CCS: Equivalences

Marco Carbone

April 2009

Outline

- Recap
- A Tale of Two Coca-Cola Machines
- About Equivalences
- *Strong* Bisimulation
- The Strong Bisimulation Game
- Properties
- Model-Checking
- CCS Tools

CCS Syntax

- CCS Syntax:

$$P ::= 0 \mid \alpha.P \mid P+P \mid P|P \mid P\backslash a \mid P[f] \mid K$$

- "0" // inaction
- "$\alpha.P$" // action prefix
- "$P+P$" // non-deterministic choice
- "$P|P$" // parallel composition
- "$P\backslash a$" // restriction (private name)
- "$P[f]$" // action relabelling
- "K" // process variable

... where $X \stackrel{def}{=} P,\ Y \stackrel{def}{=} Q,\ \ldots$

Note: ***restrictions on f***

$$f : Act \rightarrow Act$$

$$\forall a:\ f(\overline{a}) = \overline{f(a)} \quad \wedge\ f(\tau) = \tau$$

SOS for CCS

- *Structural Operational Semantics*:

$$[\text{DEF}]\ \frac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'}\ K \stackrel{\text{def}}{=} P \qquad [\text{ACT}]\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad [\text{SUM}]\ \frac{P_j \xrightarrow{\alpha} P'_j}{\sum_{i\in I} P_i \xrightarrow{\alpha} P'_j}\ j \in I$$

$$[\text{COM}_1]\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad [\text{COM}_2]\ \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'} \qquad [\text{COM}_3]\ \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\bar{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

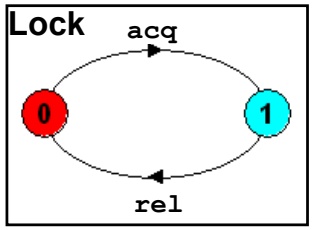
$$[\text{REN}]\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad [\text{RES}]\ \frac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L}\ \alpha, \bar{\alpha} \notin L$$

Q: why τ (tau) in communication "P | Q" (instead of propagating $\bar{a}$ or a) ?

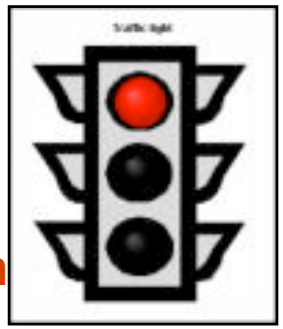
τ ~ "the unobservable hand-shake"

Example: Lock *("Mutual Exclusion")*

- Example: *("mutex")* Lock
  - $Lock \stackrel{def}{=} acq.rel.Lock$
  - $User \stackrel{def}{=} \overline{acq}.enter.exit.\overline{rel}.User$
  - $Mutex \stackrel{def}{=} (User \mid Lock) \setminus \{acq, rel\}$
  - $User' \stackrel{def}{=} User[enter'/enter, exit'/exit]$
  - $System \stackrel{def}{=} (User \mid Lock \mid User') \setminus \{acq, rel\}$



// critical region



aka. a
"semaphore"

  - **Q:** *How are the critical enters and exits related ?*
    - **A:** *never two enters without exit in between*

Historically:

- $Sema \stackrel{def}{=} p.v.Sema$

Prolaag *and* ***Verhoog*** *?* — E. Dijkstra

A TALE OF TWO *COCA-COLA* MACHINES

Keywords:

- equality
- equivalence (equivalence relations)
- congruence

April 2009

Once upon a time...

- Dispenser:
  - $\overline{coin} \,.\, (\overline{coke} + \overline{sprite})$

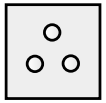
- Dispenser':
  - $\overline{coin}.\overline{coke} + \overline{coin}.\overline{sprite}$

Would you consider them *equal* '=' ?

Would you consider them *equivalent* '≈' ?

What does it at all mean for them to be equivalent '≈' ?!?

