Formal Methods in software development

a.a.2017/2018

Prof.Anna Labella

CCS: Calculus of communicating processes

Main issues:

- How to specify concurrent processes in an abstract way?
- Which are the basic relations between concurrency and non-determinism?
- Which basic methods of construction (= operators) are needed?
- When do two processes behave differently?
- When do they behave the same?
- Rules of calculation:
  - Replacing equals for equals
  - Substitutivity

- R. Milner, A Calculus of Communicating Systems . LNCS 92 (1980).

CCS

Language for describing communicating transition systems

Behaviours as algebraic terms

Calculus: Centered on observational equivalence

Elegant mathematical treatment

Emphasis on process structure and modularity

Recent extensions to security and mobile systems

- CSP - Hoare: Communicating Sequential Processes (85)
- ACP - Bergstra and Klop: Algebra of Communicating Processes (85)
- CCS - Milner: Communication and Concurrency (89)
- Pi-calculus – Milner (99), Sangiorgi and Walker (01)
- SPI-calculus – Abadi and Gordon (99)
- Many recent successor for security and mobility

CCS - Combinators

The idea: 7 elementary ways of producing or putting together labelled transition systems

Pure CCS:

- Turing complete – can express any Turing computable function

Value-passing CCS:

- Additional operators for value passing
- Definable
- Convenient for applications

Here only a taster

Cfr. intro2ccs

Actions

Names a,b,c,d,...

Co-names: $\overline{a},\overline{b},\overline{c},\overline{d},...$

$a = \overline{\overline{a}}$

In CCS, names and co-names synchronize

Labels l: Names $\cup$ co-names

$\alpha \in$ Actions $= \Sigma =$ Labels $\cup \{\tau\}$

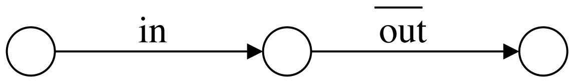
Define $\overline{\alpha}$ by:

$\overline{l} = l$, and

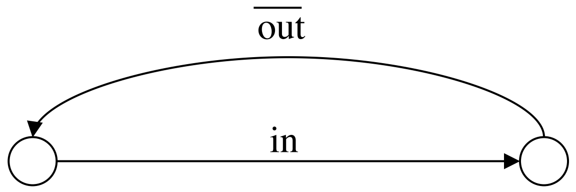
$\tau = \overline{\tau}$

CCS Combinators, II

Nil	0	No transitions
Prefix	$\alpha.P$	$\mathrm{in}.\overline{\mathrm{out}}.0 \xrightarrow{\mathrm{in}} \overline{\mathrm{out}}.0 \xrightarrow{\overline{\mathrm{out}}} 0$



Definition	$A == P$	$\mathrm{Buffer} == \mathrm{in}.\overline{\mathrm{out}}.\mathrm{Buffer}$
		$\mathrm{Buffer} \xrightarrow{\mathrm{in}} \overline{\mathrm{out}}.\mathrm{Buffer} \xrightarrow{\overline{\mathrm{out}}} \mathrm{Buffer}$



CCS Combinators, Choice

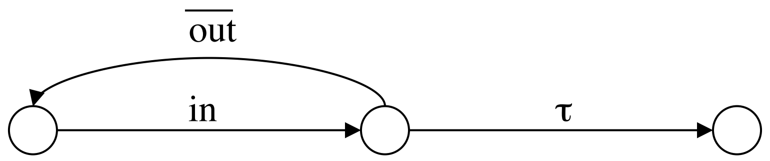
Choice $P + Q$

BadBuf == in.(τ.0 + $\overline{\text{out}}$.BadBuf)

BadBuf $\xrightarrow{in}$ τ.0 + $\overline{\text{out}}$.BadBuf

$\xrightarrow{\tau}$ 0 **or**

$\xrightarrow{\overline{out}}$ BadBuf



Obs: No priorities between τ's, $\overline{a}$'s or a's

CCS doesn't ”know” which labels represent input, and which output

May use Σ notation: $\Sigma_{i2\{1,2\}}\alpha_i.P_i = \alpha_1.P_1 + \alpha_2.P_2$

Example: Boolean Buffer

2-place Boolean Buffer

Buf^2: Empty 2-place buffer

Buf^2_0: 2-place buffer holding a 0

Buf^2_1: Do. holding a 1

Buf^2_{00}: Do. holding 00

... etc. ...

