Process algebras, bisimulation (and logics)

Nadia Busi

University of Bologna

Sequential vs concurrent computation

- Semantics of sequential programs: function from the input state to the output state
- Semantics of concurrent programs:
  - No general agreement
  - A concurrent program is **not** a function

Concurrent programs and functions

- Consider, e.g., the following programs:
  - X := 2
  - X := 1; X:= X+1
- They compute the same function, but…
  - X := 2 | X:= 2
  - (X := 1; X:= X+1) | X:= 2
- They do not compute the same function
- Viewing concurrent programs as functions gives us a notion of equivalence that is not a congruence (and produces a non-compositional semantics).

Concurrent programs and functions

- A concurrent program may not terminate (e.g. operating systems, controllers of a railway system…)
  - In sequential languages, nonterminating programs useless (wrong)
- The behaviour of concurrent programs can be nondeterministic
  - (X := 1; X:= X+1) | X := 2

Concurrent systems as reactive systems

- Concurrent system interact with the environment during the computation
- **Reactive system**: a system that computes by reacting to stimuli from the environment
  - Inherently parallel systems
  - Key role in their behaviour played by communication with the environment
- A sequential program can be viewed as a reactive system that interacts only at the beginning and at the end of the computation

Concurrent systems as reactive systems

- The behaviour of concurrent programs should tell us **when** and **how** they can interact with the environment
- The behaviour of concurrent programs is very hard to analyse and understand
  - Formal definition of behaviour

A theory of processes

- Process algebra
- Labelled transition systems
- Bisimulation
- Structural operational semantics
- (Hennessy-Milner) logics

Process Algebras

- Process = system with a specified behaviour
- Specification languages for reactive systems
  - Algebra: collection of (basic processes and) operations for building new processes from existing ones
- Key issue: communication/interaction among processes

Communication

- Information exchange between the producer of information (sender) and the consumer (receiver)
- Communication medium
  - Buffers, shared variables, tuple spaces, …
- Idea: no need to distinguish between active components (senders/receivers) and passive ones (communication media)
  - Everything is a process
  - Interaction via message passing, modeled as synchronized communication

Process algebras

- CCS [Milner '80s],
- CSP [Hoare '80s],
- ACP [BergstraKlop '80s],
- Pi-calculus [Milner, Parrow, Walker '90s]
- Mobile ambients [Cardelli, Gordon 00's]

A Calculus of Communicating Systems (CCS)

- 0 nil
  - The process that does nothing
- a.P action prefix
  - Perform action a and then behave like P
  - Clock = tick.Clock
  - Types of actions:
    - $\overline{a}$: send a signal on channel a
    - a: receive a signal on channel a
    - τ: silent action
  - CM = coin.$\overline{\text{coffee}}$.CM

A Calculus of Communicating Systems (CCS)

- P+Q choice operator
  - The process P+Q has the capabilities of both P and Q
  - Choosing to perform an action from P will preempt the further execution of actions from Q (and vice versa)
  - CTM = coin.($\overline{\text{coffee}}$.CTM + $\overline{\text{tea}}$.CTM)
  - Exercise: define a coffee machine that may steal the money and fail

A Calculus of Communicating Systems (CCS)

- P |Q parallel composition operator
  - The process P|Q describes a system where
    - P and Q may proceed independently and
    - They may communicate via complementary ports
  - “A mathematician is a device for turning coffee into theorems” (P. Erdos)
    - $M = \overline{\text{coin}}.\text{coffee}.\overline{\text{theorem}}.M$
    - $CM = \text{coin}.\overline{\text{coffee}}.CM$
    - M | CM
    - The channel theorem is used by the mathematician to communicate with its research environment
    - Processes M and CM **may** communicate on channels coffee and coin, but they can also communicate with other processes (e.g., another guy can use the coffee machine CM)

A Calculus of Communicating Systems (CCS)

- P\a restriction operator
  - In P\a the scope of channel a is restricted to P
  - Channel a can only be used for communication within P
  - Private coffee machine
    - $M = \overline{\text{coin}}.\text{coffee}.\overline{\text{theorem}}.M$
    - $CM = \text{coin}.\overline{\text{coffee}}.CM$
    - (M | CM)\coin\coffee
    - The channels coin and coffee may be only used for communication between the mathematician and the coffee machine
    - The channel theorem is visible to the environment

A Calculus of Communicating Systems (CCS)

- P[f] relabelling operator
  - In P[f] the name of each channel a in the domain of f is replaced by f(a)
  - Vending machines
    - CM = coin.$\overline{\text{coffee}}$.CM
    - ChocM = coin.$\overline{\text{chocolate}}$.ChocM
    - VM = coin.item.VM
    - CM = VM[coffee/item]
    - ChocM = VM[chocolate/item]

Behaviour of processes

- A process passes through states during an execution
- Processes change their state by performing actions
  - Example: mathematician
- No difference between processes and states:
  - By performing an action, a process evolves to another process, describing what remains to be executed of the original one
- Processes evolve by performing transitions
  - Example!

