



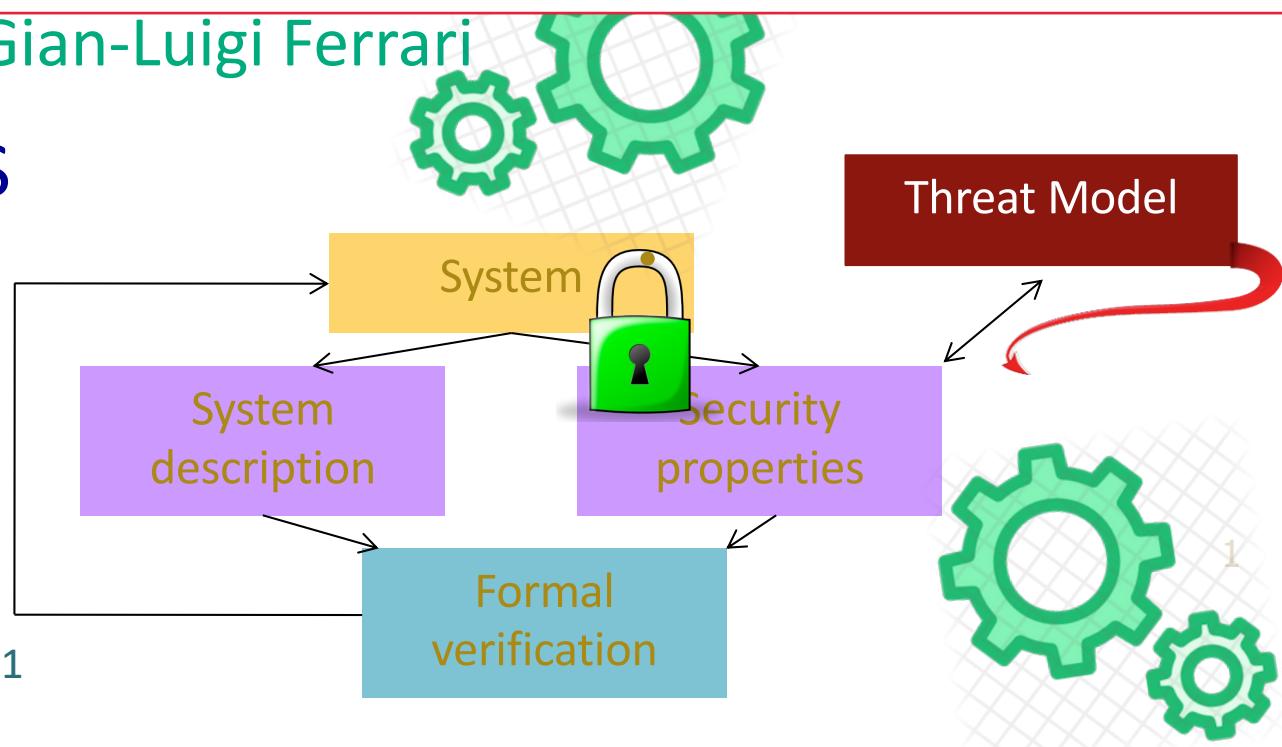
LANGUAGE-BASED TECHNOLOGY FOR SECURITY

il motivo dominante e l'anima del corso, sino a questo punto è stata la parte relativa all'uso di strumenti di natura statica per andare a considerare proprietà di sicurezza dei linguaggi di programmazione. Abbiamo visto varie forme di sistemi di tipo ultime il meccanismo dei sistemi di tipo basati su rust, relativamente al discorso dell' ownership e dal life time che sono nozioni controllate staticamente dal compilatore di rust adesso vedremo un'altro aspetto del analisi statica, dei linguaggi di programmazione che viene utilizzato, nel backend dei compilatori per definire delle proprietà, ottimizzare del codice e in modo particolare vedremo le nozioni di static analysis che vengono usate per controllare staticamente delle proprietà di sicurezza. Noi abbiamo visto due esempi di analisi statica e che vengono utilizzate nella pratica, in modo particolare la control flow integrity e la taint analisi e qui in questa parte del corso la professoressa bodei affronterà invece alcuni aspetti più vicini alle attività di ricerca che viene effettuata nel contesto del language, based security con tecniche di control flow analysis. Adesso vediamo un'altra parte più legata alla ricerca e che ha a che fare con appunto l'applicazione della control flow analysis a algebre di processo naturalmente applicate all' ambito della sicurezza, alla modellazione di sistemi per protocolli crittografici oppure per sistemi di tipo IoT. Vedremo una parte di premessa a questa parte del corso che serve per capire come applicare poi la la control flow analysis, quindi siccome l' applicheremo ad algebre di processo, faremo un escursus sulle algebre di processo che ci servono in questo caso quindi vedremo in particolare una delle più note e nel nell'ambito dei linguaggi formali che è il PI calculus.

Chiara Bodei, Gian-Luigi Ferrari

PI CALCULUS

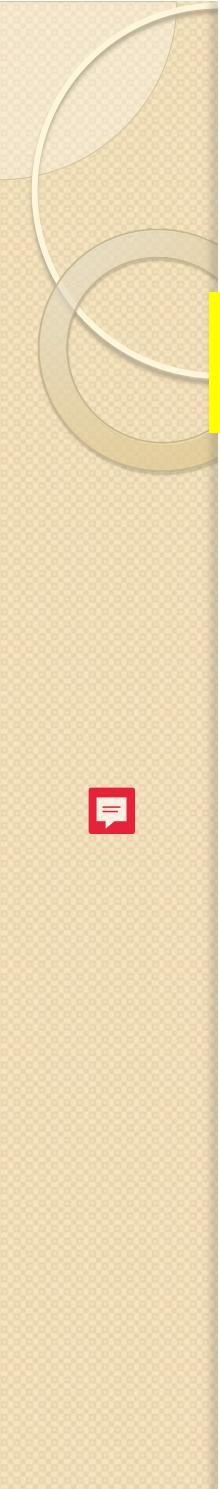
Maggio 2021





Outline

- **Motivation**
- Key notions
- CCS
- From CCS to Pi calculus
- Pi calculus: syntax and semantics
- Conclusions and what next



Motivation

π -calculus : concurrent languages = lambda calculus : sequential languages
Process calculi treat processes much like λ -calculus treats computable functions

- **λ -calculus*** is the core language of functional computation, in which “everything is a function” and all computation proceeds by function application
- **π -calculus (1989)** is the core formal calculus of message-based concurrency, in which “everything is a process” and all computation proceeds by communication on channel
 - interaction

* λ -calculus can be encoded in π -calculus



Motivation

System behaviour tends to be composed of several processes that are executed concurrently, where these processes exchange data in order to influence each other's behaviour.

- Non determinism
- Parallelism
- Interaction
- Infinite runs

quindi il comportamento di un sistema viene visto attraverso la composizione di più processi che vengono eseguiti correntemente parallelo, se volete e questi processi scambiano dati in modo da influenzare il comportamento reciproco ora se vogliamo essere più precisi, possiamo far presente che c'è una differenza tra parallelismo e concorrenza, perché parallelismo è l' effettivo andare in parallelo, quindi pensandolo dal punto di vista architettonale ci sono più elaboratori che lavorano nello stesso momento, la concorrenza può essere pensata come il parallelismo in potenza, quindi volendo si può pensare che ci siano dei momenti in cui si alternano sullo stesso calcolatore naturalmente le altre motivazioni per richiedere un tale modello sono quella di rappresentare anche altre nozioni che sono quelle del non determinismo e anche quello delle run infinite, cioè dell'esecuzione infinito, ovvero si può pensare a rappresentare dei sistemi che continuano a mantenere nel tempo lo stesso comportamento, ma questo diciamo da un punto di vista formale, l'avete già visto perché le strutture ricorsive le avete già viste anche in altri ambiti.

