Concurrency and Reaction

- Processes may run concurrently;
  add concurrency operator $P|Q$
- concurrent processes may react;
  based on comparison of labels
- Reactions are bilateral;
- Reactions are directional:
  complementary labels, sender and receiver
- Global and local names
- Internal reactions are unobservable

Describe local processes
Combine them with |
Analyse externally observable behavior

Reaction and Concurrency: Example

Two jobs (start, do ,end)

- execute concurrently
- may need a tool for exclusive use (gettool, puttool)

```
Job1  = s1.gt.do1.pt.e1.Job1
Job2  = s2.gt.do2.pt.e2.Job2
Tool = 'gt.'pt.Tool

Cjobs = new gt,pt (Job1 | Job2 | Tool)
```

- `Cjobs` consists of concurrent processes.
- reaction: between name and co-name in concurrent processes:
  is unobservable action τ
- `gt`, `‘gt`, `pt`, `‘pt` are local for `Cjobs`, can only react within `Cjobs`

Example trace:
s1, τ, s2, do1, τ, e1, τ,...

Renaming?

```
Job(s,do,e,gt,pt) = ...
```

```
new 'g,'p(Job(s1,do1,e1,g,p)|...|...))
```

CWB ∼ Old CCS notation:

- Explicit renaming:
  `Job = s.do.e.Job + s.gt.do.pt.e.Job`
  `(Job[s1/s,do1/do,e1/e]|Job[...]|...)`
- *Hiding* instead of `new`:

  `(Job|Job|Tool)\ {gt,pt}`

CCS - an overview

- Process expressions (as in the example)
  - complementary actions,
    unobservable action τ,
    process names
  - prefix operator ".": `a.P`
  - concurrency operator "|": `P1|P2`
  - local action names / hiding
  - renaming
  - process definitions
- Define the corresponding LTS:
  - states are process expressions
  - transitions:
    reactions (handshaking)
    non-local actions
- **Equivalences** of process expressions

Concurrent Process Expressions

- Actions α:
  names/conames (observable λ) `a,'a,...`
  internal (unobservable) τ (`t` in CWB)
- (Parameterized) process names `A(a,b)`
- concurrent process expression:

  $$P ::= A(\ldots) \mid \sum_i \alpha_i.P_i \mid P_1|P_2 \mid \mathit{new}\ aP$$

  Index set of sum is finite
  `0` is a sum with empty index set
- Every process name has a *definition*

  $$A(\vec{a}) = P_A$$

  P_A is a sum

Notational conventions:

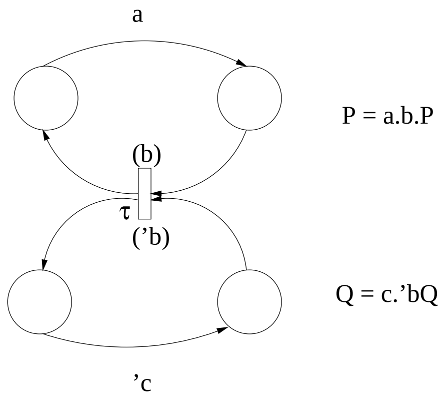
- implicit parameters
- "." and *new* bind stronger than $+$ and $|$

CWB Example

```
proc Job =  s.gt.do.pt.e.Job

proc Tool = 'gt.'pt.Tool

proc Job1 = Job[s1/s,do1/do,e1/e]
proc Job2 = Job[s2/s,do2/do,e2/e]

proc Cjob =
          (Job1 | Job2 | Tool)\{gt,pt}
```

Semantics of concurrency (rough idea)

- Extending semantics of sequential process
  :
  Transition semantics on process expressions
  Results in an LTS
- Consider *new* $b(P|Q)$ for sequential P,Q
  - states: product of the states of P, Q
  - transitions:
    individual in P or Q for global actions
    joint in P and Q on b, ‘b, unobservable.



