

Formal Modelling of Software Intensive Systems

CCS

Francesco Tiezzi

University of Camerino
`francesco.tiezzi@unicam.it`

A.A. 2017/2018



CCS Basics

Sequential Fragment

- *Nil* process (the only atomic process)
- action prefixing ($a.P$)
- names and recursive definitions (\triangleq)
- nondeterministic choice ($+$)

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

Parallelism and Renaming

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

CCS Basics

Sequential Fragment

- *Nil* process (the only atomic process)
- action prefixing ($a.P$)
- names and recursive definitions (\triangleq)
- nondeterministic choice ($+$)

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

Parallelism and Renaming

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

CCS Basics

Sequential Fragment

- *Nil* process (the only atomic process)
- action prefixing ($a.P$)
- names and recursive definitions (\triangleq)
- nondeterministic choice ($+$)

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

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
 - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$
(elements of \mathcal{A} are called names and those of $\overline{\mathcal{A}}$ 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*, $\overline{\text{coffee}}$ are actions)
- \mathcal{K} is a set of **process names (constants)** (e.g. CM).

Definition of CCS (expressions)

$P := K$		process constants ($K \in \mathcal{K}$)
$\alpha.P$		prefixing ($\alpha \in Act$)
$\sum_{i \in I} P_i$		summation (I is an arbitrary index set)
$P_1 P_2$		parallel composition
$P \setminus L$		restriction ($L \subseteq \mathcal{A}$)
$P[f]$		relabelling ($f : Act \rightarrow Act$) such that
		<ul style="list-style-type: none">• $f(\tau) = \tau$• $f(\bar{a}) = \overline{f(a)}$

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

Notation

$$P_1 + P_2 = \sum_{i \in \{1,2\}} P_i$$

$$Nil = \sum_{i \in \emptyset} P_i$$

Precedence

Precedence

- 1 restriction and relabelling (tightest binding)
- 2 action prefixing
- 3 parallel composition
- 4 summation

Example: $R + a.P | b.Q \setminus L$ means $R + ((a.P) | (b.(Q \setminus L)))$

Definition of CCS (defining equations)

CCS program

A collection of **defining equations** of the form

$$K \triangleq P$$

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. $A \triangleq \bar{a}.A \mid A$.

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:

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

SOS rules for CCS

($\alpha \in Act$, $a \in \mathcal{L}$)

$$\text{ACT} \quad \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

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

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

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

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

$$\text{RES} \quad \frac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \quad \alpha, \bar{\alpha} \notin L$$

$$\text{REL} \quad \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

$$\text{CON} \quad \frac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'} \quad K \triangleq P$$

Deriving Transitions in CCS

Let $A \triangleq a.A$. Then

$$((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a].$$

Why?

Deriving Transitions in CCS

Let $A \triangleq a.A$. Then

$$((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a].$$

Why?

$$\text{REL} \frac{}{((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a]}$$

Deriving Transitions in CCS

Let $A \triangleq a.A$. Then

$$((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a].$$

Why?

$$\text{REL} \frac{\text{COM1} \frac{}{(A \mid \bar{a}.Nil) \mid b.Nil \xrightarrow{a} (A \mid \bar{a}.Nil) \mid b.Nil}}{((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a]}}$$

Deriving Transitions in CCS

Let $A \triangleq a.A$. Then

$$((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a].$$

Why?

$$\text{REL} \frac{\text{COM1} \frac{\text{COM1} \frac{A \mid \bar{a}.Nil \xrightarrow{a} A \mid \bar{a}.Nil}{(A \mid \bar{a}.Nil) \mid b.Nil \xrightarrow{a} (A \mid \bar{a}.Nil) \mid b.Nil}}{((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a]}}$$

Deriving Transitions in CCS

Let $A \triangleq a.A$. Then

$$((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a].$$

Why?

$$\text{REL} \frac{\text{COM1} \frac{\text{COM1} \frac{\text{CON} \frac{A \triangleq a.A}{A \xrightarrow{a} A}}{A \mid \bar{a}.Nil \xrightarrow{a} A \mid \bar{a}.Nil}}{(A \mid \bar{a}.Nil) \mid b.Nil \xrightarrow{a} (A \mid \bar{a}.Nil) \mid b.Nil}}{((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a]}}$$

Deriving Transitions in CCS

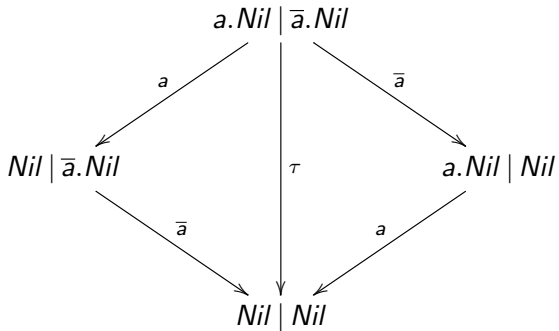
Let $A \triangleq a.A$. Then

$$((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a].$$

Why?