Milner's insight was that concurrent processes have an algebraic structure

I'algebra di processi che vediamo oggi, cioè il PI calcolo, è stato introdotto da Milner, il quale ha avuto una intuizione che è quella che i processi concorrenti possono essere rappresentati in maniera tale che si possa capirne la struttura algebrica di fondo



Outline

- Motivation
- **Key notions**
- CCS
- From CCS to Pi calculus
- Pi calculus: syntax and semantics
- Conclusions and what next



Key notion

quindi vedremo bene in particolare il pi calcolo che poi arriva ad essere l'evoluzione di precedenti calcoli proprio introdotti anche da milner, ha una nozione in più che è quella della mobilità che ci rappresenta in particolare un certo tipo di mobilità, perché sulla mobilità possiamo pensare alla mobilità dei processi, cioè i processi che si muovono all'interno di uno spazio fisico oppure all'interno di uno spazio virtuale di processi che sono collegati in realtà la mobilità come intesa qua va presa per la mobilità dei link che è un concetto che vedremo meglio nel prosieguo della lezione quindi, in sostanza, il pi calcolo modellerà la modifica della connettività dei sistemi interattivi attraverso il passaggio di link prima ancora di vedere il pi calcolo, vediamo la forma da cui è partito, cioè il linguaggio che sempre introdotto da miller, che è il CCS da cui poi il pi calcolo ha preso la sua natura.

Mobility is the key notion of Pi calculus. It can refer to:

1. **processes move** in the physical space of computing sites;
 2. **processes move** in the virtual space of linked processes;
 3. **links move** in the virtual space of linked processes.
-
- The Pi calculus models the **changing** connectivity of interactive systems (point 3) [**link passing** mobility]
 - The Pi calculus evolved from CCS

Concurrent processes have an algebraic structure



Key notion

Mobility is the key notion of Pi calculus. It can refer to:

1. **processes move** in the physical space of computing sites;
2. **processes move** in the virtual space of linked processes;
3. **links move** in the virtual space of linked processes.

- The Pi calculus models the **changing** connectivity of interactive systems (point 3) [**link passing mobility**]
- The Pi calculus evolved from CCS

Concurrent processes have an algebraic structure



Process calculi

Milner's general model:

- A concurrent system is a collection of processes
- A process is an independent agent that may perform internal activities in isolation or may interact with the environment to perform shared activities

The insight was that concurrent processes have an algebraic structure

As a consequence, process calculi are also called process algebras

quindi quello che abbiamo è un sistema concorrente che viene visto come una collezione di processi in cui ciascun processo ha una natura sia indipendente, perché può eseguire delle attività interne completamente isolate dal resto del contesto, oppure può interagire con il contesto all'esecuzione di attività condivise e proprio perché c'è questa intuizione che i processi concorrenti abbiano una struttura algebrica, un'altro modo per dire process calculi è anche process algebra, cioè algebra di processo, proprio perché si vanno a riconoscere delle leggi algebriche che governano la sintassi e anche la semantica dei nostri processi.

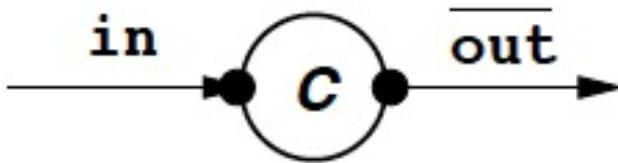


Outline

- Motivation
- Key notion
- **CCS**
- From CCS to Pi calculus
- Pi calculus: syntax and semantics
- Conclusions and what next

Calculus of Communicating Systems (CCS)

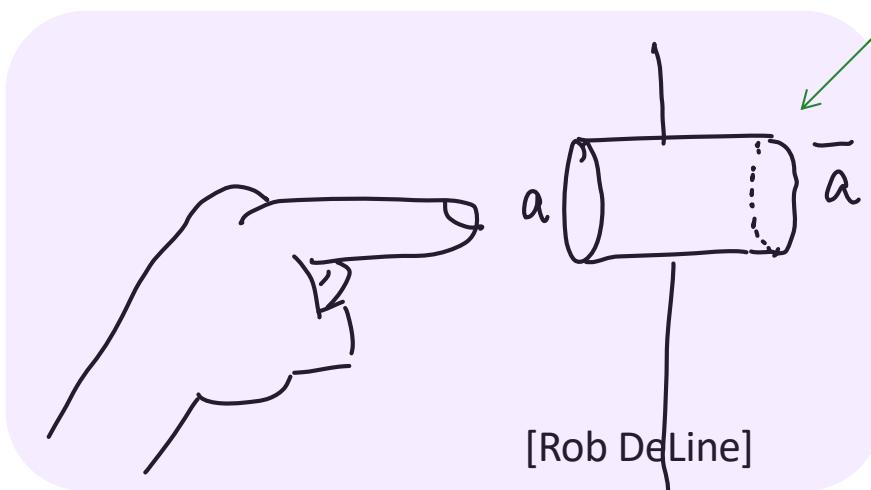
- A **CCS process** is a computing agent that may communicate with its environment via its interface [it is an **abstraction** of an independent **thread** of control]
- **Interface** = Collection of communication ports/channels, together with an indication of whether they are used for input or output.



partiamo quindi dall' antenato del pi calcolo che il ccs sempre di milner che sta per: calcolo di sistemi comunicanti, quindi è un agente di computazione che può comunicare con il suo ambiente attraverso un qualcosa che può essere visualizzato come un'interfaccia e se volete, il processo è un' astrazione di un thread indipendente dal punto di vista del controllo. Per quanto riguarda l'interfaccia, la si può visualizzare come una raccolta di porte o canali di comunicazione con un' indicazione sull'uso, cioè l'uso può essere distinto in uso in input e in output, quindi i canali possono essere utilizzati diciamo nelle due direzioni.

CCS: actions and co-actions

- Actions a and \bar{a} are viewed as being **complementary**



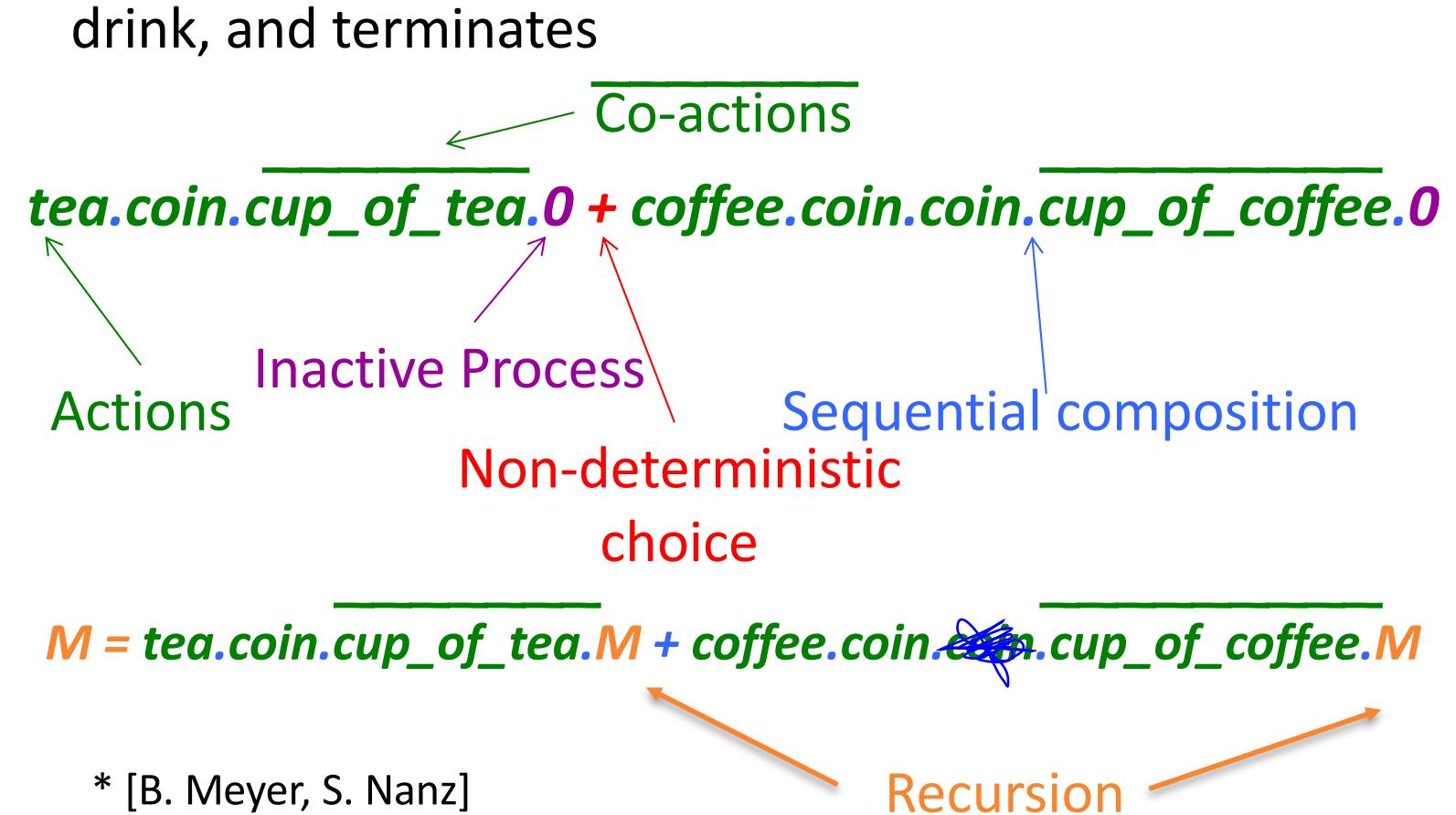
Pushing the button is one action seen from two perspectives