Behaviour of processes: transitions

$$M \xrightarrow{\overline{coin}} M_1$$

$$M_1 = coffee.\overline{theorem}.M_1$$

$$CM \mid M \xrightarrow{?} CM_1 \mid M_1$$

Binary synchronization: communication produces an unobservable transition (I.e., a transition that cannot further synchronize)

Behaviour of processes: transitions

Silent (unobservable) action τ

$$CM \mid M \xrightarrow{\tau} CM_1 \mid M_1$$

Exercise: labelled transition system describing the behaviour of CM | M

Behaviour of processes

- As silent actions are unobservable, the following process could be an appropriate high-level specification of the behaviour of CM|M:
  - Spec = $\overline{\text{theorem}}$.Spec
- Notion of "behavioural equivalence" between processes

Labelled transition systems

- Processes represented by vertices of edge-labelled oriented graphs
- A change of process state caused by performing an action corresponds to moving along an edge (labelled with the action name) that goes out of that state

Labelled transition systems

Definition 4.1 [Labelled Transition Systems] A *labelled transition system (LTS)* is a triple $(\mathsf{Proc}, \mathsf{Act}, \{\xrightarrow{a} \mid a \in \mathsf{Act}\})$, where:

- Proc is a set of *states*, ranged over by s;
- Act is a set of *actions*, ranged over by a;
- $\xrightarrow{a} \subseteq \mathsf{Proc} \times \mathsf{Proc}$ is a *transition relation*, for every $a \in \mathsf{Act}$. As usual, we shall use the more suggestive notation $s \xrightarrow{a} s'$ in lieu of $(s, s') \in \xrightarrow{a}$, and write $s \stackrel{a}{\nrightarrow}$ (read "s refuses a") iff $s \xrightarrow{a} s'$ for no state s'.

Sometimes a state is singled out as the initial state of the LTS

Example: vending machine

- A vending machine, capable of dispensing tea or coffee for 1 coin
- VM = coin.(chooseTea.$\overline{\text{tea}}$.VM + chooseCoffee.$\overline{\text{coffee}}$.VM)
- Exercise: LTS

Structural Operational Semantics

- The step from a CCS process to the LTS describing its behaviour is taking using the framework of Structural Operational Semantics [Plotkin81]
- The collection of CCS processes is the set of states of a LTS
  - The transitions of such LTS are those that can be proven to hold by means of a collection of syntax-driven rules

Formal syntax of CCS

- $\mathcal{A}$ = countably infinite collection of channel names

$$\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$$

$$\mathcal{L} = \mathcal{A} \cup \bar{\mathcal{A}}$$

$$\mathsf{Act} = \mathcal{L} \cup \{\tau\}$$

Formal syntax of CCS

Definition 4.3 The collection $\mathcal{P}$ of *CCS expressions* is given by the following grammar:

$$P, Q ::= K \;|\; \alpha.P \;|\; \sum_{i \in I} P_i \;|\; P \,|\, Q \;|\; P[f] \;|\; P \setminus L \;,$$

where

- K is a process name in $\mathcal{K}$;
- α is an action in Act;
- I is an index set;
- $f : \mathsf{Act} \rightarrow \mathsf{Act}$ is a *relabelling function* satisfying the following constraints:

$$\begin{aligned} f(\tau) &= \tau \text{ and} \\ f(\bar{a}) &= \overline{f(a)} \text{ for each label } a \;; \end{aligned}$$

- L is a set of labels.

Formal syntax of CCS

- The behaviour of each process constant is given by a defining equation

  $$K \stackrel{\mathrm{def}}{=} P \ .$$

- Example:

$$\begin{aligned} \mathrm{Counter}_0 &\stackrel{\mathrm{def}}{=} \mathrm{up}.\mathrm{Counter}_1 \\ \mathrm{Counter}_n &\stackrel{\mathrm{def}}{=} \mathrm{up}.\mathrm{Counter}_{n+1} + \mathrm{down}.\mathrm{Counter}_{n-1} \quad (n > 0) \end{aligned}$$

Formal semantics of CCS

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

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'} \qquad \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

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

Behavioural equivalence

- CCS can be used to describe both the implementation of processes and the specification of their expected behaviour
- Behavioural equivalence: two processes, say SPEC and IMPL, are equivalent if they describe essentially the same behaviour (maybe at different levels of abstraction)

Equivalence

Definition 5.1 Let X be a set. A *binary relation* over X is a subset of $X \times X$, the set of pairs of elements of X. If R is a binary relation over X, we often write $x\,R\,y$ instead of $(x, y) \in R$.

An equivalence relation over X is a relation that satisfies the following constraints:

- R is *reflexive*—that is, $x\,R\,x$ for each $x \in X$;
- R is *symmetric*—that is, $x\,R\,y$ implies $y\,R\,x$, for all $x, y \in X$; and
- R is *transitive*—that is, $x\,R\,y$ and $y\,R\,z$ imply $x\,R\,z$, for all $x, y, z \in X$.

