Concurrent Computational Models

6CCS3COM Computational Models

Josh Murphy

Contents

Sequential models of computation

- Turing machines and lambda calculus are **sequential models** of computation.
- Interaction nets allow for **distribution** and **parallel** computation, but do not account for concurrent interactions.
- They are all universal models of computation — so why care if they specifically allow for concurrency?
  - Concurrent hardware.
  - Concurrent models can be more intuitive for concurrent problems.

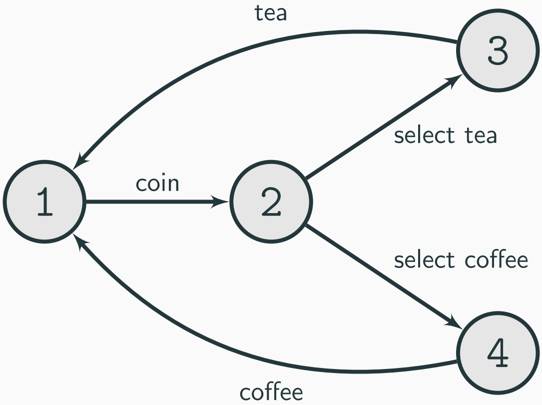
Properties of concurrent computation

- Concurrent computation can be characterised with having the following properties.
  - **Parallelism:** steps of computation can occur simultaneously.
  - **Interference:** the meaning of a program may depend on the behaviour of other programs that are being executed.
  - **Non-determinism:** the same computations do not always produce the same results.
  - **Non-termination:** the program(s) may not terminate, and indeed the output of a program may not be the focus of the computation.

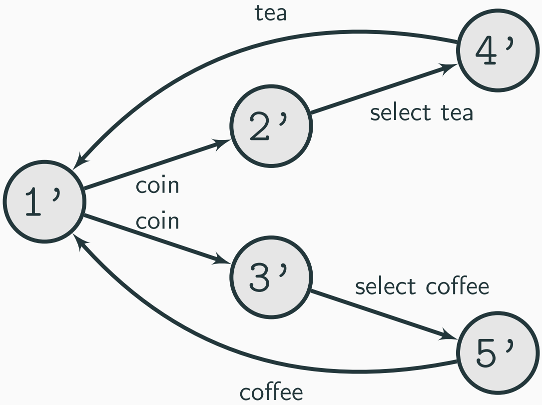
Notions for concurrent computation

- **Process:** A process is an entity that undertakes computation. In different domains they may also be referred to as agents, component, or thread.
- **Communication:** Processes can exchange information directly between themselves.
- **Interaction:** Shared resources between processes can mean the collective behaviour of the system depends on an individual's actions. Interaction can be positive and intended, or negative interference.
- **Behaviour:** The observable processes are called the behaviour of the system. Behaviour replaces the notion of *result* from sequential programs.

Example: vending machine



Example: vending machine (deterministic)



Equivalence

- How do we know if programs are **equivalent**?
    - Two *sequential* programs are equivalent if for all inputs they give the same outputs.

$$f = g \text{ iff } \forall x, f(x) = g(x)$$

    - But in concurrent programs the output is rarely the focus, and is more commonly non-deterministic/non-terminating...

- For concurrent programs, we could:
    - Compare their associated language? No (accepted words are not the only thing we care about with concurrent systems).
    - Compare their *behaviour*? First, we need some formal definitions.

Transition system

- We can describe a concurrent program with a particular kind of automaton.

- Let *Act* be an alphabet (infinite set of **labels**):

  $$Act = \mathcal{N} \cup \bar{\mathcal{N}}$$

  $\mathcal{N}$ is the set of **actions**, and $\bar{\mathcal{N}}$ is the set of **co-actions**.
  For example, for a vending machine, $\mathcal{N} = \{coin, coffee, tea, ...\}$ and $\bar{\mathcal{N}} = \{\overline{coin}, \overline{coffee}, \overline{tea}, ...\}$