Coffee machine

possiamo vedere le azioni che si possono fare di sincronizzazione come sostanzialmente azioni che sono elementari input/output oppure se volete pensare a una coffee machine, a un distributore da una parte c'è chi preme dall'altra chi accoglie la richiesta quindi è come vedete un'azione di sincronizzazione dai due lati, quindi possiamo pensare a un canale come un'astrazione del link di comunicazione tra i due processi, questa è proprio in maniera super intuitiva e superficiale, vediamo meglio

A **channel** is an abstraction of the communication link between two processes

e prendiamo proprio l'esempio della coffee machine e quindi supponiamo di voler modellare un distributore di caffè e te che prende un ordine che può essere sia per il tè che per il caffè, accetta il pagamento adeguato e poi versa la bevanda richiesta e quindi termina. Ecco questo in ccs può essere rappresentato dalla seguente stringa in cui adesso facciamo una sorta di reverse engineering per capire com'è fatta la sintassi per poi arrivare alla formale definizione della sintassi, quindi abbiamo se vedete due branch, due rami legati a un più. Il più rappresenta una scelta non deterministica quindi ci aspettiamo in un distributore chi può vendere caffè o tè di avere possibili scelto o scelgo il tè, o scelgo il caffè. A questo punto esaminiamo uno dei due rami, l'altro è del tutto simmetrico, quindi supponiamo di volere il tè, quindi c'è un'azione che è predisposta per accettare la richiesta di te a questo vediamo che il punto in azzurro rappresenta la composizione sequenziale, quindi il fatto che si parli di agenti di processi concorrenti non vuol dire che non si possano avere delle sequenze di comandi, quindi in questo caso ci sono dei rami sequenziali in cui ci si aspetta una richiesta per il tè a questo punto si aspetta la somma di monete adeguata per comprare il tè e quindi questo lo potete visualizzare come delle azioni di input a questo punto c'è un'azione di output perché a questo punto la coffee machine eroga il tè e questo è il significato di questa azione che viene chiamata in gergo coazione ed ha sopra il proprio nome questa sottolineatura che sta ad indicare che questo è un output per distinguerlo dalla corrispondente azione priva di questa sopra lineatura che sta per il corrispondente input. A questo punto quindi diciamo che lo scambio tra l'utente e chi vuole il tè dovrebbe essere finito tant'è vero che l'ultima azione è quella in viola, che sta per il processo inattivo, cioè lo skip non fa niente, è finita. In maniera del tutto analoga possiamo vedere a destra nel ramo di destra la stessa sequenza analoga, diciamo per il caffè, quindi io mi aspetto la richiesta del caffè, le monete giuste per comprarlo e quindi abbiamo l'erogazione della tazza di caffè. Quindi queste abbiamo già visto parte della sintassi, l'altra cosa che era stata fuori che ho messo nella riga in fondo invece è la ricorsione perché ci si può aspettare che un distributore di caffè e tè agisca non una volta sola ma ripetutamente, per cui il modo in questo calcolo uno dei modi in questo calcolo per vedere la ricorsione è definire il distributore come qualcosa che fa queste azioni e poi continua, ricominciando daccapo sostanzialmente quindi, anche questo è un'altro modo per vedere la sintassi.





Calculus of Communicating Systems (CCS)

Basic Entities

- A set $N = a, b, \dots$ of names and $\bar{N} = \{\bar{a} \mid a \text{ in } N\}$ of co-names.
- A set $L = N \cup \bar{N}$ of labels (ranged over by l, l', \dots).
- A set $Act = L \cup \{\tau\}$ of actions (ranged over by a, b, \dots).
- Action τ is called the silent or unobservable action.

vediamo invece più formalmente, quindi per gestire la complessità di questi sistemi servono dei nomi che rappresentano i canali di comunicazione, le interfacce, le porte come volete pensarle, quindi abbiamo dei nomi e proprio per mantenere questa corrispondenza tra l' input e l'output abbiamo quelli che in gergo vengono chiamati i conames, che sono gli stessi nomi di prima, però con la sopra lineatura quindi abbiamo un'insieme di etichetta che è fatto dall'unione dei nomi e dei conomi e abbiamo un'insieme di azioni che sono composte dalle etichette, quindi sono le azioni che si possono fare sui canali unita ad un'altra azione particolare che viene chiamata tau o azione silente o azione non osservabile. Questa è un'azione particolare che vedremo meglio nel seguito e che rappresenta o banalmente attività interna all'interno di unprocesso oppure rappresenta qualcosa che dall'esterno non si può osservare quindi uno dall'esterno vedremo che può vedere che c'è un'attività congiunta condivisa tra due processi, ma non ne può vedere la natura.

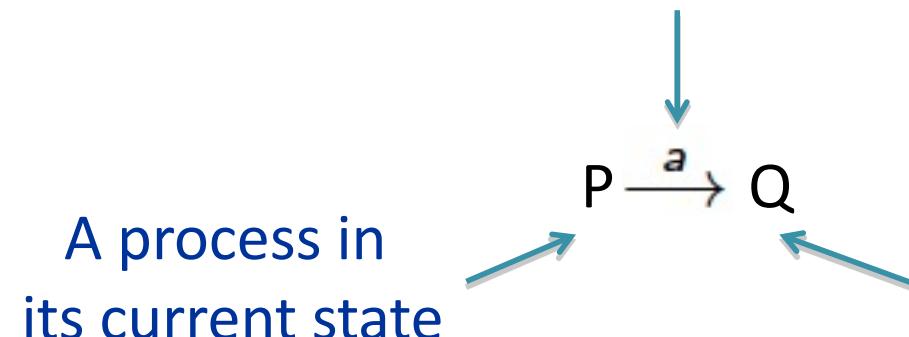
Abbiamo bisogno di capire come descrivere il comportamento di questi processi e questo lo faremo attraverso l'uso di un transition system, cioè sistemi di transizione etichettati e questo ci permetterà di utilizzare una semantica operazionale di tipo strutturato. Vedremo un processo che si evolve nel tempo e lo vedremo a questa evoluzione attraverso l'uso di transazioni che sono frecce sostanzialmente quindi mi mettono in relazione il punto di partenza con il punto d'arrivo, frecce etichettate, quindi vedremo che, per esempio, il processo p evolverà nel processo q attraverso l'azione a e quindi vedremo proprio l'interazione attraverso un grafo che mi rappresenta tutte le possibili evoluzioni del comportamento.

