Process Algebras (CCS): a fast tour

- Syntax of CCS
- Labelled transition systems - Semantics of CCS
- Value passing CCS
- Operating with process expressions: Bisimulation Semantics

CCS Basics (Sequential Fragment)

- *Nil* (or 0) process (the only atomic process)
- action prefixing ($a.P$)
- names and recursive definitions ($\stackrel{\mathrm{def}}{=}$)
- nondeterministic choice ($+$)

CCS Basics (Parallelism and Renaming)

- parallel composition ($|$)
  (synchronous communication between two components $=$ handshake synchronization)
- restriction ($P \setminus L$)
- relabelling ($P[f]$)

Definition of CCS (channels, actions, process names)

Let

- $\mathcal{A}$ be a set of channel names (e.g. *tea*, *coffee* are channel names)
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels where
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
    ($\mathcal{A}$ are called names and $\overline{\mathcal{A}}$ are called co-names)
  - by convention $\overline{\overline{a}} = a$
- $Act = \mathcal{L} \cup \{\tau\}$ is the set of actions where
  - τ is the internal or silent action

  (e.g. τ, *tea*, $\overline{coffee}$ are actions)
- $\mathcal{K}$ is a set of process names (constants) (e.g. CM).

Definition of CCS (formal syntax, expressions)

$P :=$	K	$\mid$	process constants ($K \in \mathcal{K}$)
	$\alpha.P$	$\mid$	prefixing ($\alpha \in Act$)
	$\sum_{i \in I} P_i$	$\mid$	summation (I is an arbitrary index set)
	$P_1 \mid P_2$	$\mid$	parallel composition
	$P \setminus L$	$\mid$	restriction ($L \subseteq \mathcal{A}$)
	$P[f]$	$\mid$	relabelling ($f : Act \to Act$) such that

- $f(\tau) = \tau$
- $f(\bar{a}) = \overline{f(a)}$

The set of all terms generated by the abstract syntax is called CCS process expressions (and denoted by $\mathcal{P}$).

Notation

$$P_1 + P_2 = \sum_{i \in \{1,2\}} P_i \qquad Nil = 0 = \sum_{i \in \emptyset} P_i$$

Definition of CCS (defining equations)

CCS program

A collection of defining equations of the form

$$K \stackrel{\text{def}}{=} P$$

where $K \in \mathcal{K}$ is a process constant and $P \in \mathcal{P}$ is a CCS process expression.

- Only one defining equation per process constant.
- Recursion is allowed: e.g. $A \stackrel{\text{def}}{=} \overline{a}.A \mid A$.

Labelled Transition Systems

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.

Sequencing, Nondeterminism and Parallelism

LTS explicitly focuses on interaction.

LTS can also describe:

- sequencing $(a; b)$
- choice (nondeterminism) $(a + b)$
- recursion (by loops)

This is Enough to Describe Finite Space Sequential Processes

Any finite LTS can be (up to isomorphism) described by using the operations above.

Semantics of CCS

Syntax		Semantics
CCS (collection of defining equations)	$\longrightarrow$	LTS (labelled transition systems)

HOW?

Structural Operational Semantics for CCS

Structural Operational Semantics (SOS) – G. Plotkin 1981

Small-step operational semantics where the behaviour of a system is inferred using syntax driven rules.

Given a collection of CCS defining equations, we define the following LTS $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$:

- $Proc = \mathcal{P}$ (the set of all CCS process expressions)
- $Act = \mathcal{L} \cup \{\tau\}$ (the set of all CCS actions including τ)
- transition relation is given by SOS rules of the form:

$$\text{RULE} \quad \frac{premises}{conclusion} \quad conditions$$

SOS rules for CCS ($\alpha \in Act$, $a \in \mathcal{L}$)

$$\text{ACT} \quad \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad\qquad \text{SUM}_j \quad \frac{P_j \xrightarrow{\alpha} P'_j}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_j} \quad j \in I$$

$$\text{COM1} \quad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad\qquad \text{COM2} \quad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

$$\text{COM3} \quad \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\text{RES} \quad \frac{P \xrightarrow{\alpha} P'}{P \smallsetminus L \xrightarrow{\alpha} P' \smallsetminus L} \quad \alpha, \overline{\alpha} \notin L \qquad\qquad \text{REL} \quad \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

$$\text{CON} \quad \frac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'} \quad K \stackrel{\text{def}}{=} P$$

