Vars})$$

$$\varphi ::= (\mathsf{exp}_1 {=} \mathsf{exp}_2) \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \forall \mathtt{var}.\varphi$$

Example: Algebra of Wireless Networks (AWN) [Fehnker et al.'12]
Example: E-LOTOS [ISO'01] and LNT [Garavel et al.'17]

Semantics of imperative process algebras

A state is given by a pair of a process algebraic expression, modelling the control state, and a valuation of the variables maintained by represented process.

$$\xi, a(\mathsf{exp}).P \xrightarrow{a(\xi(\mathsf{exp}))} \xi, P$$

$$\xi, [\![\mathtt{var} := \mathsf{exp}]\!].P \xrightarrow{\tau} \xi[\mathtt{var} := \xi(\mathsf{exp})], P$$

This gives rise to a transition system of which the transitions are labelled by actions, and the states by valuations.

Doubly labelled transition systems

An L^2TS (over *Act* and **AP**) [DV95] is a triple $(S, \rightarrow, \mathscr{L})$ with

- S a set (of *states*),
- $\rightarrow \subseteq S \times Act \times S$, and
- $\mathscr{L} : S \rightarrow \mathscr{P}(\mathbf{AP})$.

The special case that $\mathbf{AP} = \emptyset$ yields an LTS.
The special case that $|Act| = 1$
yields a Kripke structure.

The transition systems that arise as the semantics of imperative process algebras can be seen as L^2TSs $(S, \rightarrow, \mathscr{L})$ with states of the form (ξ, P), and $\mathscr{L}(\xi, P) = \xi$.

Doubly labelled transition systems

An L^2TS (over *Act* and **AP**) [DV95] is a triple $(S, \rightarrow, \mathscr{L})$ with

- S a set (of *states*),
- $\rightarrow \subseteq S \times Act \times S$, and
- $\mathscr{L} : S \rightarrow \mathscr{P}(\mathbf{AP})$.

The special case that $\mathbf{AP} = \emptyset$ yields an LTS.
The special case that $|Act| = 1$ (and each state has an outgoing transition) yields a Kripke structure.

The transition systems that arise as the semantics of imperative process algebras can be seen as L^2TSs $(S, \rightarrow, \mathscr{L})$ with states of the form (ξ, P), and $\mathscr{L}(\xi, P) = \xi$.

Branching bisimilarity

A *branching bisimulation* on an $\mathrm{L}^2\mathrm{TS}$ is a symmetric binary relation $\mathscr{R} \subseteq S \times S$ such that

- if $s \mathscr{R} t$ and $s \xrightarrow{\alpha} s'$ then $\exists t^{\mathrm{pre}}, t'$ with $t \Longrightarrow t^{\mathrm{pre}} \xrightarrow{(\alpha)} t'$, $s \mathscr{R} t^{\mathrm{pre}}$ and $s' \mathscr{R} t'$,
- and if $s \mathscr{R} t$ then $\mathscr{L}(s) = \mathscr{L}(t)$.

s and t are *branching bisimilar*, $s \leftrightarroweq_b t$, if there exists a branching bisimulation $\mathscr{R}$ with $s \mathscr{R} t$.

Here $\Longrightarrow$ is the reflexive-transitive closure of $\xrightarrow{\tau}$ and $t \xrightarrow{(\alpha)} u$ means $t \xrightarrow{\alpha} u \vee (\alpha = \tau \wedge t = u)$.

Restricted to LTSs this is standard branching bisimilarity [GW96].
Restricted to Kripke str. this is divergence-blind stuttering equiv.