Calculus of Communicating Systems (CCS)

To formally describe the behaviour of CCS processes, we use **Labelled Transition Systems (LTS)** and resort to **Structural Operational Semantics (SOS)**

Transitions

Ongoing interaction
with the environment
(with other processes)



- The transition describes one possible next step of the execution of P

La sintassi del CCS prevede la presenza di un frammento sequenziale che ora qui non vi ho messo la grammatica l'ho messa di lato ma vi ho messo di qua a sinistra la spiegazione quindi sostanzialmente abbiamo un processo vuoto che è il processo che dicevamo prima che non fa niente poi abbiamo un operatore di prefisso che è il punto che mi dice prima faccio l'azione a e poi prosegue come p, poi ho delle definizioni ricorsive come abbiamo visto e poi ho la scelta non deterministica e la scelta non deterministica la possiamo rappresentare in più modi qui li ho messi tutti e due, cioè possiamo pensare a una sommatoria lunga a piacere di processi oppure possiamo avere la sommatoria a due, che è la cosa più usata di solito che sostanzialmente è un caso particolare della sommatoria generale. Poi abbiamo naturalmente l'operatore di composizione parallela che è la barra verticale e che ci dà la possibilità di far vedere proprio il comportamento concorrente parallelo di due componenti. Poi abbiamo due operatori che vedremo meglio dopo e che sono la restrizione in particolare e poi il relabeling, cioè la rietichettatura delle azioni

CCS: syntax

Sequential fragment

- Nil (or **0**) process (the atomic process)
- action prexing: **a.P**
- names and recursive definitions:
- non deterministic choice: **+**

Parallelism and renaming

- parallel composition: **|** (synchronous communication
btw two components = handshake synchronization)
- Restriction: **P \ L**
- Relabelling: **P[f]**

$$\begin{aligned}
 P &:= K \\
 &\alpha.P \\
 &\sum_{i \in I} P_i \\
 &P_1 | P_2 \\
 &P \setminus L \\
 &P[f]
 \end{aligned}$$

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

$$\text{Nil} = 0 = \sum_{i \in \emptyset} P_i$$



CCS: example (cont.)

- When a process executes it performs some action and becomes a new process. The execution of an action a is represented by a transition \xrightarrow{a} with label a

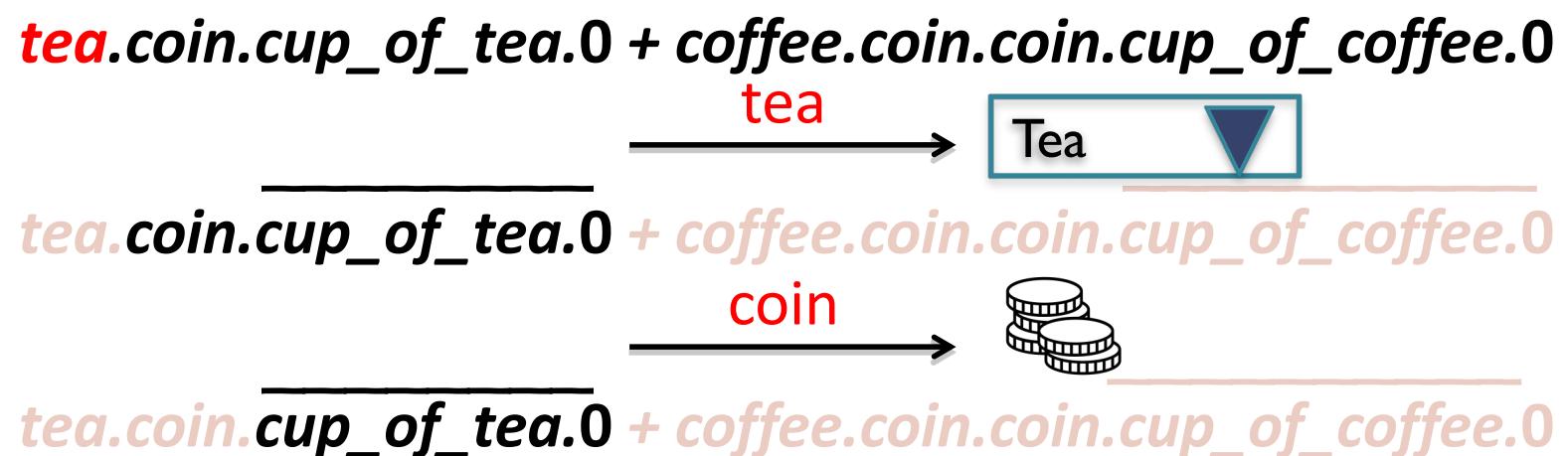


quindi riprendiamo l'esempio del distributore e vediamolo intanto dal punto di vista della macchina del distributore e quindi vediamo che si può fare evolvere questo sistema e questo processo della coffee machine nel seguente modo: cioè supponiamo che si scelga di prendere il tè a questo punto quello che succede è duplice, nel momento in cui scelgo il tè che fa parte di una scelta non deterministica, quello che succede è che sto sostanzialmente buttando via la scelta del caffè quindi vedete che nel target di questa freccia etichettata con te quello che succede è che la parte a destra appare in rosina chiaro che vuol dire che è sparita, ne lascia la traccia per capire la struttura dell'evoluzione del processo e quindi vado avanti sul branch a sinistra e allo stesso tempo avendo scelto te ho consumato se volete la prima azione che sta in pole position e mi rimane la continuazione che è coin seguita dall' output su cup of the

vado avanti,
quindi suppongo
di ricevere il
denaro e quindi
consumo la
seconda azione
che ho nel mio
processo a
sinistra e che è
diventata la
nuova azione in
pole position e
quindi non mi
resta che erogare
la cup of the,
arrivare al
processo Nil che
mi consente di
fermare il mio
andare avanti
questa è quindi
l'evoluzione
singola del
processo che fa il
distributore del
caffè

CCS: example (cont.)

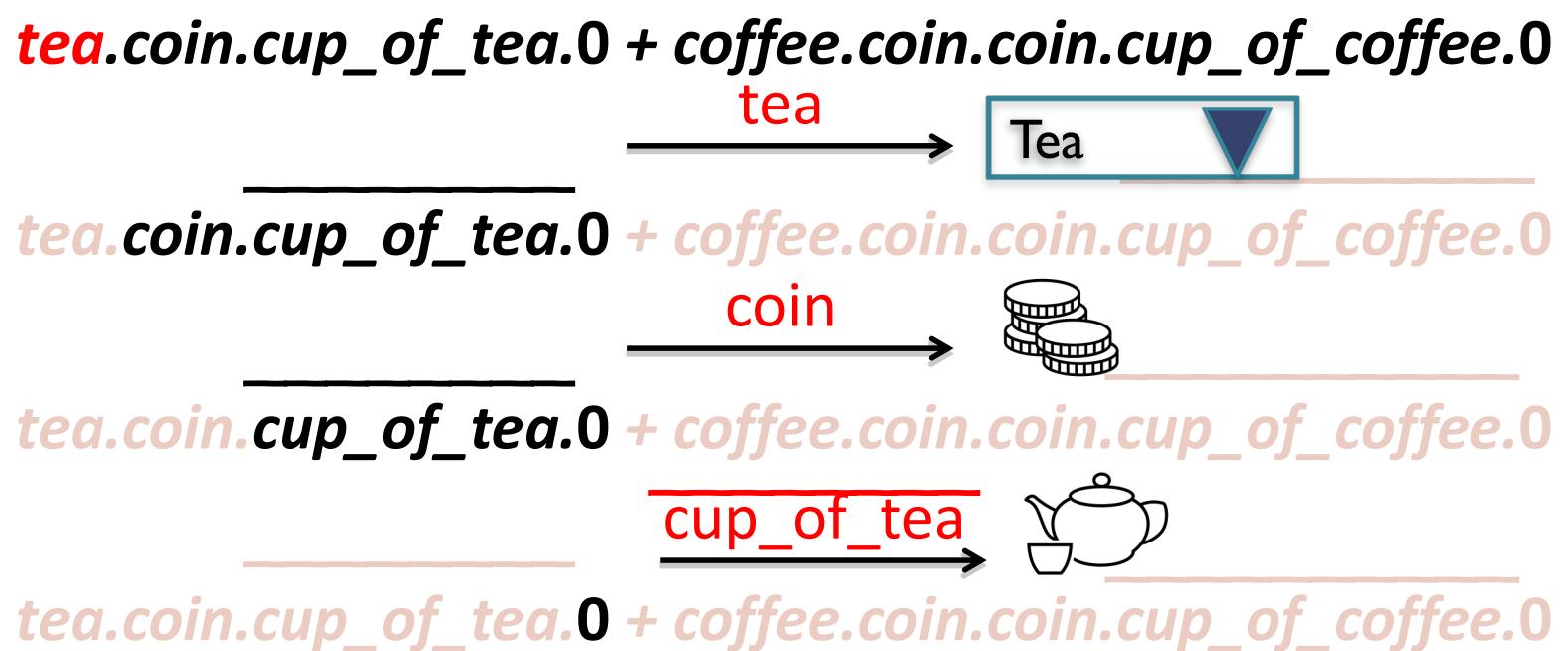
- When a process executes it performs some action and becomes a new process. The execution of an action a is represented by a transition \xrightarrow{a} with label a





CCS: example (cont.)