$Buf^2 == in_0.Buf^2_0 + in_1.Buf^2_1$

$Buf^2_0 == out_0.Buf^2 +$
$\quad in_0.Buf^2_{00} + in_1.Buf^2_{01}$

$Buf^2_1 == \ldots$

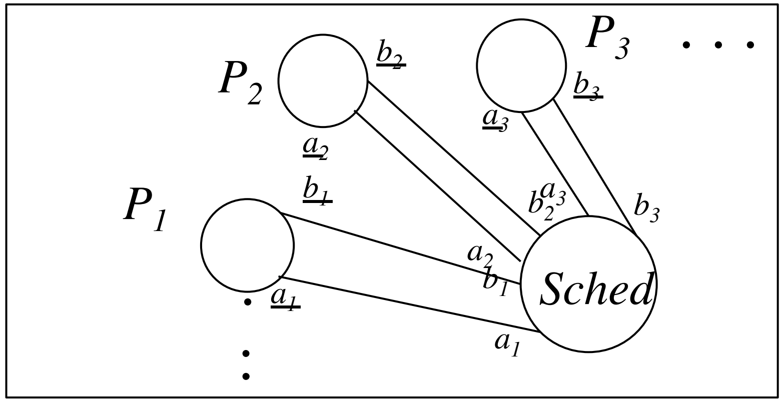
$Buf^2_{00} == out_0.Buf^2_0$

$Buf^2_{01} == out_0.Buf^2_1$

$Buf^2_{10} == \ldots$

$Buf^2_{11} == \ldots$

Example: Scheduler



a_i: start task$_i$

b_i: stop task$_i$

Requirements:

1. $a_1,\ldots,a_n$ to occur cyclically
2. a_i/b_i to occur alternately beginning with a_i
3. Any a_i/b_i to be schedulable at any time, provided 1 and 2 not violated

Let $X \subseteq \{1,\ldots,n\}$

$\mathrm{Sched}_{i,X}$:

- i to be scheduled
- X pending completion

$\mathrm{Scheduler} == \mathrm{Sched}_{1,\emptyset}$

$\mathrm{Sched}_{i,X}$

$== \Sigma_{j\in X} b_j.\mathrm{Sched}_{i,X-\{j\}}$, if $i \in X$

$== \Sigma_{j\in X} b_j.\mathrm{Sched}_{i,X-\{j\}}$

$+ a_i.\mathrm{Sched}_{i+1,X\cup\{i\}}$, if $i \notin X$

Example: Counter

Basic example of infinite-state system

$Count == Count_0$

$Count_0 == zero.Count_0 + inc.Count_1$

$Count_{i+1} == inc.Count_{i+2} + dec.Count_i$

Can do stacks and queues equally easy – try it!

CCS Combinators, Composition

Composition *P* | *Q*

$\text{Buf}_1 == \text{in.comm.Buf}_1$

$\text{Buf}_2 == \overline{\text{comm}}\text{.out.Buf}_2$

$\text{Buf}_1 \mid \text{Buf}_2$

$\xrightarrow{\text{in}} \text{comm.Buf}_1 \mid \text{Buf}_2$

$\xrightarrow{\tau} \text{Buf}_1 \mid \text{out.Buf}_2$

$\xrightarrow{\text{out}} \text{Buf}_1 \mid \text{Buf}_2$

But also, for instance:

$\text{Buf}_1 \mid \text{Buf}_2$

$\xrightarrow{\overline{\text{comm}}} \text{Buf}_1 \mid \text{out.Buf}_2$

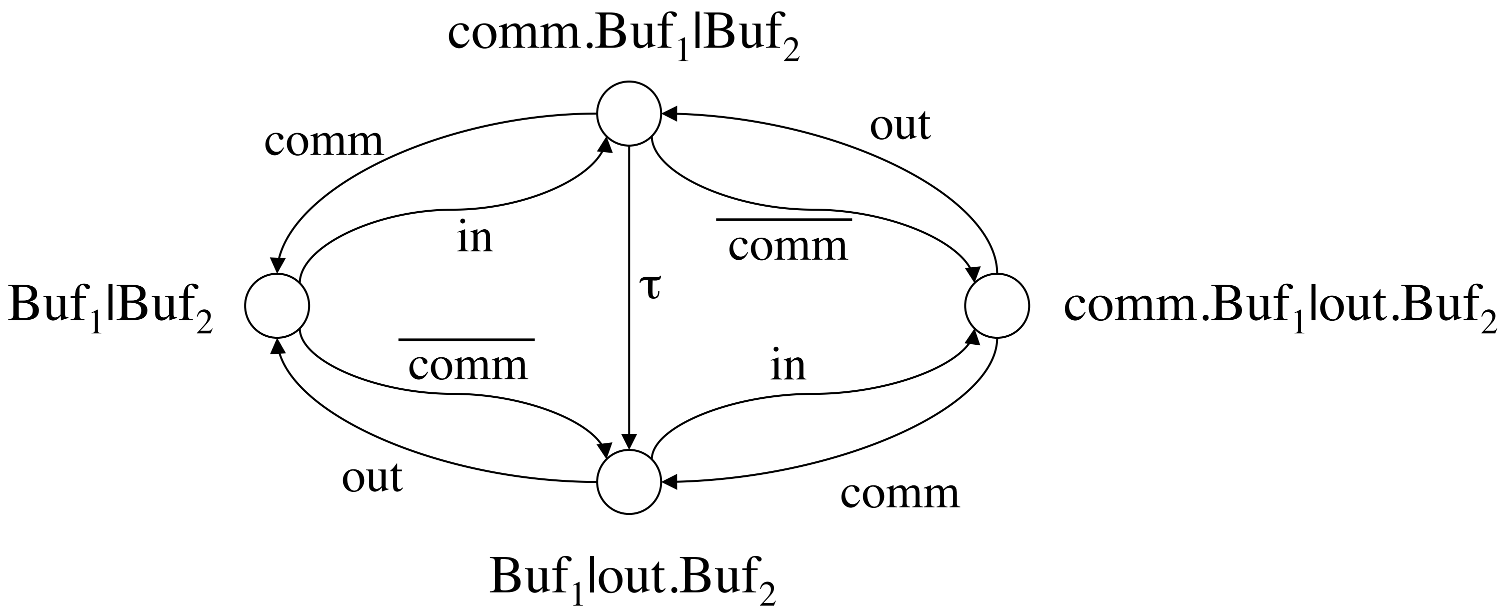
$\xrightarrow{\text{out}} \text{Buf}_1 \mid \text{Buf}_2$

Composition, Example

Buf_1 == in.comm.Buf_1

Buf_2 == $\overline{\text{comm}}$.out.Buf_2

Buf_1 | Buf_2:



CCS Combinators, Restriction

Restriction P\L

$Buf_1 == in.comm.Buf_1$

$Buf_2 == \overline{comm}.out.Buf_2$

$(Buf_1 \mid Buf_2)\backslash\{comm\}$

$\rightarrow^{in} comm.Buf_1 \mid Buf_2$

$\rightarrow^{\tau} Buf_1 \mid out.Buf_2$

$\rightarrow^{out} Buf_1 \mid Buf_2$

But *not*:

$(Buf_1 \mid Buf_2)\backslash\{comm\}$

$\rightarrow^{\overline{comm}} Buf_1 \mid out.Buf_2$

$\rightarrow^{out} Buf_1 \mid Buf_2$

CCS Combinators, Relabelling

Relabelling *P*[f] Buf == in.$\overline{\text{out}}$.Buf$_1$

Buf$_1$ == Buf[comm/out]

= in.$\overline{\text{comm}}$.Buf$_1$

Buf$_2$ == Buf[comm/in]

= comm.out.Buf$_2$

Relabelling function f must preserve complements:

$f(\bar{a}) = \overline{f(a)}$

And τ:

$f(\tau) = \tau$

Relabelling function often given by name substitution as above

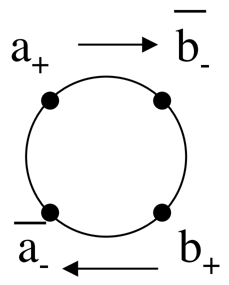
Structural congruence

Example: 2-way Buffers

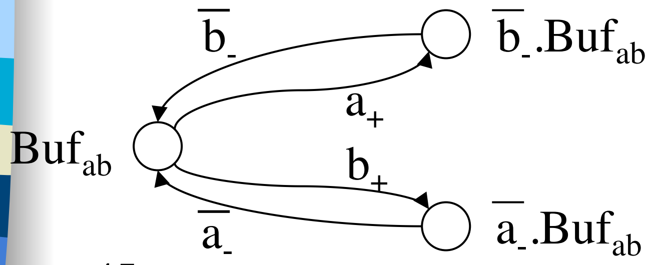
1-place 2-way buffer:

$\mathrm{Buf}_{ab} == a_+.\overline{b_-}.\mathrm{Buf}_{ab} + b_+.\overline{a_-}.\mathrm{Buf}_{ab}$

Flow graph:



LTS:



$\overline{b_-}$
$\overline{b_-}.\mathrm{Buf}_{ab}$
a_+
Buf_{ab}
b_+
$\overline{a_-}$
$\overline{a_-}.\mathrm{Buf}_{ab}$

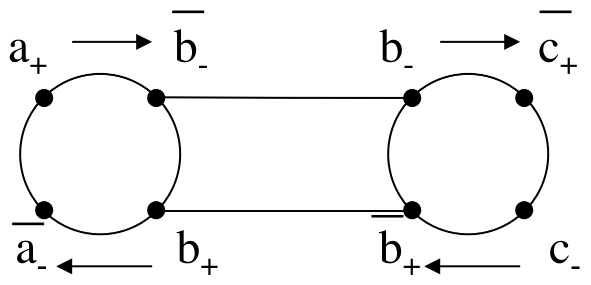
$\mathrm{Buf}_{bc} ==$

$\mathrm{Buf}_{ab}[c_+/b_+, c_-/b_-, b_-/a_+, b_+/a_-]$

(Obs: Simultaneous substitution!)

$\mathrm{Sys} = (\mathrm{Buf}_{ab} \mid \mathrm{Buf}_{bc})\backslash\{b_+, b_-\}$

Intention:



What went wrong?

Transition Semantics

To apply observational equivalence need a formalised semantics

Each CCS expression -> state in LTS derived from that expression

Compositionality: Construction of LTS follows expression syntax

Inference rules:

$$\frac{P_1 \rightarrow^{\alpha} P_2}{P_1 \mid Q \rightarrow^{\alpha} P_2 \mid Q}$$

Meaning: For all P_1, P_2, Q, α, if there is an α transition from P_1 to P_2 then there is an α transition from $P_1 \mid Q$ to $P_2 \mid Q$

CCS Transition Rules

(no rule for 0!) **Prefix** $\dfrac{-}{\alpha.P \xrightarrow{\alpha} P}$ **Def** $\dfrac{P \xrightarrow{\alpha} Q}{A \xrightarrow{\alpha} Q}$ (A == P)

Choice$_L$ $\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$ **Choice$_L$** $\dfrac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$

Com$_L$ $\dfrac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$ **Com$_R$** $\dfrac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$ **Com** $\dfrac{P \xrightarrow{l} P' \quad Q \xrightarrow{\bar{l}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$

Restr $\dfrac{P \xrightarrow{\alpha} P'}{P/L \xrightarrow{\alpha} P'/L}$ $(\alpha,\bar{\alpha} \notin L)$ **Rel** $\dfrac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$

CCS Transition Rules, II

Closure assumption: $!^{\alpha}$ is least relation closed under the set of rules

Example derivation:

$$\mathrm{Buf}_1 == \mathrm{in}.\overline{\mathrm{comm}}.\mathrm{Buf}_1$$

$$\mathrm{Buf}_2 == \mathrm{comm}.\overline{\mathrm{out}}.\mathrm{Buf}_2$$

$$(\mathrm{Buf}_1 \mid \mathrm{Buf}_2)/ \{\mathrm{comm}\}$$

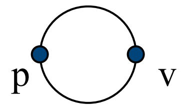
$$\rightarrow^{\mathrm{in}} \overline{\mathrm{comm}}.\mathrm{Buf}_1 \mid \mathrm{Buf}_2$$

$$\rightarrow^{\tau} \mathrm{Buf}_1 \mid \overline{\mathrm{out}}.\mathrm{Buf}_2$$

$$\rightarrow^{\mathrm{out}} \mathrm{Buf}_1 \mid \mathrm{Buf}_2$$

Example: Semaphores

Semaphore:



Unary semaphore:

$S^1 == p.S^1_1$

$S^1_1 == v.S^1$

Binary semaphore:

$S^2 == p.S^2_1$

$S^2_1 == p.S^2_2 + v.S^2$

$S^2_2 == v.S^2_1$

Result:

$$S^1 \mid S^1 \sim S^2$$

Proof: Show that

$\{(S^1 \mid S^1, S^2),$
$(S^1_1 \mid S^1, S^2_1),$
$(S^1 \mid S^1_1, S^2_1),$
$(S^1_1 \mid S^1_1, S^2_2)\}$

is a strong bisimulation relation

Example: Simple Protocol

$\text{Spec} == \text{in}.\overline{\text{out}}.\text{Spec}$

$\text{Sender} == \text{in}.\text{Transmit}$

$\text{Transmit} == \overline{\text{transmit}}.\text{WaitAck}$

$\text{WaitAck} == \text{ack}_{+}.\text{Sender} + \text{ack}_{-}.\text{Transmit}$

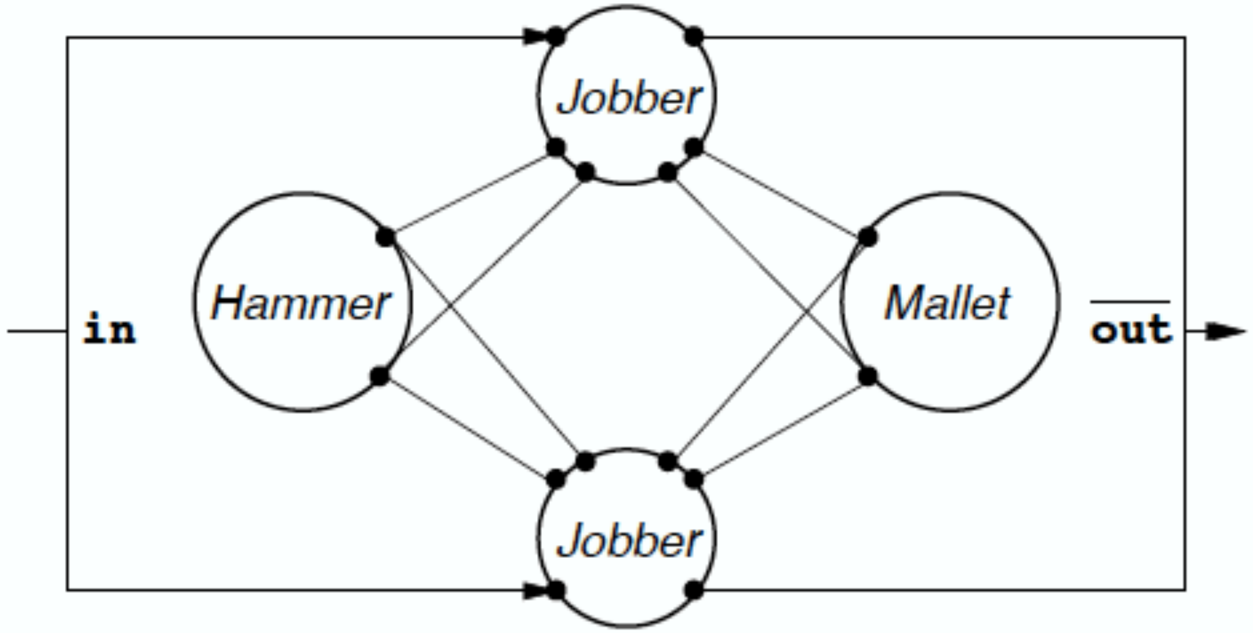
$\text{Receiver} == \text{transmit}.\text{Analyze}$

$\text{Analyze} == \tau.\overline{\text{out}}.\overline{\text{ack}_{+}}.\text{Receiver} + \tau.\overline{\text{ack}_{-}}.\text{Receiver}$

$\text{Protocol} == (\text{Sender} \mid \text{Receiver})/\{\text{transmit},\text{ack}_{+},\text{ack}_{-}\}$

