Communication and Concurrency: CCS

Why calculi?

• Prove properties on programs and languages
• Principle: tiny syntax, small semantics, to be handled on paper or mechanically
• Prove properties on the principles of a language or a programming paradigm

• Examples: lambda calculus, sigma calculus, …
Static semantics : examples

- Checks non-syntactic constraints
- compiler front-end :
  - declaration and utilisation of variables,
  - typing, scoping, … static typing => no execution errors ???
- or back-ends :
  - optimisers
- defines legal programs :
  - Java byte-code verifier

What can we do/know about a program without executing it?
Dynamic semantics

• Gives a meaning to the program (a semantic value)
• Describes the behaviour of a (legal) program
• Defines a language interpreter

\[ |- e -> e' \]
\[
\text{let } i = 3 \text{ in } 2 \times i \rightarrow 6
\]

Objective = to prove properties on Program execution
(determinacy, subject reduction, …)
The different semantic families

- **Denotational semantics**
  - mathematical model, high level, abstract

- **Axiomatic semantics**
  - provides the language with a theory for proving properties / assertions of programs

- **Operational semantics**
  - computation of the successive states of an abstract machine
  - used to build evaluators, simulators.
What about concurrency and communication?

- Different timing (synchronous/asynchronous …)
- Different programming models (what is the unit of concurrency? What is sufficient to characterize an execution?…?)
- Interaction between communication/concurrency/shared memory!

Through CCS, this course is a simple study of synchronous communications
SEMANTICS
Operational Semantics

- Describes the computation
- States and configuration of an abstract machine:
  - Stack, memory state, registers, heap...
- Abstract machine transformation steps
- Transitions: current state -> next state
- Several different operational semantics
Natural Semantics: big steps (Kahn 1986)

• Defines the results of evaluation.
• Direct relation from programs to results
  \[ \text{env} \vdash \text{prog} \Rightarrow \text{result} \]
  - \text{env}: binds variables to values
  - \text{result}: value given by the execution of \text{prog}

Reduction Semantics: small steps

describes each elementary step of the evaluation

• rewriting relation: reduction of program terms
• stepwise reduction: \(<\text{prog}, s> \Rightarrow <\text{prog’}, s ’>\)
  – infinitely, or until reaching a normal form.
Labelled Transition Systems (LTS)

• Basic model for representing reactive, concurrent, parallel, communicating systems.
• Definition:
  - $< S, s_0, L, T >$
  - $S =$ set of states
  - $S_0 \in S =$ initial state
  - $L =$ set of labels (events, communication actions, etc)
  - $T \subseteq S \times L \times S =$ set of transitions
  - Notation: $s_1 \xrightarrow{a} s_2 = (s_1, a, s_2) \in T$
An example
Deduction Rules

\[
\frac{P \rightarrow Q \quad P}{Q}
\]

\[
\frac{P}{P \lor Q}
\]

\[
\frac{Q}{P \lor Q}
\]
CCS – SYNTAX AND SEMANTICS
CCS syntax

• Channel names: \( a, b, c, \ldots \)
• Co-names: \( \bar{a}, \bar{b}, \bar{c}, \ldots \)
• Silent action: \( \tau \)
• Actions: \( \mu ::= a \mid \bar{a} \mid \tau \)
• Processes:
  \[ P, Q ::= 0 \quad \text{inaction} \]
  \[ \mu \cdot P \quad \text{prefix} \]
  \[ P \mid Q \quad \text{parallel} \]
  \[ P + Q \quad \text{(external) choice} \]
  \[ (\nu a)P \quad \text{restriction} \]
  \[ \text{rec}_K P \quad \text{process } P \text{ with definition } K = P \]
  \[ K \quad \text{(defined) process name} \]
A tiny example

\[ rec_{C1}(Tick.C1) \]

Figure: The transition graph for \( C1 \)

Labelled graph
- vertices: process expressions
- labelled edges: transitions
- Each derivable transition of a vertex is depicted
- Abstract from the derivations of transitions
CCS : behavioural semantics (1)

Operators and rules

- Action prefix:
  \[ \mu. P \xrightarrow{\mu} P \]

- Communication:
  \[ P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q' \]
  \[ P \parallel Q \xrightarrow{\tau} P' \parallel Q' \]

- Parallelism
  \[ P \xrightarrow{\mu} P' \]
  \[ P \parallel Q \xrightarrow{\mu} P' \parallel Q \]
  \[ Q \xrightarrow{\mu} Q' \]
  \[ P \parallel Q \xrightarrow{\mu} P \parallel Q' \]
CCS: behavioural semantics (2)