- When a process executes it performs some action and becomes a new process. The execution of an action a is represented by a transition \xrightarrow{a} with label a



CCS: example in parallel composition

$\overline{tea}.\overline{coin}.\overline{cup_of_tea}.0 + \overline{coffee}.\overline{coin}.\overline{coin}.\overline{cup_of_coffee}.0$

|

$\overline{tea}.\overline{coin}.\overline{cup_of_tea}.0 + \overline{coffee}.\overline{coin}.\overline{coin}.\overline{cup_of_coffee}.0$

τ

Tea



$\overline{tea}.\overline{coin}.\overline{cup_of_tea}.0 + \overline{coffee}.\overline{coin}.\overline{coin}.\overline{cup_of_coffee}.0$

|

$\overline{tea}.\overline{coin}.\overline{cup_of_tea}.0 + \overline{coffee}.\overline{coin}.\overline{coin}.\overline{cup_of_coffee}.0$

naturalmente ha senso vederlo in parallelo con chi questo caffè se lo prende, quindi supponiamo che uno di noi abbia un processo e computer scientist che si mette aa prendere il tè, il caffè al distributore di caffè, quindi a questo punto la stessa sequenza che abbiamo visto prima la vediamo in qualche modo composta dalle azioni equazioni che si sincronizzano tra tre le due parti della composizione parallela quindi vedete che da una parte ho il distributore di caffè e sotto ho lo studente, che sta davanti alla macchinetta, quindi quello che succede è che questa volta ha senso vedere le azioni e le coazioni in qualche modo collegate tra di loro perché sono fatte in maniera tale da corrispondersi quindi quello che succede è che una volta che chi sta davanti alla macchinetta del caffè preme il pulsante corrispondente a te, quello che sta facendo è una sincronizzazione tra l' input della macchinetta e il pulsante che saremo noi e questo mi porta ad avere quella che dicevo prima è un'azione tau, cioè una sincronizzazione tra i due che all'esterno non viene vista quindi all'esterno quello che vedo è che c'è stata un'interazione, questa interazione mi porta alla situazione della continuazione, quindi di nuovo come dicevamo prima, il fatto di aver scelto il tè porta all'eliminazione del secondo branch, sia nel caso del distributore che nel caso di chi lo prende il tè o caffè e quindi rimaniamo con la parte in nero e nella parte in nero è stata consumata l'azione e la coazione corrispondente.

la seconda interazione è lo scambio di monete e che la possiamo vedere come una sincronizzazione sul canale coin che mi dà le monete, quindi questo mi porta avanti nella computazione consuma anche la parte coin a questo punto e consumo la la

CCS: example (cont.)

$\overline{tea.\text{coin}.\text{cup_of_tea}.0} + \overline{coffee.\text{coin}.\text{coin}.\text{cup_of_coffee}.0}$

|

$\overline{tea.\text{coin}.\text{cup_of_tea}.0} + \overline{\overline{coffee.\text{coin}.\text{coin}.\text{cup_of_coffee}.0}}$

τ



$\overline{tea.\text{coin}.\text{cup_of_tea}.0} + \overline{\overline{coffee.\text{coin}.\text{coin}.\text{cup_of_coffee}.0}}$

|

$\overline{\overline{tea.\text{coin}.\text{cup_of_tea}.0}} + \overline{\overline{\overline{coffee.\text{coin}.\text{coin}.\text{cup_of_coffee}.0}}}$

terza sincronizzazione che è quella che dell'erogazione del tè per cui viene erogato il tè dalla macchina a chi ne fa richiesta e arrivo in entrambi i casi al parallelo dei due processi che non fanno nulla e quindi diciamo, ho finito il mio percorso

CCS: example (cont.)

$\overline{tea.coin.cup_of_tea.0} + \overline{coffee.coin.coin.cup_of_coffee.0}$

|

$\overline{tea.coin.cup_of_tea.0} + \overline{\overline{coffee}\overline{coin}\overline{coin}.cup_of_coffee.0}$

τ



$\overline{tea.coin.cup_of_tea.0} + \overline{coffee.coin.coin.cup_of_coffee.0}$

|

$\overline{tea.coin.cup_of_tea.0} + \overline{\overline{coffee}\overline{coin}\overline{coin}.cup_of_coffee.0}$

e sono tutte come vedete azioni tau e l'altra cosa che può essere interessante vedere è che quando ho un serie di transizioni, in questo caso tutte transazioni tau posso riassumere attraverso l'uso della stella di Kleene, questa avrete visto in moltissimi contesti che sta a indicare zero o più transizioni, quindi io posso dire che dal primo sistema all'ultimo in cui ho consumato tutto e sono arrivata al parallelo dei due processi nulli, ci sono arrivato in un certo numero di transazioni tau e questo è l'esempio che vi fa capire un pochino qual è la dinamica del modo in cui rappresento le interazioni

CCS: example (cont.)

$\overline{tea.coin.cup_of_tea.0} + \overline{coffee.coin.coin.cup_of_coffee.0}$

|

$\overline{tea.coin.cup_of_tea.0} + \overline{coffee}\overline{coin}\overline{coin}\overline{cup_of_coffee.0}$

τ

$*$

$0|0$

* is the **Kleene star**: means with **0 or many** transitions

ora molto più formalmente quello che abbiamo visto è un esempio di sistema di transizione etichettato che formalmente può essere visto come una tripla fatta sostanzialmente da un'insieme di processi che poi rappresentano gli stati, un'insieme di azioni rappresentati dalle etichette, e poi dall'altra, relazione di transizione, che è proprio quella che è sostanzialmente il prodotto cartesiano di processo per processo che quindi è una relazione binaria. La cosa che aiuta, diciamo l'intuizione è immaginare il sistema di transizione con un grafo dove i nodi sostanzialmente sono i processi e gli archi sono le transazioni, naturalmente vi potete immaginare che c'è uno stato di partenza e volendo uno stato di fine e lo stato di partenza è il sistema iniziale da cui partiamo

Labelled Transition Systems (LTS)

Definition

A **labelled transition system** (LTS) is a triple $(Proc, Act, \{\xrightarrow{\alpha} | \alpha \in Act\})$ where

- ▶ $Proc$ is a set of **processes** (the **states**),
- ▶ Act is a set of **actions** (the **labels**), and
- ▶ for every $\alpha \in Act$, $\xrightarrow{\alpha} \subseteq Proc \times Proc$ is a binary relation on processes called the **transition relation**

We can imagine a **graph**, where:

- **Nodes are processes**
- **Edges are transitions**
- **The start state is the initial system**

Labelled Transition Systems (LTS)

- ▶ Terminal process: 0

behavior: 0 $\not\rightarrow$

- ▶ Action prefixing: $\alpha.P$

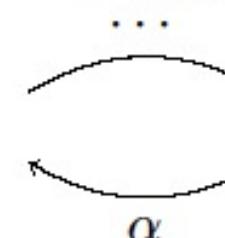
behavior: $\alpha.P \xrightarrow{\alpha} P$