Deriving Transitions in CCS

Let $A \stackrel{\mathrm{def}}{=} a.A$. Then

$$\big((A \,|\, \overline{a}.Nil) \,|\, b.Nil\big)[c/a] \xrightarrow{c} \big((A \,|\, \overline{a}.Nil) \,|\, b.Nil\big)[c/a].$$

$$\text{REL}\ \dfrac{\text{COM1}\ \dfrac{\text{COM1}\ \dfrac{\text{CON}\ \dfrac{\text{ACT}\ \dfrac{}{a.A \xrightarrow{a} A}}{A \xrightarrow{a} A}\ A \stackrel{\mathrm{def}}{=} a.A}{A \,|\, \overline{a}.Nil \xrightarrow{a} A \,|\, \overline{a}.Nil}}{(A \,|\, \overline{a}.Nil) \,|\, b.Nil \xrightarrow{a} (A \,|\, \overline{a}.Nil) \,|\, b.Nil}}{\big((A \,|\, \overline{a}.Nil) \,|\, b.Nil\big)[c/a] \xrightarrow{c} \big((A \,|\, \overline{a}.Nil) \,|\, b.Nil\big)[c/a]}$$

LTS of the Process $a.Nil \mid \overline{a}.Nil$

$$
\begin{tikzcd}
 & a.Nil \mid \overline{a}.Nil \arrow[dl, "a"'] \arrow[dd, "\tau"] \arrow[dr, "\overline{a}"] & \\
Nil \mid \overline{a}.Nil \arrow[dr, "\overline{a}"] & & a.Nil \mid Nil \arrow[dl, "a"'] \\
 & Nil \mid Nil &
\end{tikzcd}
$$

Process Algebras (CCS): a fast tour

- value passing CCS
- translation to standard CCS
- CCS is Turing-powerful

Value Passing CCS

Main Idea

Handshake synchronization is extended with the possibility to exchange integer values.

$$\overline{pay(6)}.Nil \mid pay(x).\overline{save(x/2)}.Nil \mid Bank(100)$$
$$\downarrow \tau$$
$$Nil \mid \overline{save(3)}.Nil \mid Bank(100)$$
$$\downarrow \tau$$
$$Nil \mid Nil \mid Bank(103)$$

Parametrized Process Constants

For example: $Bank(total) \stackrel{\text{def}}{=} save(x).Bank(total + x)$.

Translation of Value Passing CCS to Standard CCS

Value Passing CCS

$$C \stackrel{\text{def}}{=} in(x).C'(x)$$

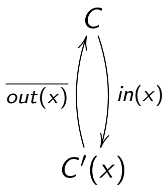
$$C'(x) \stackrel{\text{def}}{=} \overline{out(x)}.C$$

$\longrightarrow$

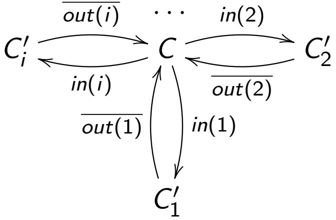
Standard CCS

$$C \stackrel{\text{def}}{=} \sum_{i \in \mathbb{N}} in(i).C'_i$$

$$C'_i \stackrel{\text{def}}{=} \overline{out(i)}.C$$



symbolic LTS



infinite LTS

Behavioural Equivalence

Implementation

$$CM \stackrel{\mathrm{def}}{=} coin.\overline{coffee}.CM$$

$$CS \stackrel{\mathrm{def}}{=} \overline{work}.\overline{coin}.coffee.CS$$

$$Uni \stackrel{\mathrm{def}}{=} (CM \mid CS) \smallsetminus \{coin, coffee\}$$

Specification

$$Spec \stackrel{\mathrm{def}}{=} \overline{work}.Spec$$

Question

Are the processes *Uni* and *Spec* behaviorally equivalent?

$$Uni \equiv Spec \ ?$$

Goals

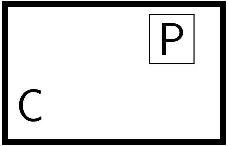
What should a reasonable behavioural equivalence satisfy?