Operators and rules

• Non-deterministic choice

\[
Q \xrightarrow{\mu} Q' \\
\overline{P + Q} \xrightarrow{\mu} Q'
\]

\[
P \xrightarrow{\mu} P' \\
P + Q \xrightarrow{\mu} P'
\]

• Scope restriction

\[
P \xrightarrow{\mu} P' \\
\mu \neq a, \overline{a}
\]

\[
(\forall a)P \xrightarrow{\mu} (\forall a)P'
\]

• Recursive definition

\[
P[\text{rec}_K P / K] \xrightarrow{\mu} P' \\
\text{rec}_K P \xrightarrow{\mu} P'
\]
Derivations
(construction of each transition step)

\[ a.P \quad \xrightarrow{a} \quad P \]

\[ a.P \mid Q \quad \xrightarrow{a} \quad P \mid Q \]

\[ (a.P \mid Q) \mid \overline{a}.R \quad \xrightarrow{\tau} \quad (P \mid Q) \mid R \]

Another one:
Par-L(Par_L(Prefix))

One amongst 3 possible derivations

\[ (a.P \mid Q) \mid \overline{a}.R \quad \xrightarrow{a} \quad (P \mid Q) \mid \overline{a}.R \]
EQUIVALENCES
Behavioural Equivalences

• Intuition:
  - Same possible sequences of observable actions
  - Finite / infinite sequences
  - Various refinements of the concept of observation

• Definition: Trace Equivalence
  For a LTS \((S, s_0, L, T)\) its Trace language \(T\) is the set of finite sequences \(\{ (t = t_1, \ldots, t_n \text{ such that } \exists s_0, \ldots, s_n \in S^{n+1}, \text{ and } (s_{n-1}, t_n, s_n) \in T \}\} \)

  Two LTSs are Trace equivalent iff their Trace languages are equal.

  Corresponding Ordering: Trace inclusion
Trace Languages, Examples

- Those 2 systems are trace equivalent:

  $T = \{(), (a), (a,b), (a,c)\}$

- A trace language can be an infinite set:

  $T = \{(), (a), (a,a), (a,...,a), \ldots (a,b), (a,a,b), (a,a,...,a,b), \ldots \}$
Bisimulation

• Behavioural Equivalence
  - non distinguishable states by observation:
    two states are equivalent if for all possible transitions labelled
    by the same action, there exist equivalent resulting states.

• Bisimulations
  \( R \subseteq S \times S \) is a simulation iff
  - It is a equivalence relation
  - \( \forall (p,q) \in R, \)
    \( (p,l,p') \in T \implies \exists q' (q,l,q') \in T \) and \( (p',q') \in R \)
  - \( R \) is a bisimulation if the same condition hold with \( q \) too:
    \( \forall (p,q) \in R, \)
    \( (q,l,q') \in T \implies \exists q' (q,l,q') \in T \) and \( (p',q') \in R \)

• \( \sim \) is the coarsest bisimulation
  2 LTS are bisimilar iff their initial states are in \( \sim \)
  quotients = canonical normal forms
Transitivity

• If $R$, $S$ are bisimulations, then so is their composition $RS = \{(P, P') \mid \exists Q. P R Q \text{ and } Q S P'\}$

• In particular, $\sim \subseteq \sim$, i.e., bisimilarity is transitive.
Bisimulation

- More precise than trace equivalence:
  - Preserves deadlock properties.
  - Can be built by adding elements in the equivalence relation.
  - Coinductive definition (biggest set verifying ...)

No state in B is equivalent to A1.
Bisimulation

- Congruence laws:
  \[ P_1 \sim P_2 \Rightarrow a.P_1 \sim a:.2 \quad (\forall P_1, P_2, a) \]
  \[ P_1 \sim P_2, \quad Q_1 \sim Q_2 \Rightarrow P_1 + Q_1 \sim P_2 + Q_2 \]
  \[ P_1 \sim P_2, \quad Q_1 \sim Q_2 \Rightarrow P_1 | Q_1 \sim P_2 | Q_2 \]
  Etc…

- \( \sim \) is a congruence for all CCS operators:

  for any CCS context \( C[.], \) \( C[P] \sim C[Q] \iff P \sim Q \)

  Basis for compositional proof methods

- Maximal trace is not an equivalence
Observational Equivalences

- Weak bisimulation
  - Abstraction: hidden actions
  - allows for arbitrary many internal actions $\mu \Rightarrow$