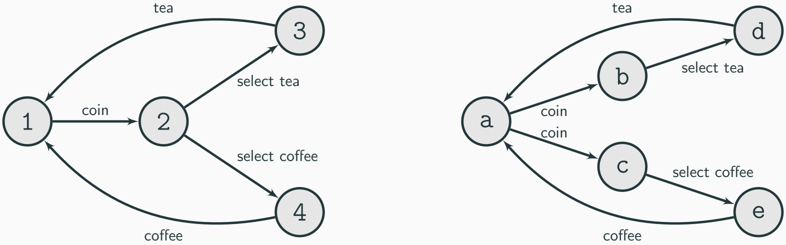
- A **transition system** with labels in *Act* is a pair (Q, T) where:
  - Q is a set of states,
  - T is a ternary relation $T \subseteq (Q \times Act \times Q)$, the **transition relation**.

  We write $q \rightarrow^a q'$ if $(q, a, q') \in T$, and say that in the state q the process can perform the action a and move to state q'.

Simulation

- Let (Q, T) be a labelled transition system on Act.
- A binary relation S on Q is a **strong simulation** if pSq implies that, for each $a \in Act$ such that $p \to^a p'$, there exists q' in Q such that $q \to^a q'$ and $p'Sq'$.
- The idea is that if pSq holds, any transition can be done from the state p can also be done from q, and the resulting states are still in the relation.
- If pSq holds we say p **simulates** q.

Simulation: example



- For these two transition systems we can show a simulation:
  - $S = \{(a, 1), (b, 2), (c, 2), (d, 3), (e, 4)\}$
- Check that, for each pair $(p, q) \in S$ and for each each action a such that $p \rightarrow^a p'$ there is a transition $q \rightarrow^a q'$ such that $(p', q') \in S$.
- This shows the deterministic vending machine scan simulate the non-deterministic vending machine.
  - But the non-deterministic vending machine can't simulate the deterministic vending machine. **Exercise:** why not?

Simulation

- Simulation allows us to compare transition systems, but just because there is simulation in a single direction it doesn't mean the systems are equivalent.
- To show equivalence there needs to be simulation in both directions.

Bisimulation

- Let (Q, T) be a labelled transition system on Act.
- A binary relation S on Q is a **strong bisimulation** if S and S^{-1} are strong simulations, where S^{-1} denotes the inverse of S (i.e., $pS^{-1}q$ if qSp).
- We will write $p \sim q$ if there is a strong bisimulation S such that $(p, q) \in S$.
- $p \sim q$ implies p simulates q and q simulates p, but the reverse is not true. **Exercise:** why not?

The calculus of communicating systems (CCS)

CCS introduction

- Introduced by Milner in 1980s to have an abstract model that focuses on the *essence* of concurrent systems.
- Milner's general model:
    - A concurrent system is a collection of processes
    - A process is an independant agent that may perform internal activities in isolation, or may interact with the environment to perform shared activities.

A simple example

- A vending machine can be represented as follows:

$$coke.coin.\overline{coke\_can}.0 + chocolate.coin.\overline{chocolate\_bar}.0$$

- With the following elements of syntax:
  1. **Actions:** $coke, \overline{coke\_can}, ...$
  2. **Sequential composition:** . (the dot)
     - $a.b$, first do a, then do b
  3. **Non-deterministic choice:** $+$
     - $c + d$, do c or d
  4. **Terminal process:** 0

- Processes **perform actions** ($\rightarrow^{action}$), and become a new process:

$$\begin{aligned} coke.coin.\overline{coke\_can}.0 + chocolate.coin.\overline{chocolate\_bar}.0 \\ \rightarrow^{coke} coin.\overline{coke\_can}.0 \\ \rightarrow^{coin} \overline{coke\_can}.0 \\ \rightarrow^{\overline{coke\_can}} 0 \end{aligned}$$

CCS syntax: terminal process

- The simplest possible process is a terminal process, which is a **terminated**, **inactive**, or **deadlocked** process.
- Represented by 0

CCS syntax: Actions

- A set of **labels** for actions:
  $Act = \mathcal{N} \cup \overline{\mathcal{N}} = \{a, b, c, ...\} \cup \{\overline{a}, \overline{b}, \overline{c}, ...\}$
- **Input actions:** $i \in \mathcal{N}$ represents the receiving of an input.
- **Output actions:** $o \in \overline{\mathcal{N}}$ represent the sending of an output.