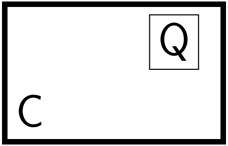
- abstract from states (consider only the behaviour – actions)
- abstract from nondeterminism
- abstract from internal behaviour

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

$$P \equiv Q \text{ implies that } C(P) \equiv C(Q)$$

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

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

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*

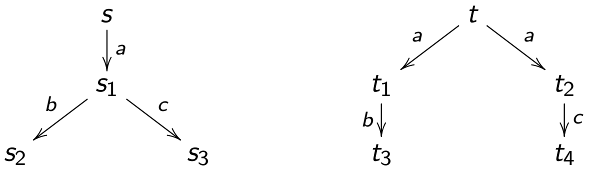
Theorem

$s \sim t$ *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* $s' \sim t'$.

Caption: BUT this IS NOT its definition !!!

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 to 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.

Strong Bisimilarity is a Congruence for CCS Operations

Theorem

Let P and Q be CCS processes such that $P \sim Q$. Then

- $\alpha.P \sim \alpha.Q$ *for each action* $\alpha \in Act$
- $P + R \sim Q + R$ *and* $R + P \sim R + Q$ *for each CCS process* R
- $P \mid R \sim Q \mid R$ *and* $R \mid P \sim R \mid Q$ *for each CCS process* R
- $P[f] \sim Q[f]$ *for each relabelling function* f
- $P \setminus L \sim Q \setminus L$ *for each set of labels* L.

Other Properties of Strong Bisimilarity

Following Properties Hold for any CCS Processes P, Q and R

- $P + Q \sim Q + P$
- $P \mid Q \sim Q \mid P$
- $P + Nil \sim P$
- $P \mid Nil \sim P$
- $(P + Q) + R \sim P + (Q + R)$
- $(P \mid Q) \mid R \sim P \mid (Q \mid R)$

A Simple Buffer

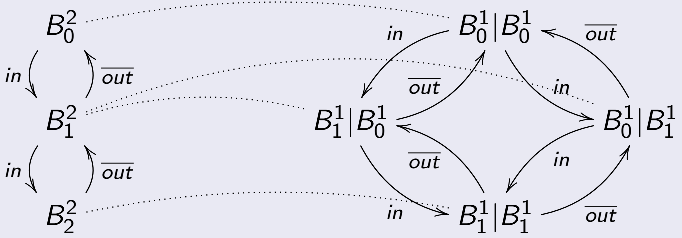
Buffer of Capacity 1

$$B_0^1 \stackrel{\mathrm{def}}{=} in.B_1^1$$
$$B_1^1 \stackrel{\mathrm{def}}{=} \overline{out}.B_0^1$$

Buffer of Capacity n

$$B_0^n \stackrel{\mathrm{def}}{=} in.B_1^n$$
$$B_i^n \stackrel{\mathrm{def}}{=} in.B_{i+1}^n + \overline{out}.B_{i-1}^n \quad \text{for } 0 < i < n$$
$$B_n^n \stackrel{\mathrm{def}}{=} \overline{out}.B_{n-1}^n$$

Example: $B_0^2 \sim B_0^1 | B_0^1$



Example – Buffer

Theorem

For all natural numbers n: $B_0^n \sim \underbrace{B_0^1 | B_0^1 | \cdots | B_0^1}_{n \text{ times}}$

Proof.

Construct the following binary relation, where $i_1, i_2, \ldots, i_n \in \{0, 1\}$:

$$R = \{ \big(B_i^n,\ B_{i_1}^1 | B_{i_2}^1 | \cdots | B_{i_n}^1 \big) \mid \sum_{j=1}^{n} i_j = i \}$$

- $\big(B_0^n,\ B_0^1 | B_0^1 | \cdots | B_0^1 \big) \in R$
- R is a strong bisimulation

$\square$

But still Internal Actions must be Abstracted away

Question

Does $a.\tau.Nil \sim a.Nil$ hold? NO!