Equal vs. Equivalent

- *Equal* (*concrete identity*):
  - 3 = 3
- *Equivalent* (*abstract*):
  - 3 ≈ 003
  - $3_{10} \approx 0x03_{16} \approx \backslash 003_{8} \approx 0011_{2}$
  - 3 ≈ three
  - 3 ≈ 
  - 3 ≈ 1+2
  - 3 ≈ $\sum_{i=0}^{2} i$
  - 3 ≈ `let n=2 in n*(n-1)+(n-2)`

more abstract

Trace Equivalence

- Definition: *Trace Equivalence*:
  - Two processes **P** and **Q** are *trace equivalent* "$\approx_{tr}$" iff:
    - They can produce the same traces:
    - **Traces(P) = { $\omega \in Act^*$ | $\exists$ Q : $P \xrightarrow{\omega}{}^* Q$ }**
- Example:
  - Traces(`coin . (coke + sprite)`)

    = { ε, `coin`, `coin;coke`, `coin;sprite` }

  - Traces(`coin.coke + coin.sprite`)

    = { ε, `coin`, `coin;coke`, `coin;sprite` }

Hence:  $\approx_{tr}$

Contextual Composition…?

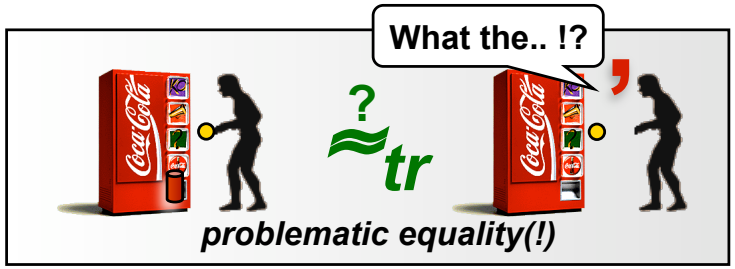
- Recall:
  - $coin \,.\, (\overline{coke} + \overline{sprite})$
  - $coin.\overline{coke} + coin.\overline{sprite}$

Idea (can we…?): put the two dispensers in a *context* where they can be differentiated !

- "Coke-*only*-drinker":
  - $\overline{coin} \,.\, coke \,.\, \overline{drink}$

Trace equivalence cannot distinguish the two dispensers.

- *Contextual composition*:



The coke drinker is certainly able to to distinguish the two dispensers !!

ABOUT EQUIVALENCES

Keywords:

- equivalence relation
- congruence
- properties

CCS: *"Single-Language Formalism"*

- **CCS** is a so-called *"Single-lang. formalism"*;
  - i.e. one may specify both:
    - *implementation* (`IMPL` $=_{def}$ `...`)
    - **and** *specification* (`SPEC` $=_{def}$ `...`)

- We would like to check via some (reasonable) *equivalence*, $\mathcal{R}$, that:
  - *"The implementation has the intended behavior"*:
    - `SPEC` $\mathcal{R}$ `IMPL` *the spec. and impl. are "equivalent"*

Def: Equivalence Relation

- Let $\mathcal{R}$ be a binary relation *over set* A:
  - $\mathcal{R} \subseteq A \times A$

- $\mathcal{R}$ is an *equivalence relation* iff:
  - *Reflexive*:
  - ∧ $\forall x \in A: \quad x \mathcal{R} x$
  - *Symmetric*:
  - ∧ $\forall x,y \in A: \quad x \mathcal{R} y \Leftrightarrow y \mathcal{R} x$
  - *Transitive*:
  - $\forall x,y,z \in A: \quad x \mathcal{R} y \wedge y \mathcal{R} z \Rightarrow x \mathcal{R} z$

Q: is trace equivalence "$\approx_{tr}$" an equivalence relation ?