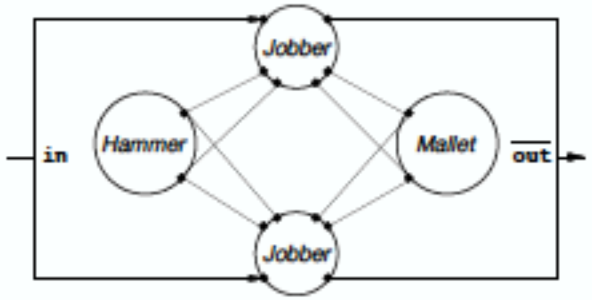
Exercise: Prove Spec $\approx$ Protocol

Example: The JobShop



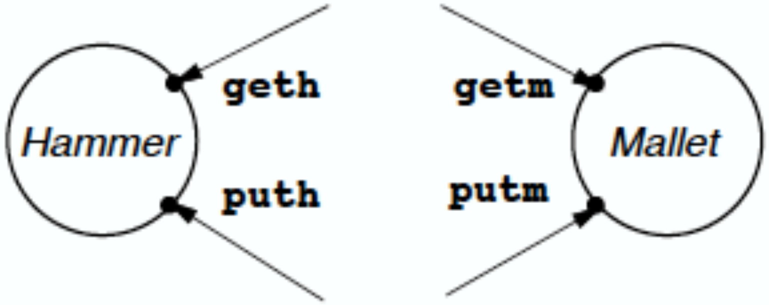
Example: The JobShop

- A simple production line:
  - Two people (the *jobbers*).
  - Two tools (hammer and mallet).
  - *Jobs* arrive sequentially on a belt to be processed.
- Ports may be linked to multiple ports.
  - Jobbers compete for use of hammer.
  - Jobbers compete for use of job.
  - Source of non-determinism.
- Ports of belt are omitted from system.
  - `in` and $\overline{\texttt{out}}$ are external.
- Internal ports are not labelled:
  - Ports by which jobbers acquire and release tools.



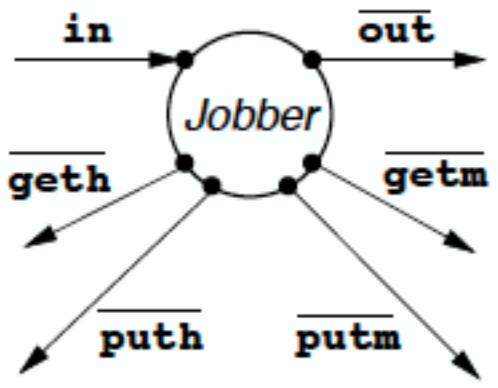
The tools of the JobShop

- Behaviors:
  - *Hammer* := geth.*Busyhammer*
    Busyhammer := puth.*Hammer*
  - *Mallet* := getm.*Busymallet*
    Busymallet := putm.*Mallet*
- *Sort* = set of labels
  - $P : L \ldots$ agent P has sort L
  - *Hammer*: $\{\mathtt{geth}, \mathtt{puth}\}$
    Mallet: $\{\mathtt{getm}, \mathtt{putm}\}$
    Jobshop: $\{\mathtt{in}, \overline{\mathtt{out}}\}$