Problem

Strong bisimilarity does not abstract away from τ actions.

Example: SmUni $\not\sim$ Spec

$$
\begin{tikzcd}
\mathrm{SmUni} \arrow[d, "\overline{pub}"] & \not\sim & \mathrm{Spec} \arrow[loop below, "\overline{pub}"] \\
(CM \mid CS_1) \setminus \{coin, coffee\} \arrow[d, "\tau"] & & \\
(CM_1 \mid CS_2) \setminus \{coin, coffee\} \arrow[d, "\tau"] & & \\
(CM \mid CS) \setminus \{coin, coffee\} \arrow[uu, bend right, "\overline{pub}"'] & &
\end{tikzcd}
$$

Weak Transitions will (mostly) hide τ actions

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

Definition of Weak Transition Relation

$$\stackrel{a}{\Longrightarrow} \; = \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^* & \text{if } a \neq \tau \\ (\xrightarrow{\tau})^* & \text{if } a = \tau \end{cases}$$

What does $s \stackrel{a}{\Longrightarrow} t$ informally mean?

- If $a \neq \tau$ then $s \stackrel{a}{\Longrightarrow} t$ means that
  from s we can get to t by doing zero or more τ actions, followed by the action a, followed by zero or more τ actions.
- If $a = \tau$ then $s \stackrel{\tau}{\Longrightarrow} t$ means that
  from s we can get to t by doing zero or more τ actions.

Weak Bisimilarity

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

Weak Bisimulations

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

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

Weak Bisimilarity

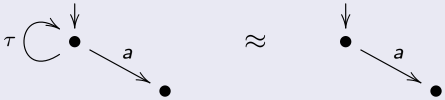
Two processes $p_1, p_2 \in Proc$ are weakly bisimilar ($p_1 \approx p_2$) if and only if there exists a weak bisimulation R such that $(p_1, p_2) \in R$.

$$\approx \; = \; \cup\{R \mid R \text{ is a weak bisimulation}\}$$

Properties of Weak Bisimilarity

Properties of $\approx$

- $\approx$ is an equivalence relation
- $\approx$ is the largest weak bisimulation
- validates lots of natural laws, e.g.
  - $a.\tau.P \approx a.P$
  - $P + \tau.P \approx \tau.P$
  - $a.(P + \tau.Q) \approx a.(P + \tau.Q) + a.Q$
  - $P + Q \approx Q + P \qquad P|Q \approx Q|P \qquad P + Nil \approx P \qquad \ldots$
- strong bisimilarity is included in weak bisimilarity ($\sim \subseteq \approx$)
- $\approx$ (totally) abstracts τ loops



Is Weak Bisimilarity a Congruence for CCS?

Theorem

Let P and Q be CCS processes such that $P \approx Q$. Then

- $\alpha.P \approx \alpha.Q$ *for each action* $\alpha \in Act$
- $P \mid R \approx Q \mid R$ *and* $R \mid P \approx R \mid Q$ *for each CCS process* R
- $P[f] \approx Q[f]$ *for each relabelling function* f
- $P \setminus L \approx Q \setminus L$ *for each set of labels* L.

But what about choice?

$\tau.a.Nil \approx a.Nil$ but $\tau.a.Nil + b.Nil \not\approx a.Nil + b.Nil$

Conclusion

Weak bisimilarity is not a congruence for CCS.

Verifying Correctness of Reactive Systems

Let *Impl* be an implementation of a system (e.g. in CCS syntax).

Equivalence Checking Approach

$$\mathit{Impl} \equiv \mathit{Spec}$$

- $\equiv$ is an abstract equivalence, e.g. $\sim$ or $\approx$
- *Spec* is often expressed in the same language as *Impl*
- *Spec* provides the full specification of the intended behaviour

Model Checking Approach

$$\mathit{Impl} \models \mathit{Property}$$

- $\models$ is the satisfaction relation
- *Property* is a particular feature, often expressed via a logic
- *Property* is a partial specification of the intended behaviour

Model Checking of Reactive Systems

Our Aim

- Develop a logic in which we can express interesting properties of reactive systems.
- Better if it is somehow connected with bisimulation semantics.