Desirable properties of a behavioural relation

- Each process is a correct implementation of itself (reflexivity)
- Support stepwise derivation of implementations from specifications (transitivity)
- Two behaviourally equivalent processes can be used interchangeably as part of large process descriptions without affecting the overall behaviour (congruence)
  - P R Q implies C[P] R C[Q]

Desirable properties of a behavioural relation

- Behavioural equivalence based on the observable behaviour of processes (not on their structure)
  - Identify two processes unless there is some sequences of interactions that an observer may have with them, leading to different outcomes
- Lack of consensus on the appropriate notion of behavioural equivalence
  - Large number of proposals
  - Lattice of behavioural equivalences [vanGlabbeek]

First attempt: trace equivalence

- A trace of a process P is a sequence

  $\alpha_1 \cdots \alpha_k \in \mathsf{Act}^* \ (k \geq 0)$

  such that there exists a sequence of transitions

  $$P = P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_k} P_k$$

P and Q are behaviourally equivalent if Traces(P) = Traces(Q)

Trace equivalence

- Is trace equivalence reasonable for reactive machines that interact with their environment?
- Example: vending machine
  - VM = coin.(chooseTea.tea.VM + chooseCoffee.coffee.VM)
  - VM’ = coin.chooseTea.tea.VM’ + coin.chooseCoffee.coffee.VM’
- VM and VM’ have the same traces

Trace equivalence

- If you want coffee and you hate tea, which machine would you like to interact with?
- U = coin.chooseCoffee.coffee.U
- *A* = {coin, chooseCoffee, coffee, chooseTea, tea}
- (U |VM)*A* performs an infinite computation consisting of silent moves
- (U | VM')*A* may deadlock (if the machine reaches the state where it is only willing to deliver tea)

Trace equivalence

- Trace equivalent processes may exhibit different deadlock behaviour when interacting with other parallel processes
- We reject the law

$$\alpha.(P + Q) = \alpha.P + \alpha.Q$$

Completed traces

Exercise 5.2 *A* completed trace *of a process P is a sequence $\alpha_1 \cdots \alpha_k \in \mathsf{Act}^*$ $(k \geq 0)$ such that there exists a sequence of transitions*

$$P = P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_k} P_k \not\rightarrow \ ,$$

for some $P_1, \ldots, P_k$. The completed traces of a process may be seen as capturing its deadlock behaviour, as they are precisely the sequences of actions that may lead the process into a state from which no further action is possible.

Exercise: completed traces

- Do the processes (U |VM)\A and (U |VM')\A have the same completed traces?
- Is it true that if P and Q are two processes with the same completed traces and L is a set of labels, then P\L and Q\L also have the same completed traces?

LTS isomorphism

- Consider, e.g., the processes X and Y, where
  - X = a.b.X
  - Y = a.Z
  - Z = b.a.Z
- The (portions of) LTS (reachable from X and Y) are not isomorphic

Strong bisimulation

- Trace equivalence is not suitable
  - VM and VM’ exhibit different deadlock behaviour when composed with user U
  - Traces focus only on sequences of actions that a process may perform but do not consider the communication capabilities of the intermediate states
  - Communication potential at intermediate states **does matter**
  - After input of a coin,
    - VM enters a state in which it is willing to output either coffee or tea
    - VM’ can only enter a state in which it is willing to deliver either coffee or tea, but not both

Properties of a behavioural relation

- Allow to distinguish processes with different deadlock behaviour when interacting with other processes
- Take into account communication capabilities of intermediate states
- Two processes are equivalent if they have the same traces <u>and the states that they reach are still equivalent</u>
- Bisimulation [Park'80]

Strong bisimulation

Definition 5.2 [Strong Bisimulation] A binary relation $\mathcal{R}$ over the set of states of an LTS is a *bisimulation* iff whenever $s_1 \ \mathcal{R} \ s_2$ and α is an action:

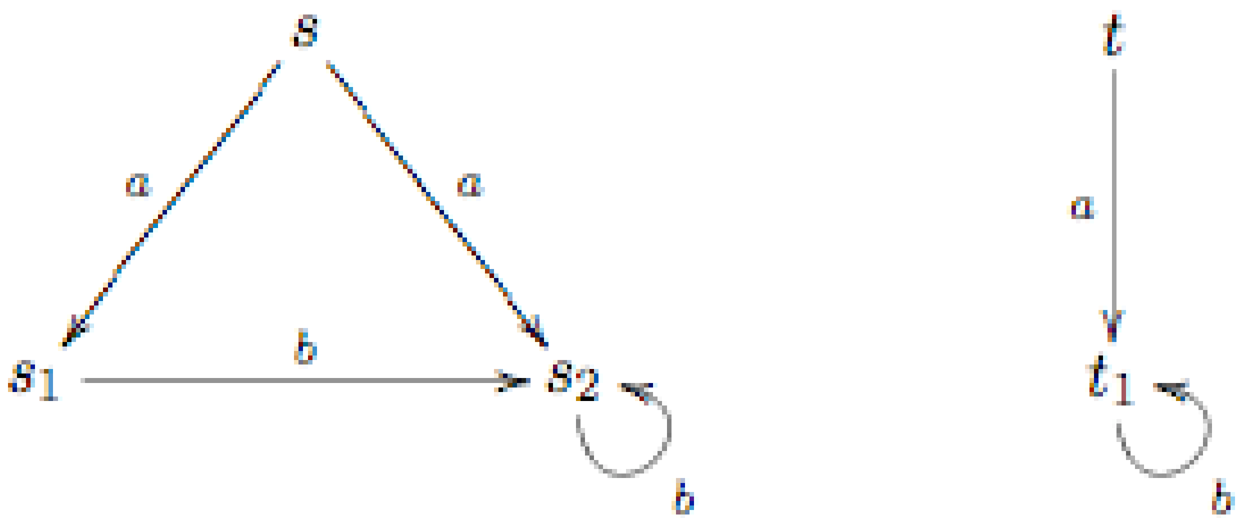
- if $s_1 \xrightarrow{\alpha} s'_1$, then there is a transition $s_2 \xrightarrow{\alpha} s'_2$ such that $s'_1 \ \mathcal{R} \ s'_2$;
- if $s_2 \xrightarrow{\alpha} s'_2$, then there is a transition $s_1 \xrightarrow{\alpha} s'_1$ such that $s'_1 \ \mathcal{R} \ s'_2$.