CCS syntax: Action prefixing

- The simplest behaviour is **sequential action**
- If P is a process, we write $\alpha.P$ to denote the **prefixing** of P with action α.
- $\alpha.P$ models a system that is ready to perform the action α and then it will behave as P
    - i.e. $\alpha.P \rightarrow^{\alpha} P$

CCS syntax: non-deterministic choice

- Alternative behaviours can be used to model **non-deterministic choice**.
- If P and Q are processes, we write $P + Q$ to denote the **non-deterministic choice** between P and Q.
- $P + Q$ models a process that can behave as P (discarding Q), or behave as Q (discarding P).

$$\begin{aligned} &\mathit{coke}.\mathit{coin}.\overline{\mathit{coke\_can}}.0 + \mathit{chocolate}.\mathit{coin}.\overline{\mathit{chocolate\_bar}}.0 \\ &\quad \rightarrow^{\mathit{coke}} \mathit{coin}.\overline{\mathit{coke\_can}}.0 \end{aligned}$$

- Note: $P + Q = Q + P$

CCS syntax: constants and recursion

- We can define **process constants**, that can be defined recursively.
- $P ::= a.b.c.0$, P is defined as a constant for the process $a.b.c.0$
- e.g., A ticking clock: $C ::= tick.tock.C$
  $tick.tock.C \rightarrow^{tick} tock.C \rightarrow^{tock} C = tick.tock.C \rightarrow^{tick} tock.C \rightarrow^{tock} ...$
- Allows processes to continue *ad infinitum*.
- A program is a collection of process contants.

CCS syntax: parallel composition

- **Concurrent behaviour** can be modelled with **parallel composition**.
- If P and Q are processes, we write $P|Q$ to denote the **parallel composition** of P and Q.
- $P|Q$ models a process that behaves like P and Q in parallel:
  - Each may proceed independently
  - If P is ready to perform an action a and Q is ready to perform the complementary action $\overline{a}$, they may **interact** ($\rightarrow^{\tau}$, a step internal to the process).
  - Note: $P|Q = Q|P$

CCS syntax: parallel composition example

- First, let us update the model of our hot drinks machine...

$$HDM ::= tea.coin.\overline{cup\_of\_tea}.HDM + coffee.coin.coin.\overline{mug\_of\_coffee}.HDM$$

- Now, consider the concurrent system, me...

$$JM ::= \overline{tea}.\overline{coin}.cup\_of\_tea.\overline{teach}.JM + \overline{coffee}.\overline{coin}.\overline{coin}.mug\_of\_coffee.\overline{research}.JM$$

CCS syntax: parallel composition example

- We can now compose a process for the average day:

$$HDM \mid JM$$

- How might the process behave?

$$\begin{gathered}(tea.coin.\overline{cup\_of\_tea}.HDM + coffee.coin.coin.\overline{mug\_of\_coffee}.HDM) \mid \\ (\overline{tea}.\overline{coin}.cup\_of\_tea.\overline{teach}.JM + \\ \overline{coffee}.\overline{coin}.\overline{coin}.mug\_of\_coffee.\overline{research}.JM)\end{gathered}$$

$$\rightarrow^{\tau} (coin.\overline{cup\_of\_tea}.HDM) \mid (\overline{coin}.cup\_of\_tea.\overline{teach}.JM)$$

$$\rightarrow^{\tau} (cup\_of\_tea.HDM) \mid \overline{cup\_of\_tea}.\overline{teach}.JM$$

$$\rightarrow^{\tau} HDM \mid \overline{teach}.JM$$

$$\rightarrow^{teach} HDM \mid JM$$

- **Exercise:** trace an execution that includes the action $\rightarrow^{research}$

CCS syntax: Restriction

- We can control **unwanted interactions** between a process and the environment by **restricting** action/co-action pairs.
- If P is a process and a is an action we write $\nu a.P$ for the restriction of a in P.
  - P can no longer take action a or $\overline{a}$.