Logical Properties of Reactive Systems

Modal Properties – what can happen now (possibility, necessity)

- drink a coffee (can drink a coffee now)
- does not drink tea
- drinks both tea and coffee
- drinks tea after coffee

Temporal Properties – behaviour in time

- never drinks any alcohol
  (safety property: nothing bad can happen)
- eventually will have a glass of wine
  (liveness property: something good will happen)

Can these properties be expressed using equivalence checking?

Hennessy-Milner Logic – Syntax

Syntax of the Formulae ($a \in Act$)

$$F, G \;::=\; tt \mid ff \mid F \wedge G \mid F \vee G \mid \langle a \rangle F \mid [a]F$$

Intuition:

- tt all processes satisfy this property
- ff no process satisfies this property
- $\wedge$, $\vee$ usual logical AND and OR
- $\langle a \rangle F$ there is at least one a-successor that satisfies F
- $[a]F$ all a-successors have to satisfy F

Remark

Temporal properties like *always/never in the future* or *eventually* are not included.

Hennessy-Milner Logic – Semantics

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

Validity of the logical triple $p \models F$ ($p \in Proc$, F a HM formula)

$$
\begin{array}{rl}
p \models tt & \text{for each } p \in Proc \\
p \models ff & \text{for no } p \text{ (we also write } p \not\models ff\text{)} \\
p \models F \wedge G & \text{iff } p \models F \text{ and } p \models G \\
p \models F \vee G & \text{iff } p \models F \text{ or } p \models G \\
p \models \langle a \rangle F & \text{iff } p \xrightarrow{a} p' \text{ for some } p' \in Proc \text{ such that } p' \models F \\
p \models [a]F & \text{iff } p' \models F, \text{for all } p' \in Proc \text{ such that } p \xrightarrow{a} p'
\end{array}
$$

We write $p \not\models F$ whenever p does not satisfy F.

Hennessy-Milner Logic – Denotational Semantics

For a formula F let $[\![F]\!] \subseteq \mathit{Proc}$ contain all states that satisfy F.

Denotational Semantics: $[\![\_]\!] : \mathit{Formulae} \rightarrow 2^{\mathit{Proc}}$

- $[\![tt]\!] = \mathit{Proc}$
- $[\![f\!f]\!] = \emptyset$
- $[\![F \vee G]\!] = [\![F]\!] \cup [\![G]\!]$
- $[\![F \wedge G]\!] = [\![F]\!] \cap [\![G]\!]$
- $[\![\langle a \rangle F]\!] = \langle \cdot a \cdot \rangle [\![F]\!]$
- $[\![[a]F]\!] = [\cdot a \cdot][\![F]\!]$

where $\langle \cdot a \cdot \rangle, [\cdot a \cdot] : 2^{(\mathit{Proc})} \rightarrow 2^{(\mathit{Proc})}$ are defined by

$$\langle \cdot a \cdot \rangle S = \{p \in \mathit{Proc} \mid \exists p'.\ p \xrightarrow{a} p' \text{ and } p' \in S\}$$

$$[\cdot a \cdot] S = \{p \in \mathit{Proc} \mid \forall p'.\ p \xrightarrow{a} p' \implies p' \in S\}.$$

The Correspondence Theorem

Theorem

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ *be an LTS,* $p \in Proc$ *and* F *a formula of Hennessy-Milner logic. Then*

$$p \models F \quad \textit{if and only if} \quad p \in [\![F]\!].$$

Proof: by structural induction on the structure of the formula F.

Image-Finite Labelled Transition Systems

Image-Finite System

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS. We call it image-finite iff for every $p \in Proc$ and every $a \in Act$ the set

$$\{p' \in Proc \mid p \xrightarrow{a} p'\}$$

is finite.

Relationship between HM Logic and Strong Bisimilarity

Theorem (Hennessy-Milner)

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an image-finite LTS and $p, q \in Proc$. Then

$$p \sim q$$

if and only if

for every HM formula F: $(p \models F \iff q \models F)$.