Two states s and s' are *bisimilar*, written $s \sim s'$, iff there is a bisimulation that relates them. Henceforth the relation $\sim$ will be referred to as *strong bisimulation equivalence* or *strong bisimilarity*.

Strong bisimulation for CCS processes

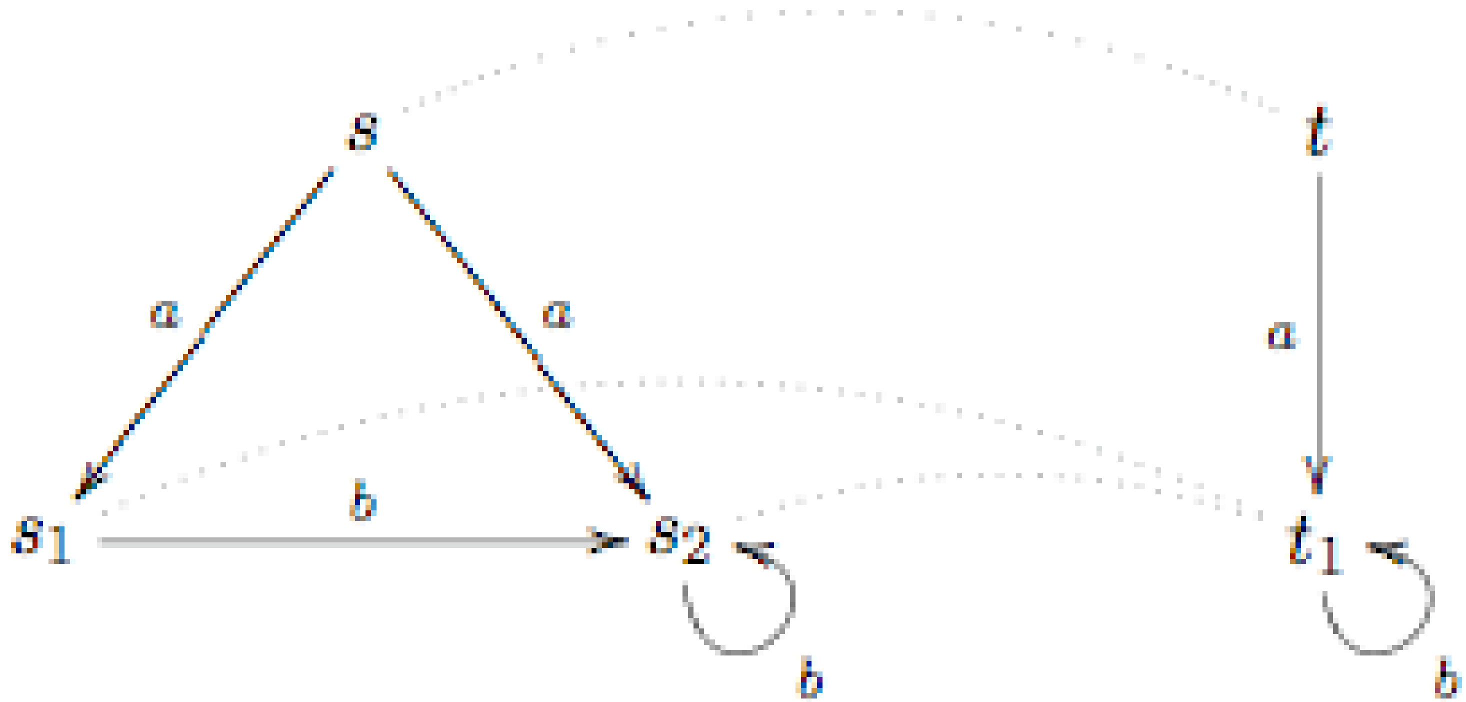
- As the semantics of CCS is given in terms of an LTS whose staes are CCS processes, the definition of strong bisimulation also applies to CCS processes
- Bisimulation proof technique
  - Two processes are bisimilar if there exists a strong bisimulation relating them
  - To prove that two processes are related by ~ it suffices to exhibit a strong bisimulation that relates them

Example: s~t



$$\mathcal{R} = \{(s,t),(s_1,t_1),(s_2,t_1)\}$$

Example: s~t



Example: s~t

- (s,t) is in R
- We need to show that R is a bisimulation:
  - For each pair of states in R, check if all possible transitions from both states can be matched by corresponding transitions from the other states

Example: $s_i \sim t$

$$s_1 \xrightarrow{a} s_2 \xrightarrow{a} s_3 \xrightarrow{a} s_4 \xrightarrow{a} \cdots$$

$$\begin{tikzcd} t \arrow[loop below, "a"] \end{tikzcd}$$

$$\mathcal{R} = \{(s_i, t) \mid i \in \mathbb{N}\}$$

Example: not VM~VM'

- Suppose VM~VM'

$$VM' \xrightarrow{coin} chooseTea.\overline{tea}.VM'$$

According to the def of bisimulation there must be a transition

$$VM \xrightarrow{coin} P$$

For some P s.t. P R chooseTea.tea.VM'

Example: not VM~VM'

- The only transition of VM labelled with coin leads to the state
  - chooseTea.tea.VM + chooseCoffee.coffee.VM
- The above state can perform a transition labelled with chooseCoffee, that cannot be matched by any transition of state
  - chooseTea.tea.VM
- Hence, any relation containing (VM,VM') cannot be a bisimulation

Exercise

Exercise 5.4 *Consider the processes*

$$P \stackrel{\text{def}}{=} a.(b.0 + c.0) \quad \textit{and}$$
$$Q \stackrel{\text{def}}{=} a.b.0 + a.c.0 \ .$$

Show that P and Q are not strongly bisimilar.

