Classical View

Characterization of a Classical Program

Program transforms an input into an output.

- Denotational semantics:
  a meaning of a program is a partial function

$$\mathit{states} \hookrightarrow \mathit{states}$$

- Nontermination is bad!
- In case of termination, the result is unique.

Is this all we need?

{ x_1, x_2 are integers satisfying C_1: $x_1 \geq 0, x_2 > 0$ }

Program P

$y_1 := 0; y_2 := x_1;$
  { $x_1 = y_1 x_2 + y_2 \wedge 0 \leq y_2$ } ... INV
while $y_2 \geq x_2$ **do** $(y_1 := y_1 + 1; y_2 := y_2 - x_2);$
$z_1 := y_1; z_2 := y_2$

{ C_2: $x_1 = z_1 x_2 + z_2 \wedge 0 \leq z_2 < x_2$ }

We want to verify: $\{C_1\}P\{C_2\}$ ... (specification of P)

Generated verification conditions:

$\{C_1\}\ y_1 := 0; y_2 := x_1\ \{\mathrm{INV}\}$
$\{\mathrm{INV} \wedge y_2 \geq x_2\}\ y_1 := y_1 + 1; y_2 := y_2 - x_2\ \{\mathrm{INV}\}$
$\{\mathrm{INV} \wedge \neg(y_2 \geq x_2)\}\ z_1 := y_1; z_2 := y_2\ \{C_2\}$

What about:

- Operating systems?
- Communication protocols?
- Control programs?
- Mobile phones?
- Vending machines?

Reactive systems

Characterization of Reactive Systems

Reactive System is a system that computes by reacting to stimuli from its environment.

Key Issues:

- communication and interaction
- parallelism

Nontermination is good!

The result (if any) does not have to be unique.

Analysis of Reactive Systems

Questions

- How can we develop (design) a system that "works"?
- How do we analyze (verify) such a system?

Fact of Life

Even short parallel programs may be hard to analyze.

Example: Peterson's protocol

Concurrent, parallel, interactive, 'nondeterministic' systems,
with ongoing behaviour ...
No input-output characterization (specification) ...
Verification of 'simple' properties ...

Peterson's protocol (to avoid critical section clash)

Process A:

** noncritical region **
$flag_A := true$
$turn := B$
waitfor
$(flag_B = false \lor turn = A)$
** critical region **
$flag_A := false$
** noncritical region **

Process B:

** noncritical region **
$flag_B := true$
$turn := A$
waitfor
$(flag_A = false \lor turn = B)$
** critical region **
$flag_B := false$
** noncritical region **

The Need for a Theory

Conclusion

We need formal/systematic methods (tools), otherwise ...

- Intel's Pentium-II bug in floating-point division unit
- Ariane-5 crash due to a conversion of 64-bit real to 16-bit integer
- Mars Pathfinder
- ...

Classical vs. Reactive Computing

	Classical	Reactive/Parallel
interaction	no	yes
nontermination	undesirable	often desirable
unique result	yes	no
semantics	$states \hookrightarrow states$	**?**

How to Model Reactive Systems

Question

What is the most abstract view of a reactive system (process)?

Answer

A process performs an action and becomes another process.

Labelled Transition System

Definition

A labelled transition system (LTS) is a triple $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ where

- $Proc$ is a set of states (or processes),
- Act is a set of labels (or actions), and
- for every $a \in Act$, $\xrightarrow{a} \subseteq Proc \times Proc$ is a binary relation on states called the transition relation.

We will use the infix notation $s \xrightarrow{a} s'$ meaning that $(s, s') \in \xrightarrow{a}$.

Sometimes we distinguish the initial (or start) state.

Interlude: Binary Relations

Definition

A binary relation R on a set A is a subset of $A \times A$.

$$R \subseteq A \times A$$

Sometimes we write $x \; R \; y$ instead of $(x, y) \in R$.

Some properties of relations

- R is reflexive if $(x, x) \in R$ for all $x \in A$
- R is symmetric if $(x, y) \in R$ implies that $(y, x) \in R$ for all $x, y \in A$
- R is transitive if $(x, y) \in R$ and $(y, z) \in R$ implies that $(x, z) \in R$ for all $x, y, z \in A$

Closures

Let R, R' and R'' be binary relations on a set A.

Reflexive Closure

R' is the reflexive closure of R if and only if

1. $R \subseteq R'$,
2. R' is reflexive, and
3. R' is the *smallest* relation that satisfies the two conditions above, which means the following:
   for any relation R'', if $R \subseteq R''$ and R'' is reflexive then $R' \subseteq R''$.

Closures

Let R, R' and R'' be binary relations on a set A.

Symmetric Closure

R' is the symmetric closure of R if and only if

1. $R \subseteq R'$,
2. R' is symmetric, and
3. R' is the *smallest* relation that satisfies the two conditions above.

Closures

Let R, R' and R'' be binary relations on a set A.

Transitive Closure

R' is the transitive closure of R if and only if

1. $R \subseteq R'$,
2. R' is transitive, and
3. R' is the *smallest* relation that satisfies the two conditions above.

Labelled Transition Systems – Notation

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

- we extend $\xrightarrow{a}$ to the elements of Act^*
- $\longrightarrow = \bigcup_{a \in Act} \xrightarrow{a}$
- $\longrightarrow^*$ is the reflexive and transitive closure of $\longrightarrow$
- $s \xrightarrow{a}$ and $s \not\xrightarrow{a}$
- reachable states

How to Describe LTS?