Strong Bisimulation as a Greatest Fixed Point

Function $\mathcal{F} : 2^{(Proc \times Proc)} \to 2^{(Proc \times Proc)}$

Let $S \subseteq Proc \times Proc$. Then we define $\mathcal{F}(S)$ as follows:

$(s, t) \in \mathcal{F}(S)$ if and only if for each $a \in Act$:

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

Observations

- $(2^{(Proc \times Proc)}, \subseteq)$ is a complete lattice and $\mathcal{F}$ is monotonic
- S is a strong bisimulation if and only if $S \subseteq \mathcal{F}(S)$

Strong Bisimilarity is the Greatest Fixed Point of $\mathcal{F}$

$$\sim = \bigcup \{ S \in 2^{(Proc \times Proc)} \mid S \subseteq \mathcal{F}(S) \}$$

HML with One Recursively Defined Variable

Syntax of Formulae

Formulae are given by the following abstract syntax

$$F ::= X \mid tt \mid ff \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle a \rangle F \mid [a]F$$

where $a \in Act$ and X is a distinguished variable with a definition

- $X \stackrel{\min}{=} F_X$, or $X \stackrel{\max}{=} F_X$

such that F_X is a formula of the logic (can contain X).

How to Define Semantics?

For every formula F we define a function $O_F : 2^{Proc} \rightarrow 2^{Proc}$ s.t.

- if S is the set of processes that satisfy X then
- $O_F(S)$ is the set of processes that satisfy F.

Definition of $O_F : 2^{Proc} \to 2^{Proc}$ (let $S \subseteq Proc$)

$$
\begin{aligned}
O_X(S) &= S \\
O_{tt}(S) &= Proc \\
O_{ff}(S) &= \emptyset \\
O_{F_1 \wedge F_2}(S) &= O_{F_1}(S) \cap O_{F_2}(S) \\
O_{F_1 \vee F_2}(S) &= O_{F_1}(S) \cup O_{F_2}(S) \\
O_{\langle a \rangle F}(S) &= \langle \cdot a \cdot \rangle O_F(S) \\
O_{[a]F}(S) &= [\cdot a \cdot] O_F(S)
\end{aligned}
$$

O_F is monotonic for every formula F

$$S_1 \subseteq S_2 \;\Rightarrow\; O_F(S_1) \subseteq O_F(S_2)$$

Proof: easy (structural induction on the structure of F).

Semantics

Observation

We know that $(2^{Proc}, \subseteq)$ is a complete lattice and O_F is monotonic, so O_F has a unique greatest and least fixed point.

Semantics of the Variable X

- If $X \stackrel{\max}{=} F_X$ then

$$[\![X]\!] = \bigcup\{S \subseteq Proc \mid S \subseteq O_{F_X}(S)\}.$$

- If $X \stackrel{\min}{=} F_X$ then

$$[\![X]\!] = \bigcap\{S \subseteq Proc \mid O_{F_X}(S) \subseteq S\}.$$

Selection of Temporal Properties

- $Inv(F)$: $X \stackrel{\max}{=} F \wedge [Act]X$
- $Pos(F)$: $X \stackrel{\min}{=} F \vee \langle Act \rangle X$

- $Safe(F)$: $X \stackrel{\max}{=} F \wedge ([Act]ff \vee \langle Act \rangle X)$
- $Even(F)$: $X \stackrel{\min}{=} F \vee (\langle Act \rangle tt \wedge [Act]X)$

- $F\ \mathcal{U}^w\ G$: $X \stackrel{\max}{=} G \vee (F \wedge [Act]X)$
- $F\ \mathcal{U}^s\ G$: $X \stackrel{\min}{=} G \vee (F \wedge \langle Act \rangle tt \wedge [Act]X)$

Using until we can express e.g. $Inv(F)$ and $Even(F)$:

$$Inv(F) \equiv F\ \mathcal{U}^w\ ff \qquad\qquad Even(F) \equiv tt\ \mathcal{U}^s\ F$$