Properties of ~

- For all LTS, the relation ~ is an equivalence
  - Reflexive: for all states s, s~s
    - R = {(s,s) | s is a state of the LTS}
  - Symmetric: for all states s,t, if s~t then t~s
    - If s~t then there exists a bisimulation R s.t. (s,t) is in R
    - Take R^{-1}
  - Transitive: for all states s,t,r: if s~t and t~r then s~r
    - If s~t and t~r then there exist two bisimulations R and R' s.t. (s,t) in R and (t,r) in R;
    - Take S = {(u,v) | there exists z s.t. (u,z) in R and (z,v) in R'}

Properties of ~

- For all LTSs, ~ is the largest strong bisimulation
  - Observe that the def of ~ states that
    - ~ = U {R | R is a bisimulation}
  - Hence, each bisimulation is included in ~
  - We need to show that U {R | R is a bisimulation} is a bisimulation

Properties of ~

- For all LTSs, ~ satisfies the following:

$s_1 \sim s_2$ iff for each action α,

- if $s_1 \xrightarrow{\alpha} s_1'$, then there is a transition $s_2 \xrightarrow{\alpha} s_2'$ such that $s_1' \sim s_2'$;
- if $s_2 \xrightarrow{\alpha} s_2'$, then there is a transition $s_1 \xrightarrow{\alpha} s_1'$ such that $s_1' \sim s_2'$.

Properties of ~

- Two strong bisimilar processes have the same sets of traces:

$$P \sim Q \textit{ and } \alpha_1 \cdots \alpha_k \in \mathit{Traces}(P) \textit{ imply } \alpha_1 \cdots \alpha_k \in \mathit{Traces}(Q)$$

Properties of ~

Exercise 5.8 *Show that the following relations are strong bisimulations:*

$$\{(P \mid Q, Q \mid P) \mid \text{where } P, Q \text{ are CCS processes}\}$$
$$\{(P \mid \mathbf{0}, P) \mid \text{where } P \text{ is a CCS process}\}$$
$$\{((P \mid Q) \mid R, P \mid (Q \mid R)) \mid \text{where } P, Q, R \text{ are CCS processes}\} \ .$$

Conclude that, for all P, Q, R,

$$P \mid Q \sim Q \mid P \tag{13}$$
$$P \mid \mathbf{0} \sim P \quad \text{and} \tag{14}$$
$$(P \mid Q) \mid R \sim P \mid (Q \mid R) \ . \tag{15}$$

Properties of ~

- ~ is a congruence
- We could replace equivalent processes for equivalent processes in any larger process expression without affecting its behaviour

$\sim$ is a congruence

Proposition 5.1 Let P, Q, R be CCS processes. Assume that $P \sim Q$. Then

- $\alpha.P \sim \alpha.Q$, for each action α;
- $P + R \sim Q + R$ and $R + P \sim R + Q$, for each process R;
- $P \mid R \sim Q \mid R$ and $R \mid P \sim R \mid Q$, for each process R;
- $P[f] \sim Q[f]$, for each relabelling f; and
- $P \setminus L \sim Q \setminus L$, for each set of labels L.

Exercise: hiding

Exercise 5.10 *For each set of labels L and process P, we may wish to build the process $\tau_L(P)$ that is obtained by turning into a τ each action α performed by P with $\alpha \in L$ or $\bar{\alpha} \in L$. Operationally, the behaviour of the construct $\tau_L(\ )$ can be described by the following two rules:*

$$\frac{P \xrightarrow{\alpha} P'}{\tau_L(P) \xrightarrow{\tau} \tau_L(P')} \quad \text{if } \alpha \in L \text{ or } \bar{\alpha} \in L$$

$$\frac{P \xrightarrow{\mu} P'}{\tau_L(P) \xrightarrow{\mu} \tau_L(P')} \quad \text{if } \mu = \tau \text{ or } \mu, \bar{\mu} \notin L$$

Prove that $\tau_L(P) \sim \tau_L(Q)$, whenever $P \sim Q$.

Consider the question of whether the operation $\tau_L(\ )$ can be defined in CCS modulo $\sim$—that is, can you find a CCS expression $C_L[\ ]$ with a "hole" (a place holder when another process can be plugged) such that, for each process P,

$$\tau_L(P) \sim C_L[P] \ ?$$

Exercise: simulation

Exercise 5.12 (Simulation) *Let us say that a binary relation $\mathcal{R}$ over the set of states of an LTS is a simulation iff whenever $s_1 \mathcal{R} s_2$ and α is an action:*

- *if $s_1 \xrightarrow{\alpha} s_1'$, then there is a transition $s_2 \xrightarrow{\alpha} s_2'$ such that $s_1' \mathcal{R} s_2'$.*

We say that s' simulates s, written $s \sqsubseteq s'$, iff there is a simulation $\mathcal{R}$ with $s \mathcal{R} s'$. Two states s and s' are simulation equivalent, written $s \simeq s'$, iff $s \sqsubseteq s'$ and $s' \sqsubseteq s$ both hold.