...and (equivalence relation)

- …and we would like ($\mathcal{R}$ to be an *equiv. rel.*):
  - *reflexitivity* :
    - **SYS** $\mathcal{R}$ **SYS** (same behavior as itself) **!**
  - *transitivity* (for stepwise modelling/refinement) **!!!** :
    - $\mathbf{S_0}\ \mathcal{R}\ \mathbf{S_1}\ \mathcal{R}\ \cdots\ \mathcal{R}\ \mathbf{S_n}\ \mathcal{R}$ **IMPL => S0** $\mathcal{R}$ **IMPL**
  - *symmetry* (just a nice property to have in that):
    - **S** $\mathcal{R}$ **S′ ≡ S′** $\mathcal{R}$ **S**

Is *trace equivalence*, '$\approx_{tr}$', an *equivalence relation* ?	**Yes ☺**

…and (satisfy properties)

- Furthermore, we would like these properties:
  - `P+Q` $\mathcal{R}$ `Q+P` // '`+`' commutative
  - `(P+Q)+R` $\mathcal{R}$ `P+(Q+R)` // '`+`' associative
  - `P|Q` $\mathcal{R}$ `Q|P` // '`|`' commutative
  - `(P|Q)|R` $\mathcal{R}$ `P|(Q|R)` // '`|`' associative
  - `0+P` $\mathcal{R}$ `P` // '`0`' is neutral wrt. '`+`'
  - `0|P` $\mathcal{R}$ `P` // '`0`' is neutral wrt. '`|`'
  - `...`

Trace equivalence '$\approx_{tr}$' ? **Yes ☺**

…and (congruence wrt. CCS)

- Definition: "$\mathcal{R}$" *congruence (wrt. CCS)*:
  - `P` $\mathcal{R}$ `Q` $\Rightarrow$ `C[P]` $\mathcal{R}$ `C[Q]` , for all *contexts* `C[]`
    - *"relation is preserved under contextual substitution"*
- A *context* = a process with a gap:
  - `C : α.[] | []+P | P+[] | []|P | P|[] | [][f] | []\a`
- Examples:
  - `P` $\mathcal{R}$ `Q` $\Rightarrow$ `P+R` $\mathcal{R}$ `Q+R` — `[]+R`
  - `P` $\mathcal{R}$ `Q` $\Rightarrow$ `P|S` $\mathcal{R}$ `P|S` — `[]|S`
  - `P` $\mathcal{R}$ `Q` $\Rightarrow$ `a.P` $\mathcal{R}$ `a.Q` — `a.[]`
  - `P` $\mathcal{R}$ `Q` $\Rightarrow$ `((a.P|R)+S)\x` $\mathcal{R}$ `((a.Q|R)+S)\x` — $?^2$

Congruence '$\approx_{tr}$' ?

Trace Equiv. ~ DFM Acceptance

- Recall: a deterministic finite automaton, A:
  - is completely identified by its *set of traces*: $\mathcal{L}$(A)
- Trace equivalence ~ DFA acceptance:
  - (without accept states - by construction)
  - $P \approx_{tr} Q$ **iff** they can produce the same traces

This point of view is totally justified and natural if we view our LTSs as non-deterministic devices that may generate or accept sequences of actions. However, is it still a reasonable one if we view our automata as reactive machines that *interact with their environment* ?

-- [Aceto, Larsen, Ingólfsdóttir, p. 33]

…and (abstract)

- We would also like $\mathcal{R}$ to:
  - abstract away from:
    - *(particular choices of) states*
    - *non-determinism*
    - *internal (unobservable) behavior*
  - Consider only *behavior* (*observable actions*)

Trace equivalence '$\approx_{tr}$' ?	**No ☹**

'$\approx_{tr}$' doesn't take communication capabilities of intermediate states into account (can make a difference when made to interact w/ another system)

STRONG BISIMULATION: (~)

Def: *A Strong Bisimulation*

- Let (***Proc***, ***Act***, $\xrightarrow{a}$) be a LTS

- **Def**: a bin. rel. $\mathcal{R} \subseteq \mathit{Proc} \times \mathit{Proc}$ is *a strong bisimulation* iff whenever $(s,t) \in \mathcal{R}$: $\forall a \in \mathit{Act}$:
  - if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s',t') \in \mathcal{R}$
  - if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s',t') \in \mathcal{R}$