Syntax		Semantics
unknown entity	$\longrightarrow$	known entity
programming language	$\longrightarrow$	what (denotational) or how (operational) it computes
???	$\longrightarrow$	Labelled Transition Systems

CCS

Goals

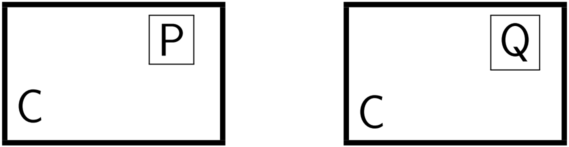
What should a reasonable behavioural equivalence satisfy?

- abstract from states (consider only the behaviour – actions)
- abstract from nondeterminism
- abstract from internal behaviour

What else should a reasonable behavioural equivalence satisfy?

- reflexivity $P \equiv P$ for any process P
- transitivity $Spec_0 \equiv Spec_1 \equiv Spec_2 \equiv \cdots \equiv Impl$ gives that
$$Spec_0 \equiv Impl$$
- symmetry $P \equiv Q$ iff $Q \equiv P$

Congruence



$C(P)$ $C(Q)$

Congruence Property

$P \equiv Q$ implies that $C(P) \equiv C(Q)$

Trace Equivalence

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

Trace Set for $s \in Proc$

$$Traces(s) = \{w \in Act^* \mid \exists s' \in Proc.\ s \xrightarrow{w} s'\}$$

Let $s \in Proc$ and $t \in Proc$.

Trace Equivalence

We say that s and t are trace equivalent ($s \equiv_t t$) if and only if

$$Traces(s) = Traces(t)$$

Is this a "good" behavioural equivalence ?

Black-Box Experiments

Experiment in A	Experiment in B	Experiment in B
$\boxed{coin}$ $\overline{tea}$ $\overline{coffee}$	$\boxed{coin}$ $\overline{tea}$ $\overline{coffee}$	$\boxed{coin}$ $\overline{tea}$ $\overline{coffee}$
press *coin*	press *coin*	press *coin*
$coin$ $\boxed{\overline{tea}}$ $\boxed{\overline{coffee}}$	$coin$ $\boxed{\overline{tea}}$ $\overline{coffee}$	$coin$ $\overline{tea}$ $\boxed{\overline{coffee}}$

Main Idea

Two processes are behaviorally equivalent if and only if an external observer cannot tell them apart.

Strong Bisimilarity

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

Strong Bisimulation

A binary relation $R \subseteq Proc \times Proc$ is a strong bisimulation iff whenever $(s,t) \in R$ then for each $a \in Act$:

- if $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ for some t' such that $(s', t') \in R$
- if $t \xrightarrow{a} t'$ then $s \xrightarrow{a} s'$ for some s' such that $(s', t') \in R$.

Strong Bisimilarity

Two processes $p_1, p_2 \in Proc$ are strongly bisimilar ($p_1 \sim p_2$) if and only if there exists a strong bisimulation R such that $(p_1, p_2) \in R$.

$$\sim \; = \; \cup\{R \mid R \text{ is a strong bisimulation}\}$$

Basic Properties of Strong Bisimilarity

Theorem

$\sim$ *is an equivalence (reflexive, symmetric and transitive)*

Theorem

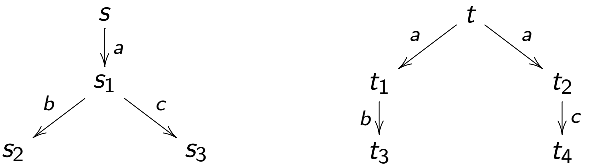
$\sim$ *is the largest strong bisimulation*

Theorem

$s \sim t$ *if and only if for each* $a \in Act$:

- *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 Show Nonbisimilarity?



To prove that $s \not\sim t$:

- Enumerate all binary relations and show that none of them at the same time contains (s, t) and is a strong bisimulation. (Expensive: $2^{|Proc|^2}$ relations.)
- Make certain observations which will enable to disqualify many bisimulation candidates in one step.
- Use game characterization of strong bisimilarity.

Strong Bisimulation Game

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS and $s, t \in Proc$.

We define a two-player game of an 'attacker' and a 'defender' starting from s and t.

- The game is played in rounds and configurations of the game are pairs of states from $Proc \times Proc$.
- In every round exactly one configuration is called current. Initially the configuration (s, t) is the current one.

Intuition

The defender wants the show that s and t are strongly bisimilar while the attacker aims to prove the opposite.

Rules of the Bisimulation Games

Game Rules

In each round the players change the current configuration as follows:

1. the attacker chooses one of the processes in the current configuration and makes an $\xrightarrow{a}$-move for some $a \in Act$, and
2. the defender must respond by making an $\xrightarrow{a}$-move in the other process under the same action a.

The newly reached pair of processes becomes the current configuration. The game then continues by another round.

Result of the Game

- If one player cannot move, the other player wins.
- If the game is infinite, the defender wins.

Game Characterization of Strong Bisimilarity

Theorem

- States s and t are strongly bisimilar if and only if the defender has a universal winning strategy starting from the configuration (s, t).
- States s and t are not strongly bisimilar if and only if the attacker has a universal winning strategy starting from the configuration (s, t).

Remark

Bisimulation game can be used to prove both bisimilarity and nonbisimilarity of two processes. It very often provides elegant arguments for the negative case.