1. *Prove that $\sqsubseteq$ is a preorder.*

2. *Build simulations showing that*

$$\begin{array}{rcl} a.\mathbf{0} & \sqsubseteq & a.a.\mathbf{0} \quad \textit{and} \\ a.b.\mathbf{0} + a.c.\mathbf{0} & \sqsubseteq & a.(b.\mathbf{0} + c.\mathbf{0}) \ . \end{array}$$

Do the converse relations hold?

Exercise: simulation

- Show that strong bisimilarity is included in simulation equivalence
- Does the converse inclusion also hold?
  - consider a.b and a + a.b

Stratification

$\sim_0 \triangleq \mathcal{P} \times \mathcal{P}$

$P \sim_{n+1} Q \triangleq$:

1. if $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \sim_n Q'$.
2. if $Q \xrightarrow{\mu} Q'$, then there is P' such that $P \xrightarrow{\mu} P'$ and $P' \sim_n Q'$.

Then set:

$\sim_\omega \triangleq \bigcap_n \sim_n$

Stratification - examples

- $a \sim_0 b$
- not $a \sim_1 b$
- $c.a + d \sim_1 c.b + d$
- not $c.a + d \sim_2 c.b + d$
- Is $\sim_w = \sim$???

Stratification

- Image-finite process: each reachable process can only perform a finite set of transitions
- $\sim = \sim_w$ for image-finite processes
- Let $a^n = a. \ldots .a.0$ and $X = a.X$
- $\Sigma_n a^n \sim_w \Sigma_n a^n + X$, but
- not $\Sigma_n a^n \sim \Sigma_n a^n + X$

Checking bisimulation

- Stratification is the basis for algorithms for checking bisimulation
- These algorithms work for finite-state processes (I.e., each process has only a finite number of derivatives)
  - They proceed by progressively refining a partition of all processes
- Complexity of bisimulation (m transitions, n states):
  - O(m log n) time, O(m + n) space [PaigeTarjan'87]

Bisimulation up-to $\sim$

We write $P \sim\mathcal{R}\sim Q$ if there are P', Q' s.t. $P \sim P'$, $P' \mathcal{R} Q'$, and $Q' \sim Q$ (and alike for similar notations).

Definition 7 (bisimulation up-to $\sim$) A relation $\mathcal{R}$ on the states of an LTS is a *bisimulation up-to* $\sim$ if $P \mathcal{R} Q$ implies:

1. if $P \xrightarrow{\mu} P'$, then there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \sim\mathcal{R}\sim Q'$.
2. if $Q \xrightarrow{\mu} Q'$, then there is P' such that $P \xrightarrow{\mu} P'$ and $P' \sim\mathcal{R}\sim Q'$.

Exercise 8 If $\mathcal{R}$ is a bisimulation up-to $\sim$ then $\mathcal{R} \subseteq \sim$. (Hint: prove that $\sim \mathcal{R} \sim$ is a bisimulation.)

Weak bisimilarity

- Strong bisimilarity satisfies many of the properties we expect by a notion of behavioural equivalence
  - It is a congruence, supports an elegant proof technique, permits to establish several natural equalities (e.g., P|Q ~ Q|P)
- Is there some item in our wish list that is not met by strong bisimilarity?

Tau actions

- τ denotes an internal, observable action
- Is is produced by synchronization of two processes
- A notion of behavioural equivalence should abstract from internal steps
- Consider $a.\tau.0$ and $a.0$
  - They should be behaviourally equivalent
  - They are not strong bisimilar
- Strong bisimulation treats internal actions in the same way as other actions

Tau actions

- We look for a notion of behavioural equivalence that
  - Has the good properties of strong bisimilarity
  - Abstracts from internal actions in the behaviour of processes
- Could we simply erase all the internal actions in the behaviour of a process?
- This works for $a.\tau.0$ and $a.0$, but…

Tau actions

- Consider the mathematician
  - M = coin.coffee.theorem.M
- And a new version of the coffee machine
  - CM'' = coin.coffee.CM'' + coin.CM''
  - Upon receipt of a coin, this coffee machine can decide to go back to its initial state without delivering coffee
- Take the system (M | CM'')\{coin, coffee}
  - The system either loops (correct computation) or reaches a deadlocked state
- Even if not directly observable, the transition leading to the deadlocked state cannot be ignored because it pre-empts other possible behaviours of the machine

Tau actions

- Unobservable actions cannot be just erased because - in light of their pre-emptive power - they may affect what we observe.
- This fact is unimportant in automata theory, where ε-transitions do not increase the expressive power
- We expect that the behaviour of the specification Spec = theorem.Spec is **not** equivalent to that of the process (M | CM")\{coin, coffee}

New transition relation

Definition 5.3 Let P and Q be CCS processes. We write $P \stackrel{\epsilon}{\Rightarrow} Q$ iff there is a (possibly empty) sequence of τ-labelled transitions that leads from P to Q. (If the sequence is empty, then $P = Q$.)