- ▶ Non-deterministic choice: $\alpha.P + \beta.Q$

behavior: $P \xleftarrow{\alpha} \alpha.P + \beta.Q \xrightarrow{\beta} Q$

- ▶ Recursion: $X \stackrel{\text{def}}{=} \dots . \alpha.X$

behavior: $X \xrightarrow{\alpha} \alpha.X$



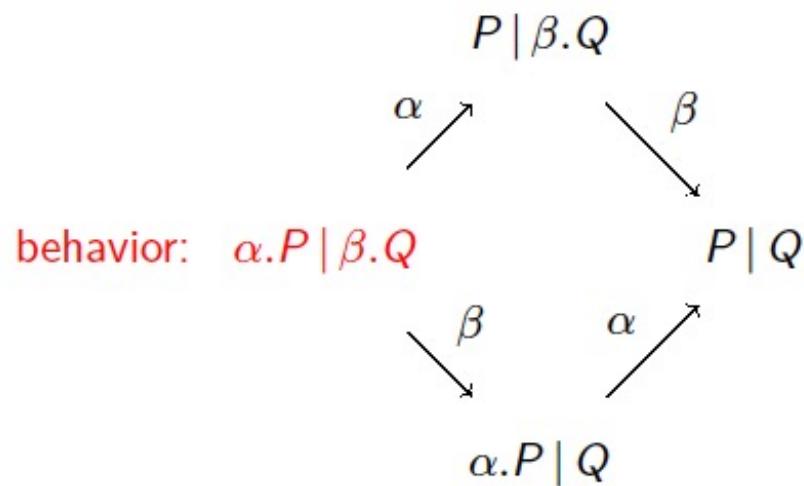
possiamo pensare che dal punto di vista del comportamento rappresentato dal transition system, se io ho un insieme, cioè diciamo ho il processo vuoto, sostanzialmente non posso evolvermi, quindi non è possibile che dal processo vuoto parta una freccia, mentre quando mi trovo ad avere una azione prefisso ad un processo quello che succede è che si evolve dall'alfa.P a P attraverso l'azione alfa, quando ho un scelta non deterministica, in realtà ho una diramazione perché è possibile andare da una parte oppure dall'altra, quindi il alfa.P oppure beta.Q mi può fare alfa ed andare in p oppure mi può fare beta ed andare in Q. Quindi questa è la rappresentazione di tutte proprio grafica diciamo se volete del del comportamento nel caso della scelta non deterministica e alla fine, come vi potrete immaginare se ho un comportamento di tipo ricorsivo, ci saranno dei cicli.



Labelled Transition Systems (LTS)

- ▶ Parallel composition: $\alpha.P \mid \beta.Q$

Combines sequential composition and choice to obtain **interleaving**

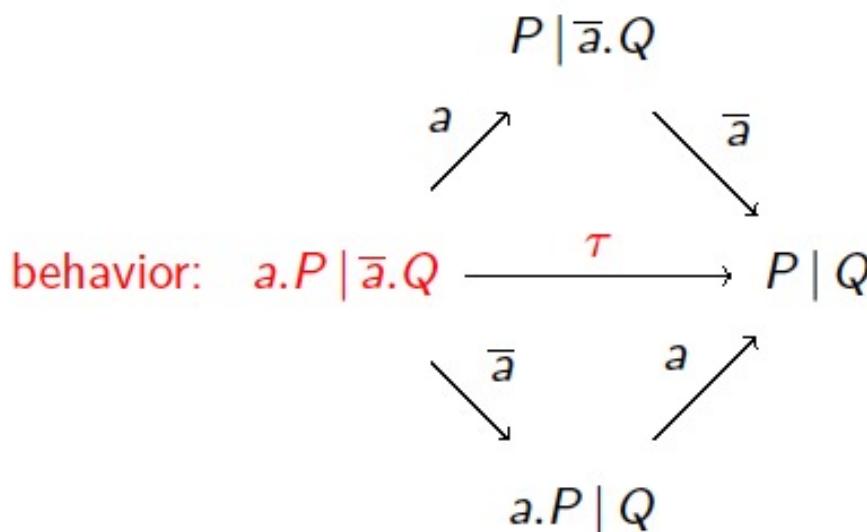


- ▶ What about interaction?

Altro discorso e merita farlo a parte quello della composizione parallela quindi che cosa ci aspettiamo? Quale sarà la rappresentazione grafica del comportamento parallelo di due processi? Abbiamo detto che due processi possono lavorare in maniera indipendente oppure coordinarsi tra di loro e quindi abbiamo la parte dell'indipendenza in cui supponete di avere una composizione parallela $\alpha.P \mid \beta.Q$ a questo punto, quello che succede è che ognuno di loro può andare avanti per conto proprio, quindi, la parte a sinistra può fare alfa e diventare P la parte destra può fare beta e diventare Q non ci sono assunzioni su chi fa prima cosa, il che vuol dire che il comportamento che posso avere può creare quelli che si chiamano interleaving, diversi, cioè può essere che questo sistema si evolva facendo prima alfa al processo a sinistra e poi beta al processo a destra o viceversa, possa partire con l'azione beta del processo a destra e continuare con l'azione alfa e arrivare comunque alla fine che è la composizione dei due stati di arrivo. Quindi questo è per quanto riguarda l'interleaving, cioè non è detto quale sia l'azione da fare per prima ci sono poi delle versioni con priorità in cui si potrebbe dire che l'azione alfa deve essere fatta sempre prima di beta, ma non è quello di cui parleremo oggi. A questo punto ci manca di capire che cosa succede sull'interazione, quindi che cosa succede quando queste due azioni che stanno a top level in entrambi i lati della composizione parallela non sono due azioni indipendenti ma sono due azioni che sono una la coazione dell'altra, diciamo due cose che si combinano tra di loro. A questo punto quello che succede è che ci sarà un'interazione stretta tra P e Q.

Labelled Transition Systems (LTS)

- ▶ Concurrent processes, i.e. P and Q in $P \mid Q$, may interact where their interfaces are **compatible**
- ▶ A synchronizing interaction between two processes (sub-systems), P and Q , is an activity that is **internal** to $P \mid Q$
- ▶ Parallel composition: $\alpha.P \mid \beta.Q$
Allows **interaction** if $\beta = \bar{\alpha}$

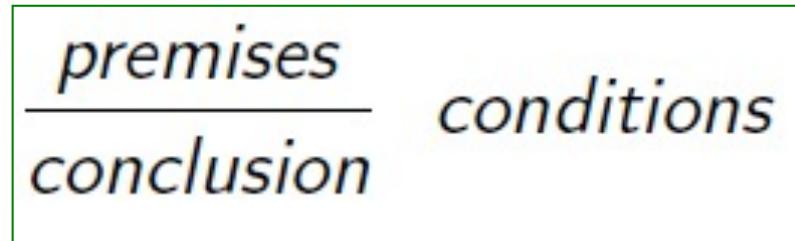


questa azione verrà etichettata da tau, ma sostanzialmente ci sta dicendo che i due processi si sincronizzano sull'azione, sul canale a quindi il processo a sinistra fornirà l'azione a e il processo a destra la sua azione corrispondente che è il coa e quindi diciamo questo conclude questo diagramma quindi questo ci consente di far vedere come le due azioni si possono in qualche modo combinare tra di loro