Structure of the Discussion

- The notion of process congruence.
- Structural congruence: adjusting syntax to intuitions.
- Transition semantics for internal reactions
- Transition semantics for observable actions and τ
- Strong bisimulation and strong equivalence
- The expansion law
- Strong equivalence is a process congruence.

Process congruence

- **Intuition:** identify some process expressions as having "the same meaning"
- **Congruence:** the equivalence should be preserved by some *contexts*.

An equivalence relation $\cong$ is a **process congruence** if, whenever $P \cong Q$ then

$$\begin{aligned}
\alpha.P + M &\cong \alpha.Q + M \\
new\ a\ P &\cong new\ a\ Q \\
P|R &\cong Q|R \\
R|P &\cong R|Q
\end{aligned}$$

- Can be defined by equations (see $\equiv$ below)

Notice: prefix condition in the sum.

The structural congruence $\equiv$

Intuition: $\equiv$ describes irrelevant syntactic aspects of process expressions

$\equiv$ is a relation on process expressions satisfying the equational axioms:

- Change of bound names
  $new\ aP(a) \equiv new\ bP(b)$
- reordering in summation
  $P + Q \equiv Q + P$
- Concurrency operator: $P|0 \equiv P$, $P|Q \equiv Q|P$, $P|(Q|R) \equiv (P|Q)|R$
- Hiding:
  $new\ a\ (P|Q) \equiv P|new\ a\ Q$ if $a \notin fn(P)$
  $new\ a\ 0 \equiv 0$, $new\ ab\ P \equiv new\ ba\ P$
- Unfolding of sequential process expressions
  $A(b) \equiv \{b/a\}P_A$ if $A(a) = P_A$

and such that: if $P \equiv Q$ then $\mathcal{C}[P] \equiv \mathcal{C}[Q]$ for any context $\mathcal{C}$ of previous slide.

Example: linking

Given: $P(l,'r), Q(l,'r)$

$$P \frown Q = new\ m(\{m/r\}P|\{m/l\}Q)$$

$$P \frown (Q \frown R) \equiv (P \frown Q) \frown R$$

Check this!

Standard form

Hiding done on top level of concurrency

For any P
$P \equiv new\ \vec{a}(M_1|...|M_i)$
where each M_i is a non-empty sum.

$$new\ a((a.Q_1 + b.Q_2)|'a.0)|('b.R_1 +' a.R_2)$$
$$\equiv$$
$$new\ c((c.Q_1 + b.Q_2)|'c.0|('b.R_1 +' a.R_2))$$

Internal reaction

$$TAU : \tau.P + M \rightarrow P$$

$$REACT : (a.P + M)|('a.Q + N) \rightarrow P|Q$$

$$PAR : \frac{P \rightarrow P'}{P|Q \rightarrow P'|Q}$$

$$RES : \frac{P \rightarrow P'}{new\ a\ P \rightarrow new\ a\ P'}$$

$$STRUCT : \frac{P \rightarrow P'}{Q \rightarrow Q'}\ if\ P \equiv Q\ and\ P' \equiv Q'$$

Internal reaction

$new\ a\ ((a.Q1 + b.Q2)|'a)|('b.R1 +' a.R2) \rightarrow$
$new\ a\ Q1|('b.R1 +' a.R2)$
Prove this using the rules!

- $\rightarrow$ describes reaction **within** a given expression
- reaction is captured by transformation of the expressions
- the actions are unobservable by an external observer

Transition Rules

$$SUM_t \ : \ M + \alpha.P + N \rightarrow^{\alpha} P$$

$$REACT_t \ : \ \frac{P \rightarrow^{\lambda} P' \quad Q \rightarrow^{'\lambda} Q'}{P|Q \rightarrow^{\tau} P'|Q'}$$

$$LPAR_t \ : \ \frac{P \rightarrow^{\alpha} P'}{P|Q \rightarrow^{\alpha} P'|Q}$$

$$RPAR_t \ : \ \frac{Q \rightarrow^{\alpha} Q'}{P|Q \rightarrow^{\alpha} P|Q'}$$

$$RES_t \ : \ \frac{P \rightarrow^{\alpha} P'}{new\ a\ P \rightarrow^{\alpha} new\ a\ P'} if\ \alpha \notin \{a,' a\}$$

$$IDENT_t \ : \ \frac{\{b/a\}P_A \rightarrow^{\alpha} P'}{A(b) \rightarrow^{\alpha} P'}\ if\ A(a) = P_A$$