- Note:
  - 1. Definition on LTS (*not necessarily wrt. processes*)
  - 2. Definition *relative to a (SOS) semantics* (via LTS)

Intuition: ***"Only equate as consistently allowed by the semantics"***

Def: *Strongly Bisimilar* (~)

- *A Strong Bisimulation*:

- **Def**: a bin. rel. $\mathcal{R} \subseteq \mathit{Proc} \times \mathit{Proc}$ is *a strong bisimulation* iff whenever $(s,t) \in \mathcal{R}$: $\forall a \in \mathit{Act}$:
  - if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s',t') \in \mathcal{R}$
  - if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s',t') \in \mathcal{R}$

- *The Strong Bisimilarity relation* (~):

- **Def**: two (processes) s and t are *strongly bisimilar* ($s \sim t$) iff $\exists$ *strong bisimulation* $\mathcal{R}$: $(s,t) \in \mathcal{R}$.
  - i.e. ‘~’ := $\cup\{\mathcal{R} \mid \mathcal{R}$ *is a strong bisimulation*$\}$

Basic Properties of (~)

- **Theorem:**
  - '~' is an *equivalence relation*

- **Theorem:**
  - '~' is *the largest strong bisimulation*
    - i.e. for any *bisimulation* $\mathcal{R}$ we have that: $\mathcal{R} \subseteq$ '~'

- **Theorem:**
  - $s \sim t$ iff $\forall a \in Act$:
    - if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $s' \sim t'$
    - if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $s' \sim t'$

How to Prove Strong Bisimilarity ?

- *How to prove* strong bisimilarity for two processes **?**
  - i.e. $p \sim q$ **?**:

- *Exhibit a (any) bisimulation* $\mathcal{R}$, for which:
  - $(s,t) \in \mathcal{R}$
    - By definition we get that:
      - $(s,t) \in \mathcal{R} \subseteq$ '$\sim$' since '$\sim$' was the *largest* bisimulation

Example Proof of Bisimilarity

- Example:
  - Buffer (capacity 1):

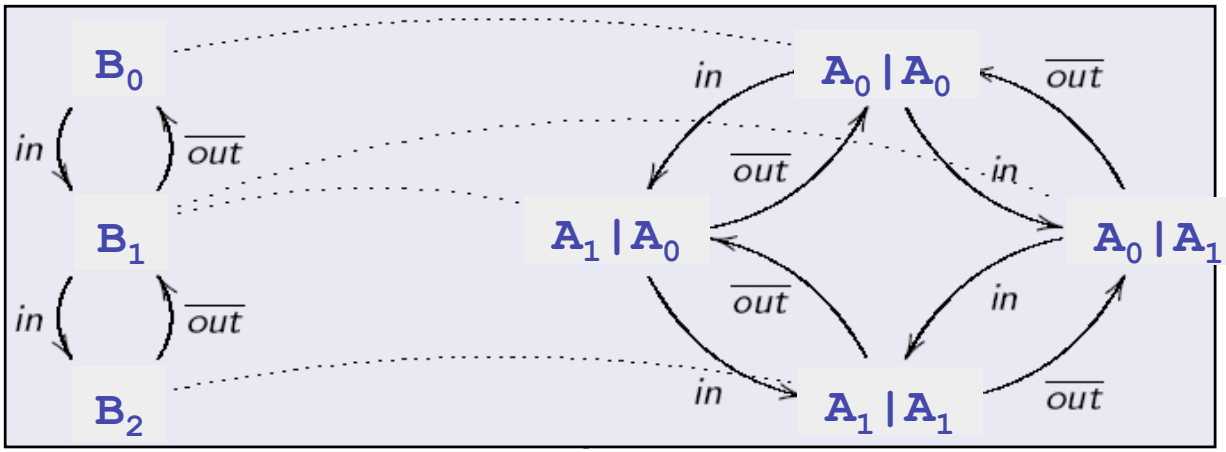
$$A_0 =_{def} \underline{in} \,.\, A_1$$
$$A_1 =_{def} \overline{out} \,.\, A_0$$