Divergence-preserving branching bisimilarity

A *divergence-preserving branching bisimulation* on an L^2TS is a symmetric binary relation $\mathscr{R} \subseteq S \times S$ such that

- if $s \mathscr{R} t$ and $s \xrightarrow{\alpha} s'$ then $\exists t^{\mathrm{pre}}, t'$ with $t \Longrightarrow t^{\mathrm{pre}} \xrightarrow{(\alpha)} t'$, $s \mathscr{R} t^{\mathrm{pre}}$ and $s' \mathscr{R} t'$,
- and if $s \mathscr{R} t$ then $\mathscr{L}(s) = \mathscr{L}(t)$,
- if $s \mathscr{R} t$ and $s = s_0 \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \dots$ then $\exists t'$ with $t \xrightarrow{\tau} t'$ and $s_k \mathscr{R} t'$ for some $k \geq 0$.

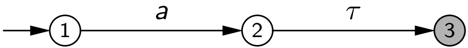
s and t are *divergence-preserving branching bisimilar*, $s \leftrightarrow_b^\Delta t$, if there exists a d-p branching bisimulation $\mathscr{R}$ with $s \mathscr{R} t$.

Here $\Longrightarrow$ is the reflexive-transitive closure of $\xrightarrow{\tau}$ and $t \xrightarrow{(\alpha)} u$ means $t \xrightarrow{\alpha} u \vee (\alpha = \tau \wedge t = u)$.

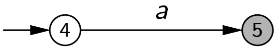
Restricted to LTSs this is standard d-p br. bis. [GW96,GLT09].
Restricted to Kripke str. this is standard stuttering equiv. [BCG88].

Termination predicates versus temporal logic predicates

In standard process algebra (ACP, CSP): $a = a.\tau$.

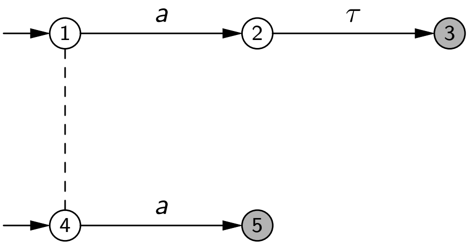


.



Termination predicates versus temporal logic predicates

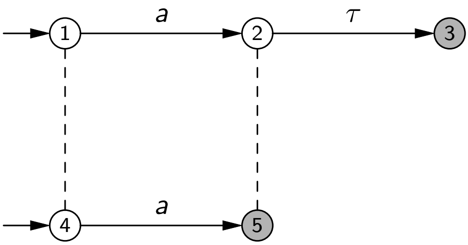
In standard process algebra (ACP, CSP): $a = a.\tau$.



.

Termination predicates versus temporal logic predicates

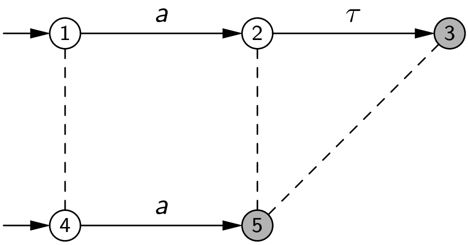
In standard process algebra (ACP, CSP): $a = a.\tau$.



.

Termination predicates versus temporal logic predicates

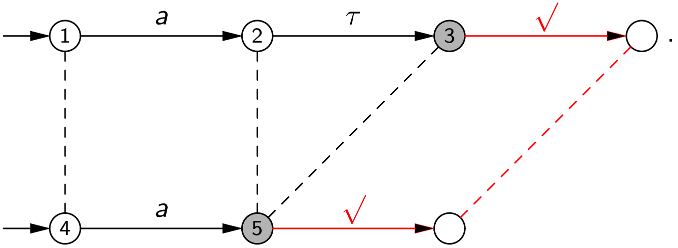
In standard process algebra (ACP, CSP): $a = a.\tau$.



.

Termination predicates versus temporal logic predicates

In standard process algebra (ACP, CSP): $a = a.\tau$.