$$\begin{array}{c} \text{ACT} \frac{}{a.A \xrightarrow{a} A} \\ \text{CON} \frac{a.A \xrightarrow{a} A}{A \xrightarrow{a} A} \quad A \triangleq a.A \\ \text{COM1} \frac{}{A \mid \bar{a}.Nil \xrightarrow{a} A \mid \bar{a}.Nil} \\ \text{COM1} \frac{}{(A \mid \bar{a}.Nil) \mid b.Nil \xrightarrow{a} (A \mid \bar{a}.Nil) \mid b.Nil} \\ \text{REL} \frac{}{((A \mid \bar{a}.Nil) \mid b.Nil)[c/a] \xrightarrow{c} ((A \mid \bar{a}.Nil) \mid b.Nil)[c/a]} \end{array}$$

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



CCS: vending machine example



Examples at the blackboard. . .

CCS in pseuCo

pseuCo

Web application allowing to create CCS specifications and interactively explore the resulting transition systems

The screenshot displays the pseuCo web application interface. The browser address bar shows 'pseuco.com'. The page title is 'pseuco.com'. The main content area is divided into two panes:

- Editing:** Shows the CCS specification for 'RegularExpressionExample'. The code is:

```
CCS
1 X := ((a.1 + b.3).X) + 1
2
3 Y := ((T(a + Yb).T) + 1
4 Yb := a. Yb + 1
5 Yb := a. Yb + 1
6
7 // this is the initial process
8 //X
9 Y1
```
- LTS:** Shows the resulting Labeled Transition System (LTS) diagram. The diagram features a central state 'Y' (pink circle) with three outgoing transitions labeled 'a', 'b', and 'T'. The 'a' transition leads to a state 'X' (pink circle), the 'b' transition leads to a state '0' (blue circle), and the 'T' transition leads to another state 'Y' (pink circle). Both 'X' and the right 'Y' have self-loops labeled 'a' and 'b' respectively.

At the bottom of the interface, there are several utility buttons: 'Random path', 'Export LTS', and 'Show this file'. A status bar at the bottom left indicates a warning: 'multiple issues, first one: warning in line 1: TypeError: You are'.

<http://pseuco.com>

CCS in pseuCo: regular expressions

$(a + b)^*$

```
X := ((a.1 + b.1);X) + 1
```

```
// this is the initial process  
X
```

$(a^* + b^*)^*$

```
Y := ((Ya + Yb);Y) + 1
```

```
Ya := a. Ya + 1
```

```
Yb := b. Yb + 1
```

```
// this is the initial process  
Y
```

Demo!

CCS in pseuCo: regular expressions

$(a + b)^*$

```
X := ((a.1 + b.1);X) + 1
```

```
// this is the initial process  
X
```

$(a^* + b^*)^*$

```
Y := ((Ya + Yb);Y) + 1
```

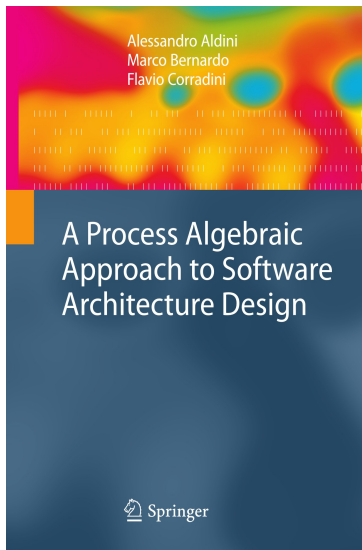
```
Ya := a. Ya + 1
```

```
Yb := b. Yb + 1
```

```
// this is the initial process  
Y
```

Demo!

Producer-Consumer Example



Producer-Consumer Example

- The **system** is composed of
 - a **producer**
 - a finite-capacity **buffer**
 - a **consumer**
- The **producer deposits** items into the **buffer** as long as the **buffer** capacity is not exceeded
- Stored items can be **withdrawn** by the **consumer** according to some predefined discipline, like FIFO or LIFO
- Assumptions:
 - The **buffer** has only **two positions**
 - Items are all identical, so that the specific discipline that has been adopted for withdrawals is not important from the point of view of an external observer

Demo!

Producer-Consumer Example

- The **system** is composed of
 - a **producer**
 - a finite-capacity **buffer**
 - a **consumer**
- The **producer deposits** items into the **buffer** as long as the **buffer** capacity is not exceeded
- Stored items can be **withdrawn** by the **consumer** according to some predefined discipline, like FIFO or LIFO
- Assumptions:
 - The **buffer** has only **two positions**
 - Items are all identical, so that the specific discipline that has been adopted for withdrawals is not important from the point of view of an external observer

Demo!