  - Buffer (capacity 2):

$$B_0 =_{def} in \,.\, B_1$$
$$B_1 =_{def} \underline{in} \,.\, B_2 + \overline{out} \,.\, B_0$$
$$B_2 =_{def} \overline{out} \,.\, B_1$$

  - Show that: $B_0 \sim A_0 \mid A_0$

$$R = \{ (B_0 , A_0 \mid A_0) , (B_1 , A_1 \mid A_0) , (B_1 , A_0 \mid A_1) , (B_2 , A_1 \mid A_1) \}$$



Example Proof of Bisimilarity

Consider the processes

$$\mathtt{Cl} \stackrel{\text{def}}{=} \mathtt{tick.Cl}$$

$$\mathtt{Cl}_2 \stackrel{\text{def}}{=} \mathtt{tick.tick.Cl}_2$$

$B_1 = \{(\mathtt{Cl}, \mathtt{Cl}_2)\}$ is not a bisimulation.

$B_2 = \{(\mathtt{Cl}, \mathtt{Cl}_2), (\mathtt{Cl}, \mathtt{tick.Cl}_2)\}$ is a bisimulation.

Exercise Prove that B_2 is a bisimulation.

Example Proof of Bisimilarity

$$\begin{aligned}
\mathtt{Sem} &\stackrel{\text{def}}{=} \mathtt{get}.\mathtt{Sem}' \\
\mathtt{Sem}' &\stackrel{\text{def}}{=} \mathtt{put}.\mathtt{Sem}
\end{aligned}$$

$$\begin{aligned}
\mathtt{Sem2}_0 &\stackrel{\text{def}}{=} \mathtt{get}.\mathtt{Sem2}_1 \\
\mathtt{Sem2}_1 &\stackrel{\text{def}}{=} \mathtt{get}.\mathtt{Sem2}_2 + \mathtt{put}.\mathtt{Sem2}_0 \\
\mathtt{Sem2}_2 &\stackrel{\text{def}}{=} \mathtt{put}.\mathtt{Sem2}_1
\end{aligned}$$

$$\begin{aligned}
B = \{ \quad & (\mathtt{Sem2}_0, \mathtt{Sem} \mid \mathtt{Sem}), \\
& (\mathtt{Sem2}_1, \mathtt{Sem}' \mid \mathtt{Sem}), \\
& (\mathtt{Sem2}_1, \mathtt{Sem} \mid \mathtt{Sem}'), \\
& (\mathtt{Sem2}_2, \mathtt{Sem}' \mid \mathtt{Sem}') \quad \}
\end{aligned}$$

is a bisimulation.

Example Proof of Bisimilarity

The processes $\mathtt{a}.(\mathtt{b}.0 + \mathtt{c}.0)$ and $\mathtt{a}.\mathtt{b}.0 + \mathtt{a}.\mathtt{c}.0$ are not bisimilar.

Assume there is a bisimulation B containing the pair

$$(\mathtt{a}.(\mathtt{b}.0 + \mathtt{c}.0), \mathtt{a}.\mathtt{b}.0 + \mathtt{a}.\mathtt{c}.0)$$

Then B also contains

$$((\mathtt{b}.0 + \mathtt{c}.0), \mathtt{b}.0)$$

The left process can do a $\mathtt{c}$, but the right one cannot. Contradiction.