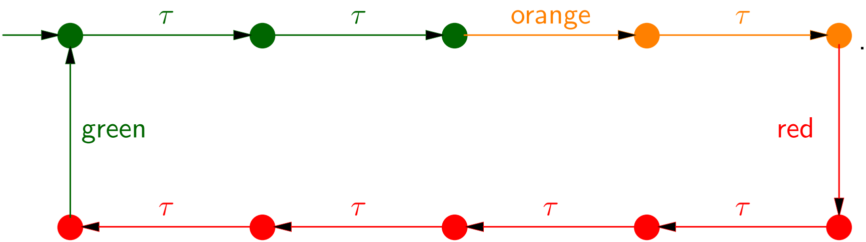


LTL and CTL on LTSs

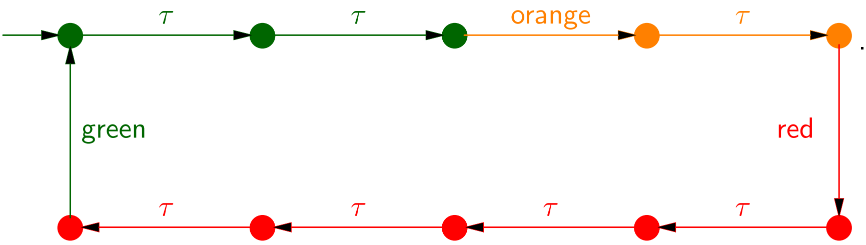
$$\text{LTS} \xrightarrow{\eta} \text{KS} \models \varphi$$

$s \models_\eta \varphi$

Action versus state-based modelling

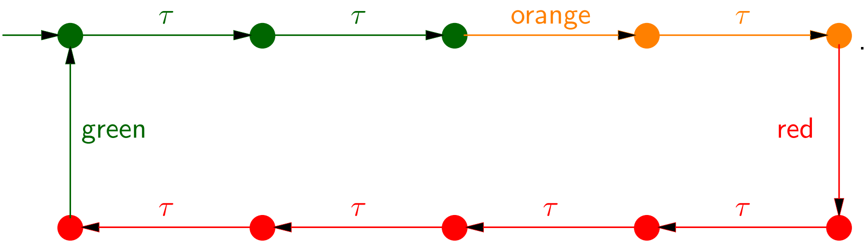


Action versus state-based modelling



LTL and CTL without next-state operator: LTL_{-X} and CTL_{-X}

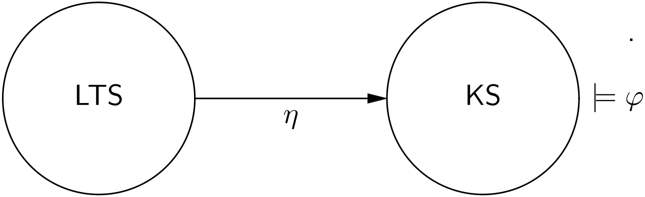
Action versus state-based modelling



LTL and CTL without next-state operator: LTL_{-X} and CTL_{-X}

On Kripke structures: $s \,\underline{\leftrightarrow}_b^\Delta\, t$ iff $\forall \varphi \in \mathrm{CTL}_{-X}(s \models \varphi \Leftrightarrow t \models \varphi)$.

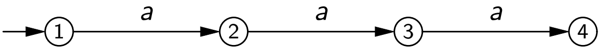
LTL and CTL on LTSs



$s \models_\eta \varphi$

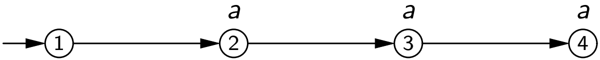
η should be such that $s \leftrightarroweq^\Delta_b t \Leftrightarrow \eta(s) \leftrightarroweq^\Delta_b \eta(t)$.

Converting LTSs to KSs

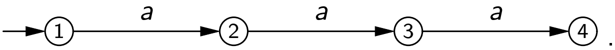


.