CCS syntax: Restriction example

- Restricting the HDM on coffee makes the coffee button inaccessible to JM: $\nu \mathit{coffee}.(\mathit{HDM} \mid \mathit{JM})$
- As a consequence, JM can only teach, and never research

Summary of CCS syntax

- We can define a grammar for CCS processes as follows.

$$
\begin{array}{lll}
P \rightarrow & K & | \quad \text{Process constants} \\
& \alpha.P & | \quad \text{Action prefixing } (\alpha \in Act) \\
& \Sigma_{i \in I} P_i & | \quad \text{Summation } (P_0 + P_1 + ... + P_I) \\
& P_1 \mid P_2 & | \quad \text{Parallel composition} \\
& \nu\alpha.P & | \quad \text{Restriction } (\alpha \in Act) \\
& 0 & \quad\;\; \text{Terminal process}
\end{array}
$$

Syntax of CCS: exercises

- **Exercise:** Which of the following expressions are well-formed processes in CCS?
  - $a.b.A + B$
  - $\nu a.(a.0 + \overline{a}.A)$
  - $(a.b.A + \overline{a}.0) \mid B$
  - $((a.b.A) + \overline{a}.0).B$

Formal semantics for CCS

RULE	Premise	Conclusion	Conditions
ACT		$\alpha.P \rightarrow^{\alpha} P$	
SUM	$P \rightarrow^{\alpha} P'$	$M + P \rightarrow^{\alpha} P'$	
COM_1	$P \rightarrow^{\alpha} P'$	$P \mid Q \rightarrow^{\alpha} P' \mid Q$	
COM_2	$Q \rightarrow^{\alpha} Q'$	$P \mid Q \rightarrow^{\alpha} P \mid Q'$	
COM_3	$P \rightarrow^{a} P', Q \rightarrow^{\overline{a}} Q'$	$P \mid Q \rightarrow^{\tau} P' \mid Q'$	
RES	$P \rightarrow^{\alpha} P'$	$\nu\beta.P \rightarrow^{\alpha} \nu\beta.P'$	$\alpha, \overline{\alpha} \neq \beta$
CON	$P \rightarrow^{\alpha} P'$	$K \rightarrow^{\alpha} P'$	$K ::= P$

Exercise: reducing expressions

- **Exercise:** Show a possible run for the following process, identifying the rules used in each step.

$$\nu b.((a.b.P_1 + b.P_2 + c.0) \mid \overline{a}.0) \mid \overline{b}.P_3 + \overline{a}.P_4$$

- **Challenge:** Are CCS process reductions confluent (can different orders of reduction give different results)? Justify your answer.

Exercise: reducing expressions

- **Exercise:** Show a possible run for the following process.

$$\nu b.((a.b.P_1 + b.P_2 + c.0) \mid \overline{a}.0) \mid \overline{b}.P_3 + \overline{a}.P_4$$

- We can use SUM in combination with COM_3 to reduce the expression.
    - SUM: $(a.b.P_1 + b.P_2 + c.0) \rightarrow^{a} b.P_1$
    - COM_3: $(a.b.P_1 + b.P_2 + c.0) \mid \overline{a}.0) \rightarrow^{\tau} b.P_1 \mid 0$
- After reduction step:

$$\nu b.(b.P_1 \mid 0) \mid \overline{b}.P_3 + \overline{a}.P_4$$

- We can use SUM in combination with COM_2 to reduce the expression.
    - SUM: $\overline{b}.P_3 + \overline{a}.P_4 \rightarrow^{\overline{a}} P_4$
    - COM_2: $\nu b.(b.P_1 \mid 0) \mid \overline{b}.P_3 + \overline{a}.P_4 \rightarrow^{\overline{\alpha}} \nu b.(b.P_1 \mid 0) \mid P_4$
- After reduction step:

$$\nu b.(b.P_1 \mid 0) \mid P_4$$

Summary

- Concurrent systems have particular properties, and so bespoke computational models (e.g. CCS) have been developed to represent them.
- Equivalence of concurrent systems can be shown through bisimulation.
- CCS
  - Syntax
  - Semantics
  - *Lecture slides use slightly different notation compared to textbook*