$a.0$	$a.a.0$
$a.0$	$a.0 + a.0$
$a.0$	$a.0 \mid a.0$
$a.a.0$	$a.0 \mid a.0$
$a.b.0$	$a.0 \mid b.0$
$a.b.0 + b.a.0$	$a.0 \mid b.0$
$a.\overline{a}.0 + \overline{a}.a.0$	$a.0 \mid \overline{a}.0$
$a.\overline{a}.0 + \overline{a}.a.0 + \tau.0$	$a.0 \mid \overline{a}.0$
$\tau.0$	new $a\,(a.0 \mid \overline{a}.0)$

Which ones are bisimilar?

How to Prove Non-Bisimilarity ?

- *How to prove* non-bisimilarity **?**
  - i.e. $s \not\sim t$ **?**
- *Enumerate all binary relations*:
  - Check that none are bisimulations and contain (p,q)
    - **However**: *extremely expensive* $O(2^{|p||q|})$
- Use *"Feynman Problem-Solving Algorithm"*:
  - (1). Write down the problem;
  - (2). Think very hard;
  - (3). Write down the answer. ☺
- Or…

(~) Bisimulation Games

April 2009

The (Strong) Bisimulation Game

- Let (***Proc***, ***Act***, $\xrightarrow{a}$) be a LTS and $s, t \in$ ***Proc***

- Define *2-player game*: [*attacker* v *defender*]
  - The game is played in "*rounds*" and the *configurations* of the game are (***Proc*** × ***Proc***);
    - The game starts (first round) in $(s,t) \in$ ***Proc*** × ***Proc***

- Intuition (objectives):
  - The *defender* wants to show that: $s \sim t$
  - The *attacker* wants to show that: $s \not\sim t$

Rules of the Bisimulation Game

- In round k the players change the current configuration (s_k, t_k) as follows:
  - First, the *attacker chooses*:
    - 1) one of the processes (e.g. t_k); i.e., *left* or *right*;
    - 2) a legal action from that process: $a \in \mathbf{Act}$;
    - 3) a legal transition according to the LTS: $t_k \xrightarrow{a} t_{k+1}$
  - Then, the *defender chooses*:
    - -) a *"counter-move"* using *same action*, a: $s_k \xrightarrow{a} s_{k+1}$
  - (s_{k+1}, t_{k+1}) becomes the next round's configuration…

- Winning:
  - If one player (only) cannot move, the other player wins
  - If the game is infinite (repeats configuration), the defender wins

Game Characterization of (~)

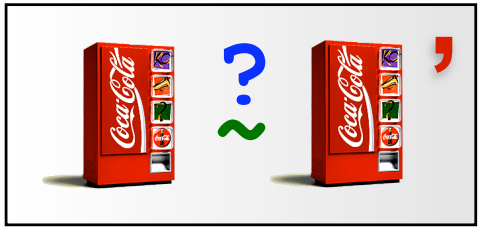
- **Theorem:**
  - States (processes) `s` and `t` *are **not** strongly bisimilar*
    iff the *attacker* has a ***universal winning strategy***
  - States (processes) `s` and `t` *are strongly bisimilar*
    iff the *defender* has a ***universal winning strategy***