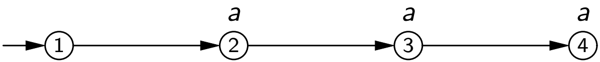
moving right:



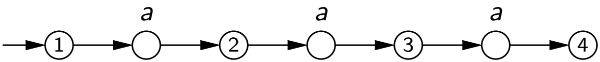
Converting LTSs to KSs



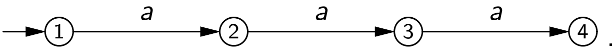
moving right:



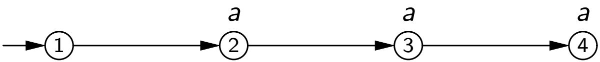
[DV95]:



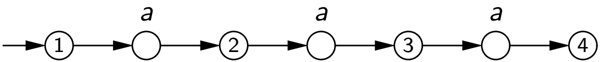
Converting LTSs to KSs



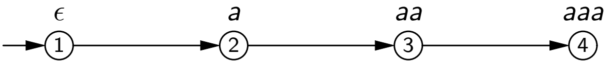
moving right:



[DV95]:



[vGV06]:



Converting LTSs to KSs

De Nicola - Vaandrager translation:
Insert a fresh state halfway each visible transition; move its label to that state.
Do not insert fresh states halfway τ-transitions; drop τ-labels.

Voorhoeve translation:
Unwind process graph (= LTS plus initial state) into a tree.
Label each state with the sequence of visible actions on the unique path leading to that state.

Preservation of equivalence upon conversion LTS to KS

$$s \underline{\leftrightarrow}_b^{\Delta} t \Leftrightarrow \eta_{DV}(s) \underline{\leftrightarrow}_b^{\Delta} \eta_{DV}(t)$$

$$s \underline{\leftrightarrow}_b t \Leftrightarrow \eta_{DV}(s) \underline{\leftrightarrow}_b \eta_{DV}(t)$$

Preservation of equivalence upon conversion LTS to KS

$$s \leftrightarroweq_b^\Delta t \Leftrightarrow \eta_{DV}(s) \leftrightarroweq_b^\Delta \eta_{DV}(t)$$

$$s \leftrightarroweq_b t \Leftrightarrow \eta_{DV}(s) \leftrightarroweq_b \eta_{DV}(t)$$

$$s \leftrightarroweq_w t \not\Leftrightarrow \eta_{DV}(s) \leftrightarroweq_w \eta_{DV}(t)$$

Preservation of equivalence upon conversion LTS to KS

$$s \underline{\leftrightarrow}_b^{\Delta}\ t \Leftrightarrow \eta_{DV}(s) \underline{\leftrightarrow}_b^{\Delta}\ \eta_{DV}(t)$$

$$s \underline{\leftrightarrow}_b\ t \Leftrightarrow \eta_{DV}(s) \underline{\leftrightarrow}_b\ \eta_{DV}(t)$$

$$s \underline{\leftrightarrow}_w\ t \not\Leftrightarrow \eta_{DV}(s) \underline{\leftrightarrow}_w\ \eta_{DV}(t)$$

$$s \underline{\leftrightarrow}_b^{\Delta}\ t \Leftrightarrow \eta_{V}(s) \underline{\leftrightarrow}_b^{\Delta}\ \eta_{V}(t)$$

$$s \underline{\leftrightarrow}_b\ t \Leftrightarrow \eta_{V}(s) \underline{\leftrightarrow}_b\ \eta_{V}(t)$$

$$s \underline{\leftrightarrow}_w\ t \Leftrightarrow \eta_{V}(s) \underline{\leftrightarrow}_w\ \eta_{V}(t)$$

Converting L^2TSs to KSs

Both translations work equally well for L^2TSs.

Blocking actions

$$X \stackrel{def}{=} coin.coffee.X$$

$\mathbf{G}(coin \Rightarrow \mathbf{F}coffee)$

Blocking actions

$$X \stackrel{def}{=} coin.coffee.X$$

$\mathbf{G}(coin \Rightarrow \mathbf{F} coffee)$

Valid under *progress assumption*: one doesn't stop without reason.