Transition Rules at work

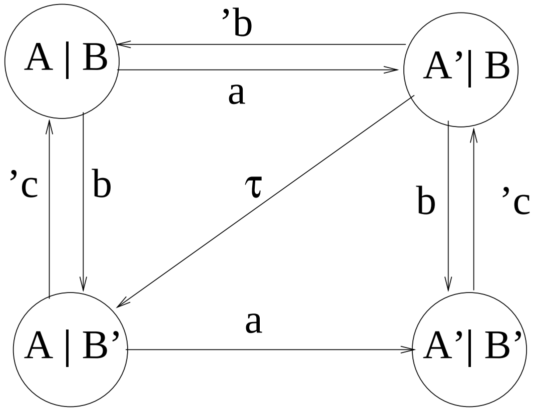
$A = a.A' \quad A' = \ 'b.A$
$B = b.B' \quad B' = \ 'c.B$

The only transitions of $A'|B$:
$A'|B \rightarrow^{'b} A|B$
$A'|B \rightarrow^{b} A'|B'$
$A'|B \rightarrow^{\tau} A|B'$

The only transition of $new\ b\ (A'|B)$:
$new\ b\ (A'|B) \rightarrow^{\tau} new\ b\ (A|B')$

LTS of a process

- **States:** process expressions
- **Transitions:** defined by the transition rules
- A given expression P has a finite number of derivatives, hence LTS is finite



A = a.A' A' ='b.A

B = b.B' B' = 'c.B

Reaction agrees with τ-transition

$P \rightarrow^{\tau} \equiv P'$ iff $P \rightarrow P'$

Transitions do not capture $\equiv$, but:

- Structural congruence respects transition

  If $P \equiv Q$ and $P \rightarrow^{\alpha} P'$ then there exists Q' s.t. $Q \rightarrow^{\alpha} Q'$ and $Q' \equiv P'$