- (`s` ~ `t`) basically means that:
  *"the '**perfect attacker**' always wins"*

- (`s ~ t`) basically means that:
  *"the '**perfect defender**' always wins"*

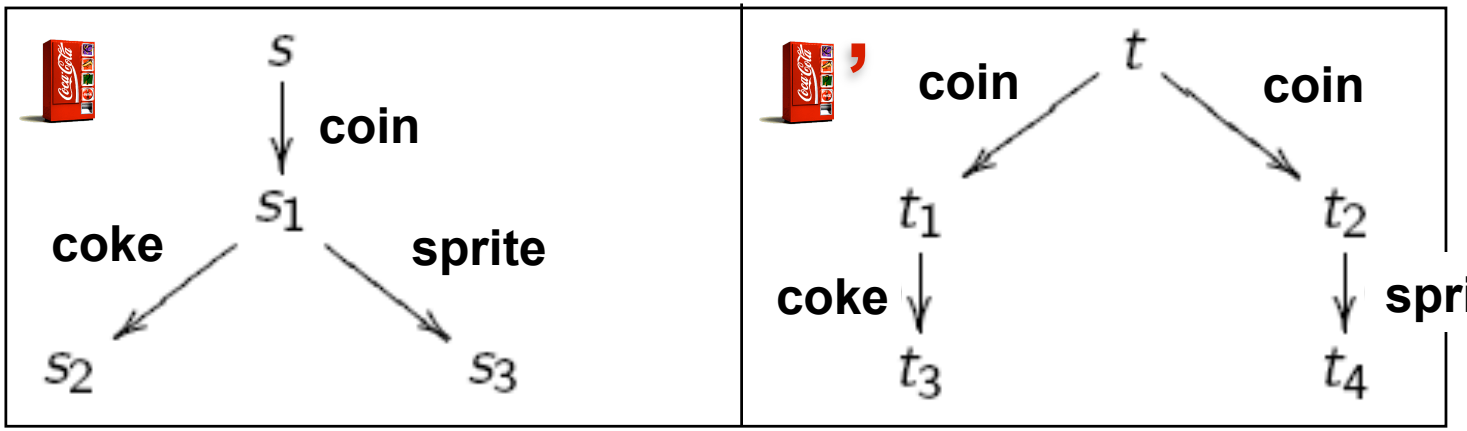
Let's Play…

- Let's play…:



$\mathtt{coin}\ .\ (\overline{\mathtt{coke}}\ +\ \overline{\mathtt{sprite}})$

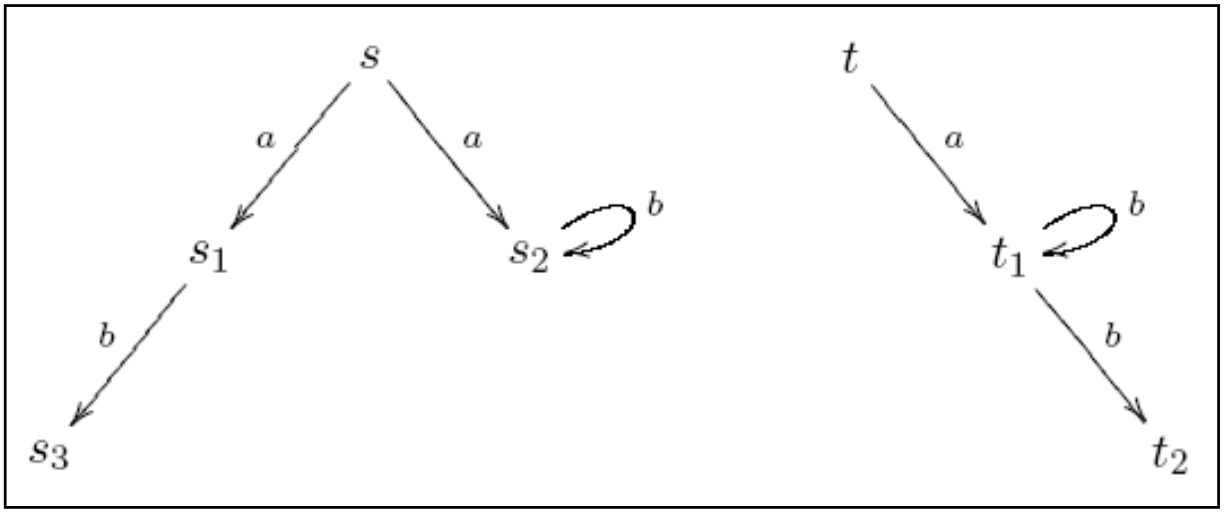
$\mathtt{coin}.\overline{\mathtt{coke}}\ +\ \mathtt{coin}.\overline{\mathtt{sprite}}$



- ~ / ≁ ? show of hands…

Another Game…

- Are the following two LTS(/processes) **`s`** and **`t`** *strongly bisimilar*: **`s ~ t`** ?