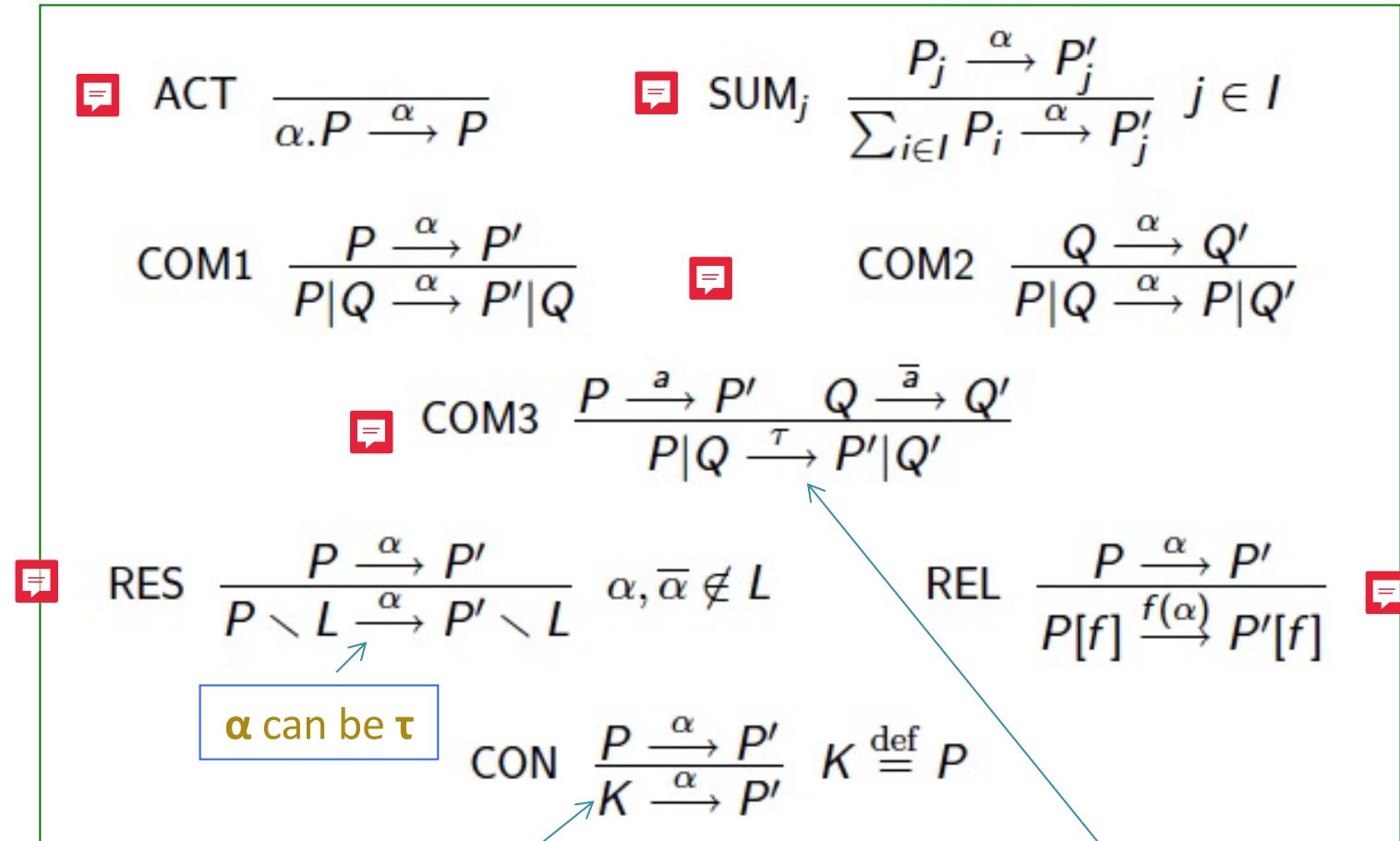
Structural Operational Semantics (SOS)

The behaviour of a system is inferred using **syntax-driven rules** in the form



a questo punto vediamo meglio formalmente come si vede la semantica operazionale di tipo strutturato, come si vede, la vediamo attraverso delle regole logiche che sono fatte nella forma premesse conclusioni con eventualmente condizioni al lato, quello che succede sostanzialmente che avremo quindi una sorta di forma logica per cui le conclusioni possono essere tratte soltanto se sono valide le premesse e in particolare ci sono alcune regole che oltre a queste, avranno bisogno di rispettare alcune condizioni che sono messe alla destra

SOS rules for CCS



The constant behaves as its defining expression

τ represents a completed and perfect action, internal to the composite agent P|Q



Derivations

- Derive

A cosa servono queste regole? Aervono a creare sistemi di transizione etichettati del tipo che abbiamo visto qua e naturalmente servono a capire l'evoluzione di un processo come abbiamo visto ancora prima dalla forma iniziale alla forma finale. Adesso dobbiamo cercare di capire come abbiamo agito in questi esempi in maniera strutturale rispetto alle regole della semantica quindi l'esempio che vi ho messo e che vorrei derivare insieme a voi è questo: cioè cosa voglio ottenere con quelle derivazioni? Attraverso le premesse, cioè attraverso le regole della semantica, è possibile derivare la transizione che mi porta da questo stato allo stato seguente, quindi è un singolo passo di transizione, come faccio a derivarlo? Utilizzo in maniera adeguata tutte le regole e le compongo in quello che viene chiamato albero di derivazione, cioè devo mettere insieme tutti i passi che mi consentono di passare dalle azioni di input e di output che quindi sono gli assiomi che stanno in cima pian piano ad arrivare ad utilizzare le regole per poter dire quello che mi aspetto, cioè io mi aspetto che ci sia una interazione tra il ramo sinistro e il ramo destro, che fa sì che si scelga la a e la coa e quindi si faccia una transizione tau che mi porta nel processo che nelle prime due parti ha consumato ha fatto quello che doveva fare e quindi è arrivato alla sua forma Nil. Allora diciamo che ci sono due modi di partire: una bottom up e un top down. Il vostro collega propone di partire dalla somma, la regola della somma + mi dice che se va avanti uno dei 2 rami gli altri li butto via e quindi è corretto la somma sarà sicuramente un ingrediente. Concentriamoci sulla premessa della somma: quindi io ho bisogno di sapere che c'è una transizione tra il ramo sinistro, in questo caso della somma, e quello che ottengo a destra. Adesso riguardiamo un attimo la transizione che vogliamo ottenere per arrivare a utilizzare la somma ho bisogno di una premessa, la premessa con cosa la costruisco? Quindi io devo sostanzialmente dire che da a.0 + b.0 una volta che ho fatto l'azione dove arriverà? A zero, quindi se io faccio questa somma che ho di qua a sinistra mi arriverà a zero. Però per fare questa operazione di scelta ho bisogno di sapere perché a.0 va a zero. Devo usare l'assioma che mi dice che posso fare l'azione.

$$((a.0 + b.0) \mid (\bar{a}.0 + c.0)) \mid Q \xrightarrow{\tau} 0 \mid 0 \mid Q$$