  Induction on the depth of inference of $P \rightarrow^{\alpha} P'$

- Thus, structural congruence is a strong bisimulation over concurrent processes

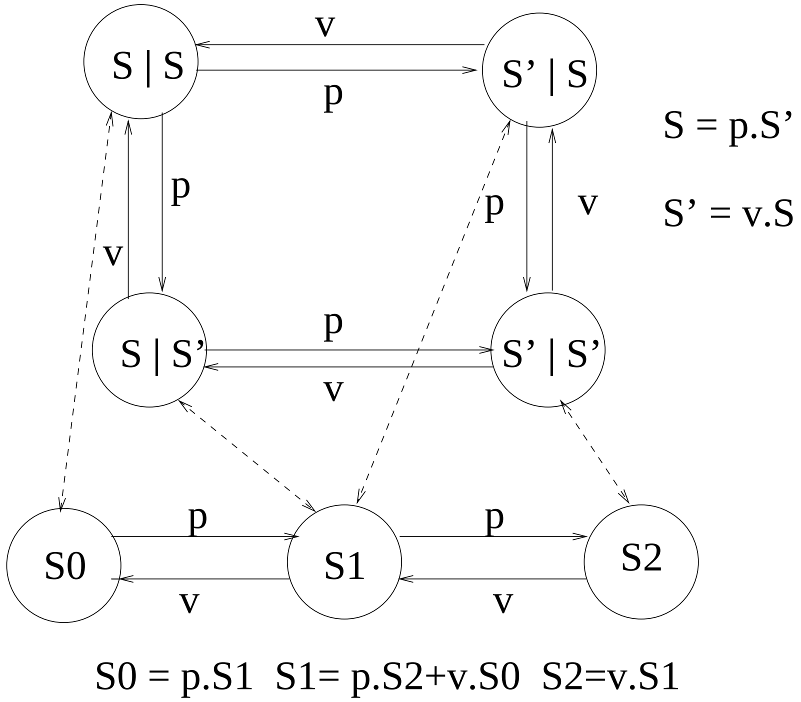
Strong Equivalence

- $P \sim Q$ iff P, Q strongly bisimilar
  the same moves possible in P and Q if the history was the same.
- $\sim$ is a process congruence
  s. equivalent components can be exchanged within process expressions

Applied as follows:

- A simple process expression E describes the expected behavior of a system (specification)
- The components of a concurrent system (implementation) are described by a concurrent process expression C.
- Verification: check $E \sim C$

Verification

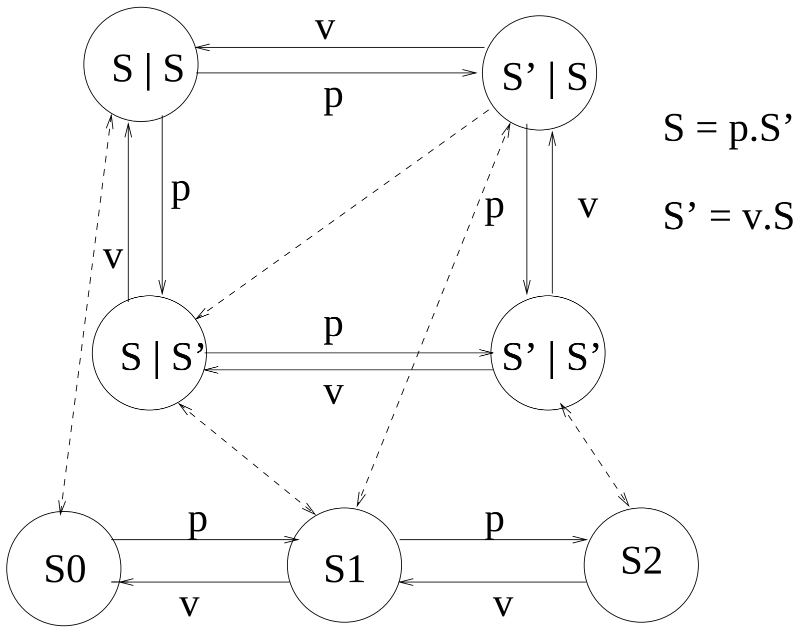


Reducing States

- S is **strong simulation up to** $\equiv$ iff whenever $P\ S\ Q$:
  if $P \rightarrow^{\alpha} P'$ then there exists Q' such that $Q \rightarrow^{\alpha} Q'$ and $P'(\equiv S \equiv)Q'$
- If S is a strong bisimulation up to $\equiv$ and $P\ S\ Q$ then $P \sim Q$. (i.e. S is a strong equivalence)

Thus: ignore irrelevant syntactic differences

Reducing states



S0 = p.S1 S1= p.S2+v.S0 S2=v.S1

We can collapse structurally equivalent states

The Expansion Law

- **Standard Form**
  `new` *if not inside sumation can be brought to the top.*

  $P \equiv new\ \vec{a}(M_1|...|M_n)$;
  M_i's - nonempty sums

  Example: $P|new\ aQ \equiv new\ a(P|Q)$

- **The Expansion Law**
  concurrency reduced to nondeterminism

  $new\ \vec{a}(P_1|...|P_n)\ \sim$
  $\Sigma(\alpha.new\ \vec{a}(P_1|...|P_i'|...|P_n))+$
  $+\Sigma(\tau.new\ \vec{a}(P_1|...|P_i'|...|P_j'|...|P_n))$
  for appropriate transitions.

  Example:
  $new\ a(b.B|a.A|'a.C) \sim b.new\ a(B|a.A|'a.C)+$
  $\tau.new\ a(b.B|A|C)$