- There's a *universal attack strategy* ⇒ hence, they are ***not*** *strongly bisimilar* : **`s`** ≁ **`t`**

(~) CONGRUENCE

April 2009

(~) is a Congruence for CCS

- **Theorem:**
  - Let `P` and `Q` be processes such that `P ~ Q`; then:
    - `α.P ~ α.Q` $\quad \forall \alpha \in \mathit{Act}$
    - `P+R ~ Q+R` $\quad \forall R \in \mathit{Proc}$
    - `R+P ~ R+Q` $\quad \forall R \in \mathit{Proc}$
    - `P|R ~ Q|R` $\quad \forall R \in \mathit{Proc}$
    - `R|P ~ R|Q` $\quad \forall R \in \mathit{Proc}$
    - `P[f] ~ Q[f]` $\quad \forall f : P(\mathit{Act}) \rightarrow P(\mathit{Act})$ relabellings
    - `P\a ~ Q\a` $\quad \forall a \in \mathit{Act} \setminus \{\tau\}$

- i.e. '~' is a congruence for CCS

The Expansion Law

The expansion law states that a process with parallel compositions is bisimilar to a (possibly huge) process without parallel compositions

$$P_1 \sim a.P_{11} + b.P_{12} + a.P_{13}$$

$$P_2 \sim \overline{a}.P_{21} + c.P_{22},$$

$$\begin{aligned} P_1|P_2 \sim\ & a.(P_{11}|P_2) + b.(P_{12}|P_2) + a.(P_{13}|P_2) \\ & + \overline{a}.(P_1|P_{21}) + c.(P_1|P_{22}) \\ & + \tau.(P_{11}|P_{21}) + \tau.(P_{13}|P_{21}) \end{aligned}$$

The Expansion Law (2)

$$P_1 \mid \ldots \mid P_m \sim$$

$$\sum\{\alpha_{ij}.Q_{ij} : 1 \leq i \leq m \text{ and } 1 \leq j \leq n_i\} +$$

$$\sum\{\tau.Q_{klij} : 1 \leq k < i \leq m \text{ and } \alpha_{kl} = \overline{\alpha}_{ij} \neq \tau\},$$

$$Q_{ij} = P_1 \mid \ldots \mid P_{i-1} \mid P_{ij} \mid P_{i+1} \mid \ldots \mid P_m$$

$$Q_{klij} = P_1 \mid \ldots \mid P_{k-1} \mid P_{kl} \mid P_{k+1} \mid$$
$$\ldots \mid P_{i-1} \mid P_{ij} \mid P_{i+1} \mid \ldots \mid P_m.$$

($\alpha_{kl} = \overline{\alpha}_{ij} \neq \tau$ means $\alpha_{kl} = a$ and $\alpha_{ij} = \overline{a}$ or $\alpha_{ij} = a$ and $\alpha_{kl} = \overline{a}$, for action name a)

Other Properties of (~)

- The following properties hold ∀P, Q, R:
  - `P+Q ~ Q+P` // '+' commutative
  - `(P+Q)+R ~ P+(Q+R)` // '+' associative
  - `P|Q ~ Q|P` // '|' commutative
  - `(P|Q)|R ~ P|(Q|R)` // '|' associative
  - `P+0 ~ P` // '0' neutral wrt. '+'
  - `P|0 ~ P` // '0' neutral wrt. '|'
  - `...`

Summary: Strong Bisimilarity (~)

- Properties of (~):
  - an equivalence relation:
    - reflexive, symmetric, and transitive
  - the largest strong bisimulation:
    - for proving bisimilarity (exhibit a bisimulation)
  - strong bisimulation game:
    - for proving non-bisimilarity (winning attack strategy)
  - a congruence:
    - `P ~ Q  =>  C[P] ~ C[Q]`
  - obeys the following algebraic laws:
    - '`+`' and '`|`' commutative, associative, and '`0`' neutrality, …

"Three minutes paper"

- Please spend three minutes writing down the most important things that you have learned today (***now***).