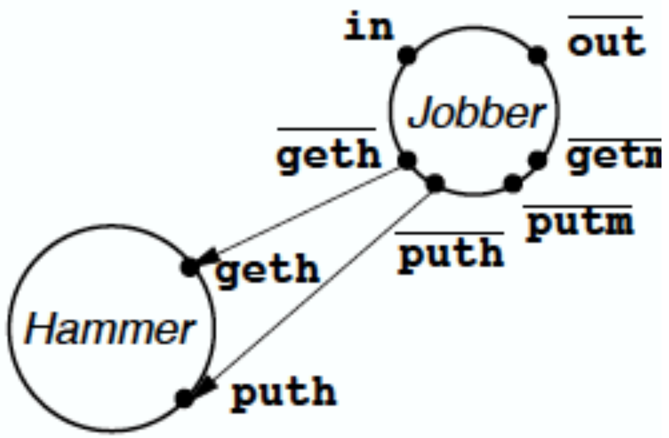
The jobbers of the JobShop

- Different kinds of jobs:
  - Easy jobs done with hands.
  - Hard jobs done with hammer.
  - Other jobs done with hammer or mallet.
- Behavior:
  - $\mathit{Jobber} := \mathtt{in}(\mathit{job}).\mathit{Start}(\mathit{job})$
  - $\mathit{Start}(\mathit{job}) :=$ **if** $\mathit{easy}(\mathit{job})$ **then** $\mathit{Finish}(\mathit{job})$
    else if $\mathit{hard}(\mathit{job})$ **then** $\mathit{Uhammer}(\mathit{job})$
    else $\mathit{Usetool}(\mathit{job})$
  - $\mathit{Usetool}(\mathit{job}) := \mathit{Uhammer}(\mathit{job}) + \mathit{Umallet}(\mathit{job})$
  - $\mathit{Uhammer}(\mathit{job}) := \overline{\mathtt{geth}}.\overline{\mathtt{puth}}.\mathit{Finish}(\mathit{job})$
  - $\mathit{Umallet}(\mathit{job}) := \overline{\mathtt{getm}}.\overline{\mathtt{putm}}.\mathit{Finish}(\mathit{job})$
  - $\mathit{Finish}(\mathit{job}) := \overline{\mathtt{out}}(\mathit{done}(\mathit{job})).\mathit{Jobber}$

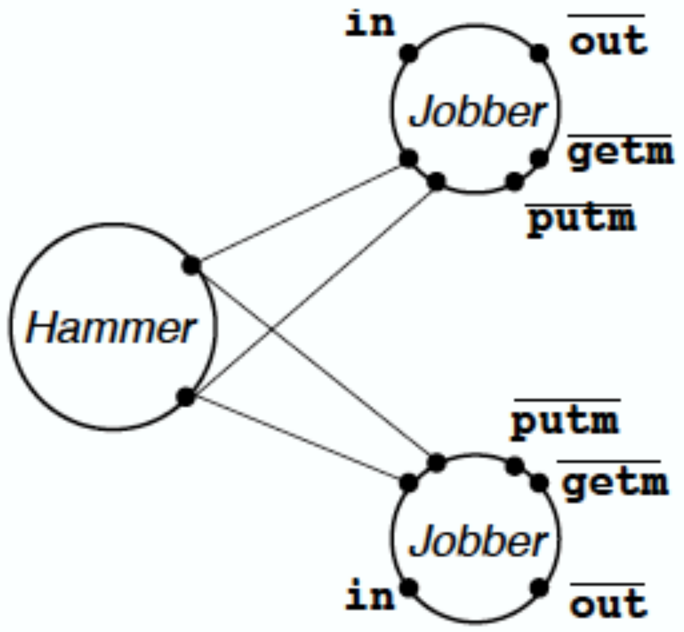


Composition of the agents

- *Jobber-Hammer* subsystem
  - *Jobber* | *Hammer*
  - *Composition* operator |
  - Agents may procced independently or interact through *complementary* ports.
  - Join complementary ports.
- Two jobbers sharing hammer:
  - *Jobber* | *Hammer* | *Jobber*
  - Composition is commutative and associative.



Further composition



in
$\overline{\texttt{out}}$
Jobber
$\overline{\texttt{getm}}$
$\overline{\texttt{putm}}$
Hammer
$\overline{\texttt{putm}}$
$\overline{\texttt{getm}}$
Jobber
in
$\overline{\texttt{out}}$

- *Internalisation* of ports:
  - No further agents may be connected to ports:
  - *Restriction* operator \
  - $\backslash L$ internalizes all ports L.
  - (*Jobber* | *Jobber* | *Hammer*)$\backslash\{\texttt{geth},\texttt{puth}\}$
- Complete system:
  - *Jobshop* := (*Jobber* | *Jobber* | *Hammer* | *Mallet*)$\backslash L$
  - $L := \{\texttt{geth},\texttt{puth},\texttt{getm},\texttt{putm}\}$

Example: Jobshop

i_E: input of easy job
i_N: input of neutral job
i_D: inpu$\underline{\text{t}}$ of difficult job
O: output of finished product

$A == i_E.A' + i_N.A' + i_D.A'$
$A' == o.A$

Spec = A | A

Hammer: $H == gh.ph.H$
Mallet: $M == gm.pm.M$
Jobber:
$J == \Sigma_{x\in\{E,N,D\}} i_x.J_x$
$J_E == o.J$
$J_N == \overline{gh}.\overline{ph}.J_E + \overline{gm}.\overline{pm}.J_E$
$J_D == \overline{gh}.\overline{ph}.J_E$
Jobshop ==
$(J \mid J \mid H \mid M)/\{gh,ph,gm,pm\}$

Theorem:
Spec $\approx$ Jobshop

Exercise: Prove this.