For each action α, we write $P \stackrel{\alpha}{\Rightarrow} Q$ iff there are processes P' and Q' such that

$$P \stackrel{\epsilon}{\Rightarrow} P' \stackrel{\alpha}{\rightarrow} Q' \stackrel{\epsilon}{\Rightarrow} Q \ .$$

For each action α, we use $\hat{\alpha}$ to stand for ϵ if $\alpha = \tau$, and for α otherwise.

Weak bisimulation

Definition 5.4 [Weak Bisimulation and Observational Equivalence] A binary relation $\mathcal{R}$ over the set of states of an LTS is a *weak bisimulation* iff whenever $s_1 \; \mathcal{R} \; s_2$ and α is an action:

- if $s_1 \xrightarrow{\alpha} s_1'$, then there is a transition $s_2 \overset{\hat{\alpha}}{\Rightarrow} s_2'$ such that $s_1' \; \mathcal{R} \; s_2'$;
- if $s_2 \xrightarrow{\alpha} s_2'$, then there is a transition $s_1 \overset{\hat{\alpha}}{\Rightarrow} s_1'$ such that $s_1' \; \mathcal{R} \; s_2'$.

Two states s and s' are *observationally equivalent* (or *weakly bisimilar*), written $s \approx s'$, iff there is a weak bisimulation that relates them. Henceforth the relation $\approx$ will be referred to as *observational equivalence* or *weak bisimilarity*.

Weak bisimulation - example

Example 5.4 Let us consider the following labelled transition system.

$$s \xrightarrow{\tau} s_1 \xrightarrow{a} s_2 \qquad\qquad t \xrightarrow{a} t_1$$

Obviously $s \not\sim t$. On the other hand $s \approx t$ because

$$\mathcal{R} = \{(s,t), (s_1,t), (s_2,t_1)\}$$

is a weak bisimulation such that $(s,t) \in \mathcal{R}$. It remains to verify that $\mathcal{R}$ is indeed a weak bisimulation.

Example - livelock

- Consider the processes
  - A = a.0 + t.B
  - B = b.0 + t.A
- We have *A* is weakly bisimilar to
  - C = a.0 + b.0
- Observe that *A* has a livelock (I.e. possibility of divergence) whereas *C* hasn't
- Weak bisimilarity assumes that is a process can escape from a loop consisting of internal transitions, then it will eventually do so.
  - Crucial property for verification of communication protocols

Example - divergence

- Process 0 is weakly bisimilar to process
  - Div = τ.Div
- A process that can only diverge is observationally equivalent to deadlock
- Motivation: if we can only observe a process by communicating with it, 0 and Div are observationally equivalent because both refuse any attempt of communication

Weak bisimilarity - properties

Theorem 5.2 For all LTSs, the relation $\approx$ is

1. an equivalence relation,
2. the largest weak bisimulation and
3. satisfies the following property:

   $s_1 \approx s_2$ iff for each action α,

   - if $s_1 \xrightarrow{\alpha} s_1'$, then there is a transition $s_2 \stackrel{\hat{\alpha}}{\Rightarrow} s_2'$ such that $s_1' \approx s_2'$;
   - if $s_2 \xrightarrow{\alpha} s_2'$, then there is a transition $s_1 \stackrel{\hat{\alpha}}{\Rightarrow} s_1'$ such that $s_1' \approx s_2'$.

Weak bisimilarity - equivalences

Exercise 5.19 *Show that, for all P, Q, the following equivalences, that are usually referred to as* Milner's τ-laws, *hold:*

$$\alpha.\tau.P \approx \alpha.P \tag{18}$$

$$P + \tau.P \approx \tau.P \tag{19}$$

$$\alpha.(P + \tau.Q) \approx \alpha.(P + \tau.Q) + \alpha.Q \ . \tag{20}$$

Weak bisimilarity - congruence

- Unlike strong bisimilarity, weak bisimilarity is not a congruence.
- Note that 0 is equivalent to $\tau.0$, but
  - $a.0 + 0$ is not equivalent to $a.0 + \tau.0$

Weak bisimilarity - congruence

Proposition 5.3 Let P, Q, R be CCS processes. Assume that $P \approx Q$. Then

- $\alpha.P \approx \alpha.Q$, for each action α;
- $P \mid R \approx Q \mid R$ and $R \mid P \sim R \mid Q$, for each process R;
- $P[f] \approx Q[f]$, for each relabelling f; and
- $P \setminus L \approx Q \setminus L$, for each set of labels L.

Hennessy-Milner logic

- Observational semantics can be used to check the correctness of a system w.r.t. its specification
- However, to adopt this verification technique, we are forced to specify the overall behaviour of the system
- E.g. we want to check if the system can perform an a-labelled transition now
  - Rephrasing this requirement in terms of observational equivalence is at best unnatural (or impossible)

Behavioural properties

- The mathematician
  - Is not willing to drink tea now
  - Is willing to drink both coffee and tea now
  - Is willing to drink coffee, but not tea, now
  - Always produces a theorem after drinking coffee
- It's easier to check thes properties by exploring the state space of the process, rather than by trasforming them in equivalence checking problems.

Behavioural properties

- To check behavioural properties, we need
  - A language for expressing them
  - Equipped with a formal syntax and semantics
  - The formal semantics also allows us to overcome the imprecision of natural language
  - “the mathematician is willing to drink both coffee and tea now”
    - M can perform either a coffee-labelled or a tea-labelled transition?
    - M can perform such transitions one after the other?

