

Concurrency 5

The theory of CCS
Specifications and Verification
Expressive Power

Catuscia Palamidessi
catuscia@lix.polytechnique.fr

Definitions with parameters

- CCS originally had dynamic scope. This is problematic for various reasons, for instance alpha conversion does not hold.
 - Static scope (with constant definitions) is not an option, because it would diminish the expressive power
 - The modern version of CCS [Milner'99] has parametric definitions and no free variables in the body. So the scope is not an issue anymore

Example of parametric definition: $A(a) \equiv \underline{a}.(v b)(a.0 \mid \underline{b}.A(b))$

- In the following, I will consider CCS with parametric definitions.
- Note: I will use \equiv for definitions and \underline{a} for the output action on a .
- Sometime I will omit parameters when they are implicit. Example: $A = a.A$ instead of $A(a) = a.A(a)$

The theory of CCS

- An equational theory, correct wrt observational congruence, which can be used to show that two processes are observationally congruent.
- Equational means that we have the usual laws of equality:
 - Reflexivity $P = P$
 - Symmetry $P = Q \Rightarrow Q = P$
 - Transitivity $P = Q$ and $Q = R \Rightarrow P = R$
 - Congruence $P = Q \Rightarrow C[Q] = C[P]$ for every context $C[]$
- Correct wrt Observational Congruence means that if we can derive $P = Q$, then $P \cong Q$

Proper axioms of the theory

- The dynamic laws

- The dynamic laws involve the dynamic operators (i.e. those operator which are not static in transtions). In *CCS* these are the + and the process names

Monoid laws

1. $P + Q = Q + P$
2. $P + (Q + R) = (P + Q) + R$
3. $P + P = P$
4. $P + 0 = P$

τ laws

1. $\alpha.\tau.P = \alpha.P$
2. $P + \tau.P = \tau.P$
3. $\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$

Proper Axioms of the theory

- The dynamic laws (cont.ed)

Process definitions

1. If $A \equiv P$, then $A = P$
2. If the "hole" in P is "guarded" then if $P = C[P]$, and $Q = C[Q]$, then $P = Q$

"Guarded" means that the hole appears only after a visible action

- The second law is very useful for proving equality of processes defined recursively and for finding solutions of recursive definitions.
- Example: Assume $A \equiv a.A$ and $B \equiv a.a.B$. It's easy to prove that $A = B$. In fact, $A = a.A = a.a.A$ by congruence and then we can apply the above axiom
- Note that the condition about guarded is essential for the unicity of solutions
- Exercise 1: Assume $A \equiv a.A + \tau.A$. Show that any process of the form $\tau.(\tau.P + a.0)$ is a solution

Proper Axioms of the theory

- The static laws

Parallel Composition laws

1. $P \mid Q = Q \mid P$
2. $P \mid (Q \mid R) = (P \mid Q) \mid R$
3. $P \mid 0 = P$

Restriction laws

1. $(\nu a) P = P$ if a is not free in P
2. $(\nu a) (\nu b) P = (\nu b) (\nu a) P$
3. $(\nu a) (P \mid Q) = (\nu a) P \mid (\nu a) Q$
4. $(\nu a) P = (\nu b) P\{b/a\}$ alpha conversion

Proper Axioms of the theory

- Expansion law

$$\begin{aligned} P|Q = & \sum \{ a.(P' | Q) \mid P \xrightarrow{a} P' \text{ for some } a, P' \} \\ & + \\ & \sum \{ b.(P | Q') \mid Q \xrightarrow{b} Q' \text{ for some } b, Q' \} \\ & + \\ & \sum \{ \tau.(P' | Q') \mid P \xrightarrow{a} P' \text{ and } Q \xrightarrow{\underline{a}} Q' \text{ for some } a, P', Q' \} \end{aligned}$$

- The expansion law expresses the parallel operator in terms of nondeterminism and sequentiality (parallelism as interleaving)
- Exercise 2: Assume $A \equiv a.A$ and $B \equiv a.B \mid a.0$. Prove that $A=B$ using the axioms.

Example: A distributed scheduler

- 1,...,n are tasks identifiers. Tasks have to be executed repeatedly, in a cyclic order. There can be more than one task executed at the same time, but the next instance of Task i cannot start before previous instance has finished.
- **Specification:** We use:
 - a_k as the signal **start** to Task k and
 - b_k as the signal that Task k has **terminated**
- **Assume:**
 - $X \subseteq \{1, \dots, n\}$ are the tasks in progress
 - Task i is next

$$\text{ScSpec}(i, X) = \sum \{ b_k \cdot \text{ScSpec}(i, X - \{k\}) \mid k \in X \} \quad \text{if } i \in X$$

$$\begin{aligned} \text{ScSpec}(i, X) = & a_i \cdot \text{ScSpec}(i+1, X \cup \{i\}) \\ & + \\ & \sum \{ b_k \cdot \text{ScSpec}(i, X - \{k\}) \mid k \in X \} \quad \text{if } i \notin X \end{aligned}$$

Example: A distributed scheduler

- **Implementation:** We build the scheduler, Sched, as a ring of n cells each linked to one task
- Cell:

$$\begin{array}{lll} A = a.C & C = c.E & E = b.D + d.B \\ B = b.A & D = d.A & \end{array}$$

Note: A stands for $A(a,b,c,d)$, B stands for $B(a,b,c,d)$, etc.

$$\text{Sched} = (\nu c_1) \dots (\nu c_n) (A(a_1, b_1, c_1, \underline{c_n}) \mid \prod \{D(a_k, b_k, c_k, \underline{c_{k-1}}) \mid k \neq 1\})$$

Proposition (Correctness of the implementation wrt the specification):

$$\text{Sched} = \text{ScSpec}(1, \emptyset)$$

Exercise 3: Prove it.

Example: Unbounded Buffers

- It is possible in CCS to create structures which grow and shrink dynamically. Examples include unbounded queues and stacks.
- Exercise 4: specify an unbounded stack, and then provide an implementation and the proof of correctness as in previous example. You can ignore the actual data, and assume that they are just tokens, so the only relevant info is how many items are in the stack. This is equivalent to a *counter*.
- Exercise 5: Same thing, with an unbounded queue.
- Hint: Follow the intuition of the 2-position buffer seen in previous class, built by concatenating two 1-position buffers.

Expressive power of CCS

- CCS is a Turing-complete formalism, i.e. it is able to express all computable functions.
- To prove this, it is sufficient to show that it is possible to simulate the behavior of Turing machines.
- Assuming that we know already how to implement an unbounded queue, it is convenient to consider the variant definition of Turing machines which use an unbounded queue instead of a tape.
- There is also a variant definition which uses *two stacks* instead of a tape.
- Exercise 6: Show how to simulate a Turing Machine