$\tau \quad \tau^*$

$\tau^*$ \quad \text{act} \quad \tau^*$
Weak bisimulation

- The following def is a tractable version of weak bisimulation:
  A weak bisimulation is a relation $R$ such that
  $P \ R \ Q \Rightarrow \forall \mu, P, P' (P \rightarrow P' \overset{\mu}{\Rightarrow} \exists Q'. Q \overset{\mu}{\Rightarrow} Q' \text{ and } P' \ R \ Q')$

and conversely

- Note the dissymmetry between the use of $\rightarrow$ on the left and of $\Rightarrow$ on the right

- Two processes are weakly bisimilar if (notation $P \approx Q$) if there exists a weak bisimulation $R$ such that $P \ R \ Q$. 
Branching bisimulation

- only staying in equivalent states

Still existence of a canonical minimal automata
Computation is polynomial
ADDITIONAL NOTATIONS AND CONSTRUCTS
Alternative Notations

• Alternative Notations

\[ \text{def} \]

\[ \text{rec}_{C_1}(\text{Tick}.C1) \iff \text{Cl} \overset{\text{def}}{=} \text{tick.Cl} \]

a little more complex for several definitions

-> exercise?

• Input/output: \( a=?a ; a = !a \)

• | or ||
Extension: Parameterized actions

- **input of data at port a**: \(a(x).E\)
- \(a(x)\) binds free occurrences of \(x\) in \(E\).
- **Port a** represents \(\{a(v) : v \in D\}\) where \(D\) is a family of data values.
- **Output of data at port a**: \(\overline{a}(e).E\) where \(e\) is a data expression.
- **Transition Rules**: depend on extra machinery for expression evaluation. Let \(\text{Val}(e)\) be data value in \(D\) (if there is one) to which \(e\) evaluates.
  - **R (in)**: \(a(x).E \xrightarrow{a(v)} E \{v/x\} \) if \(v \in D\) where \(\{v/x\}\) is substitution
  - **R (out)**: \(a(e).E \xrightarrow{\overline{a}(v)} E \) if \(\text{Val}(e) = v\)
Example: a register

\[ \text{Reg}_i = \text{read}(i) \cdot \text{Reg}_i + \text{write}(x) \cdot \text{Reg}_x \]
EXAMPLES
Example: dining philosophers

(ch_{\text{idling,eating}} \cdot (\text{idle.idling} + \text{take_left.take_right.eating} + \text{take_right.take_left.eating},

\text{eat.eating} + \text{drop_left.drop_right.idling} + \text{drop_right.drop_left.idling})

Deadlock or not?  
Mutual exclusion?
(trivial) example: Milner’s Scheduler

- Processes iteratively start and finish executing tasks (one task per process)
- Task starts are cyclically ordered

```plaintext
cycler = \overline{\alpha}.\text{start}(\beta.0 || \text{end.cycler})
scheduler_3 = \text{local } \alpha_1, \alpha_2, \alpha_3 \text{ in }
(\ [\alpha_1/\alpha, \alpha_2/\beta, \text{start1/start, end1/end}] \text{ cycler }
|| [\alpha_2/\alpha, \alpha_3/\beta, \text{start2/start, end2/end}] \text{ cycler }
|| [\alpha_3/\alpha, \alpha_1/\beta, \text{start3/start, end3/end}] \text{ cycler }
|| \alpha_1.0)
```

**vérification des propriétés ?**
Scheduler_2 reduced
CONCLUSION

• A synchronous communication language
• A (complex but) efficient notion of equivalence on processes
• What is missing?
  – Channel communication (like in pi-calculus) -> much more complex
  – No computational construct by nature
EXERCISES
Example: Alternated Bit Protocol

Hypotheses: channels can lose messages

Requirement: the protocol ensures no loss of messages

Write in CCS?
Example: Alternated Bit Protocol (2)

• \textbf{emitter} =
  \begin{align*}
  & \text{let rec } \{\text{em0} = \text{?ack1 :em0 + ?imss:em1} \\
  & \text{and em1} = \text{!in0 :em1 + ?ack0 :em2} \\
  & \text{and em2} = \text{?ack0 :em2 + ?imss :em3} \\
  & \text{and em3} = \text{!in1 :em3 + ?ack1 :em0} \}
  \\
  & \text{in em0}
  \end{align*}

• \textbf{ABP} = \text{local } \{\text{in0, in1, out0, out1, ack0, ack1, …} \}
  \text{in emitter || Fwd\_channel || Bwd\_channel || receiver}
Example: Alternated Bit Protocol (3)

Channels that loose and duplicate messages (in0 and in1) but preserve their order?