Blocking actions

$$X \stackrel{def}{=} coin.coffee.X$$

$\mathbf{G}(coin \Rightarrow \mathbf{F} coffee)$

Valid under *progress assumption*: one doesn't stop without reason.
Progress assumption built in in the definition of LTL/CTL

Blocking actions

$$X \stackrel{def}{=} coin.coffee.X$$

$\mathbf{G}(coin \Rightarrow \mathbf{F}coffee)$ $\mathbf{G}(coffee \Rightarrow \mathbf{F}coin)$

Valid under *progress assumption*: one doesn't stop without reason.
Progress assumption built in in the definition of LTL/CTL

Blocking actions

$$X \stackrel{def}{=} coin.coffee.X$$

$\mathbf{G}(coin \Rightarrow \mathbf{F}coffee)$ $\mathbf{G}(coffee \Rightarrow \mathbf{F}coin)$

Valid under *progress assumption*: one doesn't stop without reason.
Progress assumption built in in the definition of LTL/CTL

Conclusion: standard LTL / CTL not suitable for reactive systems

Blocking actions

$$X \stackrel{def}{=} coin.coffee.X$$

$\mathbf{G}(coin \Rightarrow \mathbf{F}coffee)$ $\mathbf{G}(coffee \Rightarrow \mathbf{F}coin)$

Valid under *progress assumption*: one doesn't stop without reason.
Progress assumption built in in the definition of LTL/CTL

Conclusion: standard LTL / CTL not suitable for reactive systems

Solution: Postulate a set B of *blocking* actions

Blocking actions

$$X \stackrel{def}{=} coin.coffee.X$$

$\mathbf{G}(coin \Rightarrow \mathbf{F}coffee)$ $\mathbf{G}(coffee \Rightarrow \mathbf{F}coin)$

Valid under *progress assumption*: one doesn't stop without reason.
Progress assumption built in in the definition of LTL/CTL

Conclusion: standard LTL / CTL not suitable for reactive systems

Solution: Postulate a set B of *blocking* actions

coin could be blocking, and *coffee* not, for instance

Blocking actions

$$X \stackrel{def}{=} coin.coffee.X$$

$\mathbf{G}(coin \Rightarrow \mathbf{F}coffee)$ $\mathbf{G}(coffee \Rightarrow \mathbf{F}coin)$

Valid under *progress assumption*: one doesn't stop without reason.
Progress assumption built in in the definition of LTL/CTL

Conclusion: standard LTL / CTL not suitable for reactive systems

Solution: Postulate a set B of *blocking* actions

coin could be blocking, and *coffee* not, for instance

$X \models_B \mathbf{G}(coin \Rightarrow \mathbf{F}coffee)$ $X \not\models_B \mathbf{G}(coffee \Rightarrow \mathbf{F}coin)$

Blocking actions

$$X \stackrel{def}{=} coin.coffee.X$$

$\mathbf{G}(coin \Rightarrow \mathbf{F}coffee)$ $\mathbf{G}(coffee \Rightarrow \mathbf{F}coin)$

Valid under *progress assumption*: one doesn't stop without reason. Progress assumption built in in the definition of LTL/CTL

Conclusion: standard LTL / CTL not suitable for reactive systems

Solution: Postulate a set B of *blocking* actions

coin could be blocking, and *coffee* not, for instance

$X \models_B \mathbf{G}(coin \Rightarrow \mathbf{F}coffee)$ $X \not\models_B \mathbf{G}(coffee \Rightarrow \mathbf{F}coin)$

Instead of evaluating LTL / CTL formulas on *infinite* paths, use *complete* paths.

Here a path is *complete* if it is infinite, or ends in a state whose outgoing transitions all have labels from B.

Summary

This talk:

- Proposed the concept of an imperative process algebra;
- Pointed out its natural semantics is a L^2TS;
- Proposed a definition of (div.-pres) branching bisimilarity on L^2TSs
- — it matches perfectly with CTL_{-X} —;
- Postulated that termination(-like) predicates should be treated different from observable state-properties;
- Reviewed 2 good ways to translate L^2TSs to Kripke structures
- — one preserves (div.-pres) branching bisimilarity; the other all reasonable equivalence —;
- Made CTL and LTL usable for analysing reactive systems.