Model checking

- Systems are specified by CCS processes
- Properties are specified in Hennessy-Milner logic (HML)

Hennessy-Milner formulae

Definition 6.1 The set of Hennessy-Milner formulae over a set of actions Act (from now on referred to as $\mathcal{M}$) is given by the following abstract syntax:

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

where $a \in \mathsf{Act}$. If $A = \{a_1, \ldots, a_n\} \subseteq \mathsf{Act}$ ($n \geq 0$), we use the abbreviation $\langle A \rangle F$ for the formula $\langle a_1 \rangle F \vee \ldots \vee \langle a_n \rangle F$ and $[A]F$ for the formula $[a_1]F \wedge \ldots \wedge [a_n]F$. (If $A = \emptyset$, then $\langle A \rangle F = ff$ and $[A]F = tt$.)

Meaning of formulae

- All processes satisfy tt.
- No process satisfies ff.
- A process satisfies $F \wedge G$ (respectively, $F \vee G$) iff it satisfies both F and G (respectively, either F or G).
- A process satisfies $\langle a \rangle F$ for some $a \in \mathsf{Act}$ iff it affords an a-labelled transition leading to a state satisfying F.
- A process satisfies $[a]F$ for some $a \in \mathsf{Act}$ iff all of its a-labelled transitions lead to a state satisfying F.

Meaning of formulae

- Formula <a>F states that it is **possible** to perform action a and thereby satisfy property F
- Formula [a]F states that no matter how a process performs action a, the state it reaches in doing so will **necessarily** have property F
- The semantics of a formula consists of a the set of processes which satisfy the formula

Semantics of formulae

Definition 6.2 We define $[\![F]\!] \subseteq \mathsf{Proc}$ for $F \in \mathcal{M}$ by:

1. $[\![tt]\!] = \mathsf{Proc}$,
2. $[\![ff]\!] = \emptyset$
3. $[\![F \wedge G]\!] = [\![F]\!] \cap [\![G]\!]$,
4. $[\![F \vee G]\!] = [\![F]\!] \cup [\![G]\!]$,
5. $[\![\langle a \rangle F]\!] = \langle \cdot a \cdot \rangle [\![F]\!]$,
6. $[\![[a]F]\!] = [\cdot a \cdot][\![F]\!]$,

where we use the set operators $\langle \cdot a \cdot \rangle, [\cdot a \cdot] : \mathcal{P}(\mathsf{Proc}) \to \mathcal{P}(\mathsf{Proc})$ defined by

$$\begin{aligned} \langle \cdot a \cdot \rangle S &= \{p \in \mathsf{Proc} \mid \exists p'.\, p \xrightarrow{a} p' \text{ and } p' \in S\} \quad \text{and} \\ [\cdot a \cdot] S &= \{p \in \mathsf{Proc} \mid \forall p'.\, p \xrightarrow{a} p' \implies p' \in S\}. \end{aligned}$$

We write $p \models F$ iff $p \in [\![F]\!]$.

Two formulae are *equivalent* if, and only if, they are satisfied by the same processes in every transition system.

Expressing behavioural properties in HML

- The mathematician is willing to drink coffee now
  - The mathematician has the possibility of performing a coffee-labelled transition
  - <coffee>F
  - Formula F should be satisfied by the mathematician after having drunk the coffee
  - Since we are requiring nothing of the subsequent behaviour of the mathematician, take F = tt

Expressing behavioural properties in HML

- The formula <coffee>tt is satisfied exactly by all processes having an outgoing coffee-labelled transition

$$
\begin{aligned}
[\![\langle \mathrm{coffee} \rangle tt]\!] &= \langle \cdot \mathrm{coffee} \cdot \rangle [\![tt]\!] \\
&= \langle \cdot \mathrm{coffee} \cdot \rangle \mathsf{Proc} \\
&= \{P \mid P \xrightarrow{\mathrm{coffee}} P' \text{ for some } P' \in \mathsf{Proc}\}
\end{aligned}
$$

Expressing behavioural properties in HML

- The mathematician cannot drink tea now
- [tea]ff
- All the states that a process can reach by performed a tea-labelled transition must satisfy ff
- Since no state satisfies ff, the only way that a process can satisfy [tea]ff is that it has no tea-labelled transition.

Exercise

- Find a formula which is satisfied by a.b.0 + a.c.0 but not by a.(b.0 + c.0)
- Gvien two non-bisimilar processes, does there exist a formula that distinguishes them?
- If two processes satisfy the same formulae, are they guaranteed to be strongly bisimilar?

HML and strong bisimilarity

Definition 6.3 [Image Finite Process] A process P is *image finite* iff the collection $\{P' \mid P \xrightarrow{a} P'\}$ is finite for each action a.

An LTS is image finite if so is each of its states.

Theorem 6.1 [Hennessy and Milner [9]] Let $(\mathsf{Proc}, \mathsf{Act}, \{\xrightarrow{a} \mid a \in \mathsf{Act}\})$ be an image finite LTS. Assume that P, Q are states in Proc. Then $P \sim Q$ iff P and Q satisfy exactly the same formulae in $\mathcal{M}$.

HML and strong bisimilarity

- A consequence of the theorem is that if two image finite processes are not strongly bisimilar, then there exists a formula that tells us the reason why they are not
- The proof of the theorem provides a constructive method to exhibit the distinguishing formula

Thank you!