• Exercise:
1) Draw an LTS describing the loosy channel behaviour
2) Write the same description in CCS
Exercise 2

- $\text{rec}_K \text{coin.}(\text{coffee} \cdot \text{ccup}.K + \text{tea} \cdot \text{tcup}.K)$
- $\text{coin.} \text{rec}_K (\text{coffee} \cdot \text{ccup}.\text{coin}.K + \text{tea} \cdot \text{tcup}.\text{coin}.K)$
- $\text{rec}_K (\text{coin} \cdot \text{coffee} \cdot \text{ccup}.K + \text{coin} \cdot \text{tea} \cdot \text{tcup}.K)$

Question: which of these machines can we safely consider equivalent?

Note that these machines have all the same traces.
Exercice 3 : Bisimulations

Are those 3 LTSs equivalent by:
- Strong bisimulation?
- Weak bisimulation?

In each case, give a proof.
Exercice 3 : Bisimulation

- Exercice:
  1) Compute the strong minimal automaton for A₁.
  2) Compute the weak minimal automaton for A₁.
Exercise 5

- Compare the construct $\mathit{def}$ and $\text{rec}_K$:

1. Let us start by a simple pair of processes

\[
A \overset{\text{def}}{=} \overline{a}.A + b.B
\]

\[
B \overset{\text{def}}{=} a.A
\]

2. Suppose $\text{rec}$ can accept several variables:

\[
\text{rec} (K=P,L=Q) \text{ express the same term}
\]

3. Is it possible to express the same thing with a single variable $K$?

Here are some possible hints:

- Define a recursive process $\text{All}$ that contains $A$ and $B$ and can trigger each of them by the reception of a message on channel $cA$ or $cB$
- (we suppose $cA$ and $cB$ cannot be used elsewhere)
- What kind of equivalence between the two expressions do you have?
Exercice: Alternated Bit Protocol

**Correction (1):**

Channels that lose and duplicate messages (in0 and in1) but preserve their order?

1) Draw an automaton describing the loopy channel behaviour

- It is a symmetric system, receiving ?in0 and ?in1 messages, then delivering 0, 1 or more times the corresponding !out0 or !out1 message.
- On each side (bit 0 or 1), the initial state has a single transition for the reception.
- In the next state, it can either: return silently to the initial state (= lose the message), deliver the message and return to the initial state (exactly one delivery), or deliver the message and stay in the same state (thus enabling duplication).
Exercice: Alternated Bit Protocol

Correction (2):

Channels that lose and duplicate messages (in0 and in1) but preserve their order?

2) Write it in CCS

• Lousy channel =

\[
\text{let rec } \{ \text{ch0} = \text{in0} : \text{ch1} + \text{in1} : \text{ch2} \\
\quad \text{and ch1} = \tau : \text{ch1} + \tau : \text{ch0} + \text{out0} : \text{ch1} + \text{out0} : \text{ch0} \\
\quad \text{and ch2} = \tau : \text{ch2} + \tau : \text{ch0} + \text{out0} : \text{ch2} + \text{out0} : \text{ch0} \\
\} \\
\text{in ch0}
\]
Exercice: Alternated Bit Protocol

Correction (3):

Channels that lose and duplicate messages (in0 and in1) but preserve their order?

Other Solutions:

More generally, parameterized model:
Exercice 2 : Bisimulations

Are those 3 LTSs equivalent by:

- **Strong bisimulation?**
  
  NO! Need find non equivalent states. E.g. counter example for 1 \( \neq \) 2:

  States 1.0 and 1.1 are different because 1.0 can do \(?\text{in}0\) and 1.1 cannot.

  Then 1.1 and 2.1 are different because 1.1 can do \(!\text{out}0\) \(\rightarrow\) 1.0, while no 2.1 \(!\text{out}0\) transitions can go to a state equivalent to 1.0.

- **Weak bisimulation?**
  
  YES. Exhibit a partition of equivalent states:

  \(1 = \{1.0, 2.0\}\), \(2 = \{1.1, 2.1\}\)

  Check all possible \((\tau^*a\tau^*)\) transitions:

  1 - \(!\text{in}0\) \(\rightarrow\) 2, \ldots, 2 - \(!\text{out}0\,\tau^*\) \(\rightarrow\) 1

  Remark: this transition set defines the minimal representant modulo weak bisimulation…
Exercice 4 : Produit synchronisé

Compute the synchronized product of the LTS representing the ABP emitter with the (forward) Channel:

Local \{in0, in1\} in
(Emitter || Channel)
Exercice 4 : Produit synchronisé

Correction ? partially...

local \{in0, in1\} in
(Emitter \parallel Channel)
Exercice 4 : Produit synchronisé

Correction ? Tool generated LTS...