Derivations

- Derive

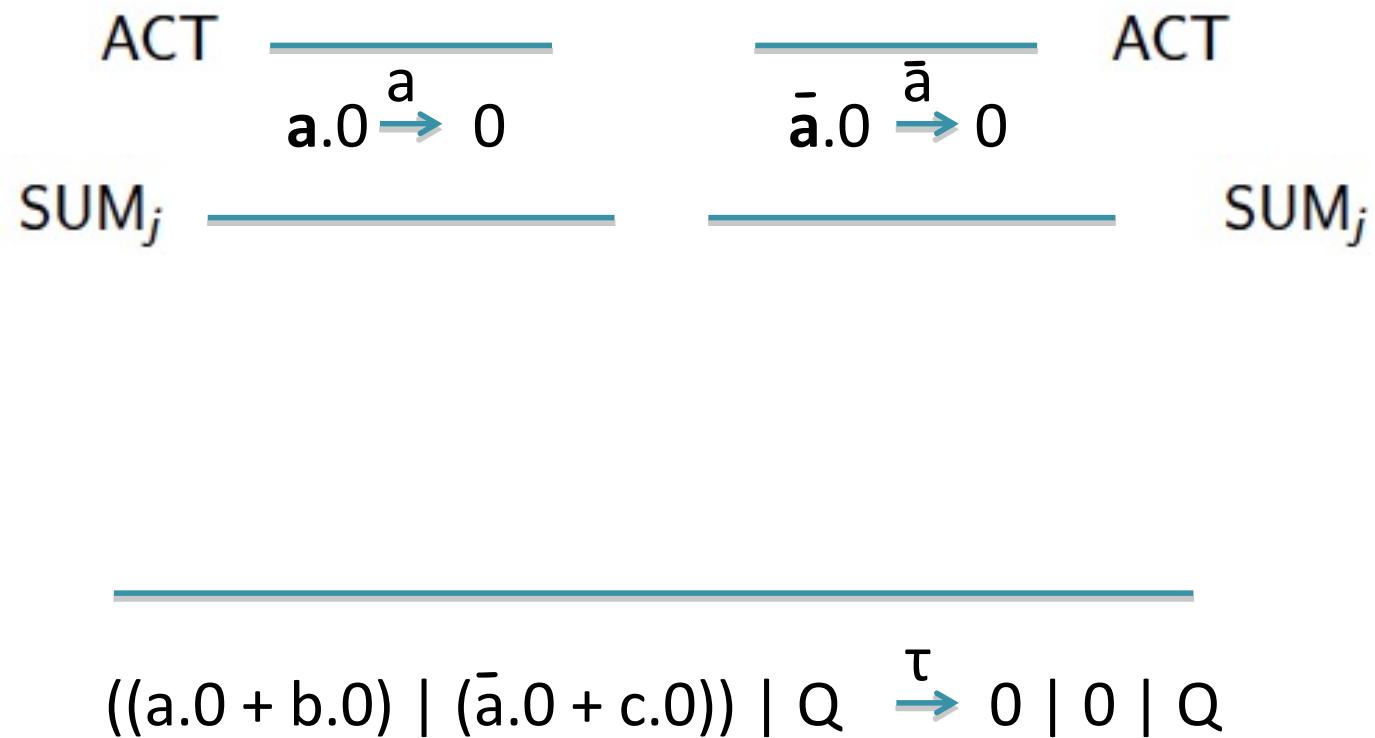


Quindi avrò che sul ramo sinistro della composizione sinistra della combinazione parallela avrò che $a.0$ fa a e diventa 0, e nello stesso modo a destra avrà che $\bar{a}.0$ diventa zero, Adesso posso usare l'assioma della somma. Quello che vi sto cercando di estrarre e di tirar fuori è che per derivare quella transizione, io ho bisogno di stanziare le regole, di metterle una sopra l'altra, quindi di fatto se ci pensate bene, se vi scordate l'aspetto grafico, stiamo facendo una sorta di dimostrazione, perché come faccio a derivare quella cosa, la derivo mettendo una sopra l'altra le regole istanziate al caso che abbiamo sotto mano, quindi io ho che $a.0$, sto instanziando la ACT, va in zero attraverso a . A questo punto devo istanziare la sum e nella mia sum io ho $a.0 + b.0$ che va a finire in zero. Io ho bisogno di sistemare questa parte qua ($a.0 + b.0$)

$$((a.0 + b.0) \mid (\bar{a}.0 + c.0)) \mid Q \xrightarrow{\tau} 0 \mid 0 \mid Q$$

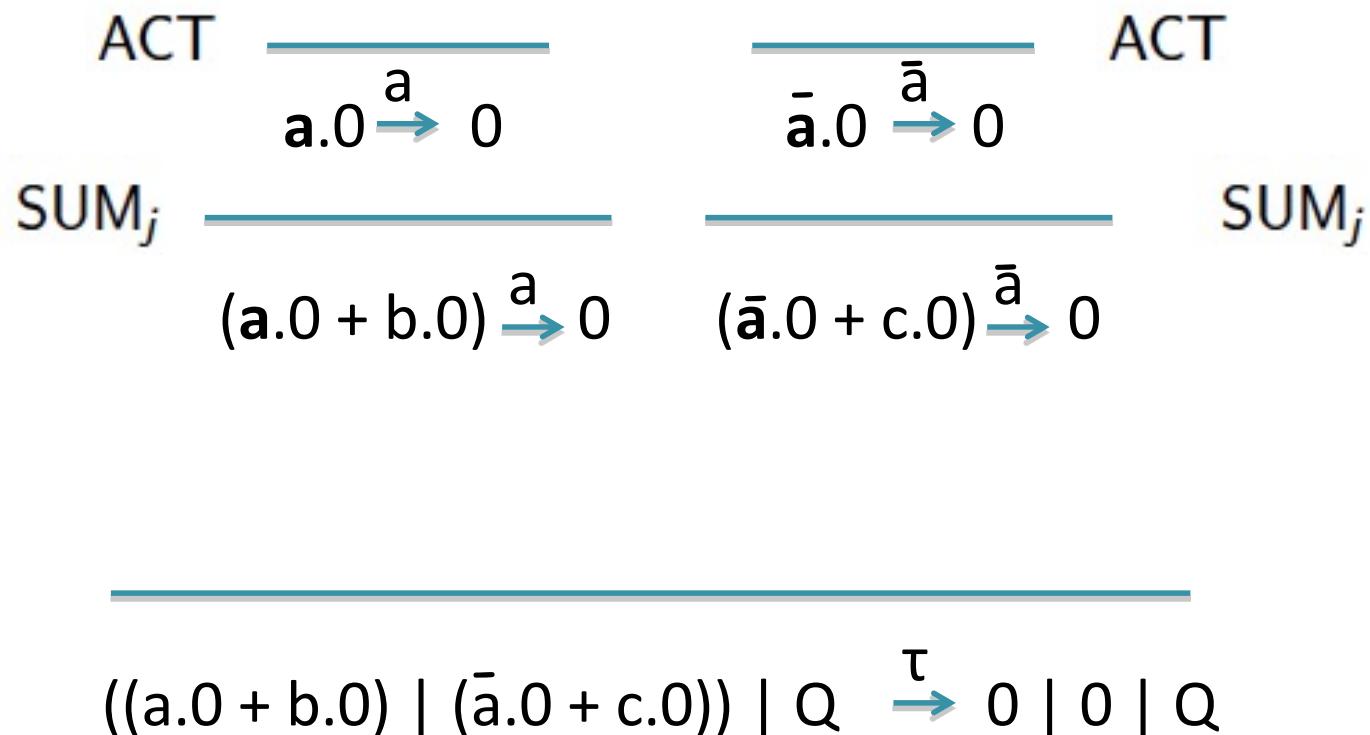
Derivations

- Derive



Derivations

- Derive



quindi io uso la somma in tutti e due casi e cosa ne viene fuori? Siccome $a.0$ va in zero, posso dire che $a.0 + b.0$ va in 0 e naturalmente in maniera simmetrica, quindi ho sistemato intanto quel pezzettino, quindi io sono sicura che le due parti della composizione parallela adesso fanno proprio l'azione e la coazione che mi servono poi per la sincronizzazione su a . Non è finito perché c'è quel dannato processo Q che non ci sta a fare niente e che non fa niente. Proprio perché non fa niente che cosa succede? Siamo in uno dei casi che abbiamo visto prima, abbiamo il caso in cui P si muove e Q no, cioè la parte a sinistra della mia composizione parallela si muove, ma Q non fa nulla.

Derivations

- Derive

$$\begin{array}{c} \text{ACT} \quad \overline{\qquad\qquad\qquad} \\ a.0 \xrightarrow{a} 0 \end{array} \quad \begin{array}{c} \overline{\qquad\qquad\qquad} \quad \text{ACT} \\ \bar{a}.0 \xrightarrow{\bar{a}} 0 \end{array}$$
$$\begin{array}{c} \text{SUM}_j \quad \overline{\qquad\qquad\qquad} \\ (a.0 + b.0) \xrightarrow{a} 0 \end{array} \quad \begin{array}{c} \overline{\qquad\qquad\qquad} \quad \text{SUM}_j \\ (\bar{a}.0 + c.0) \xrightarrow{\bar{a}} 0 \end{array}$$
$$\begin{array}{c} \text{COM3} \quad \overline{\qquad\qquad\qquad} \\ (a.0 + b.0) \mid (\bar{a}.0 + c.0) \xrightarrow{\tau} 0 \mid 0 \end{array}$$
$$\overline{\qquad\qquad\qquad}$$
$$((a.0 + b.0) \mid (\bar{a}.0 + c.0)) \mid Q \xrightarrow{\tau} 0 \mid 0 \mid Q$$

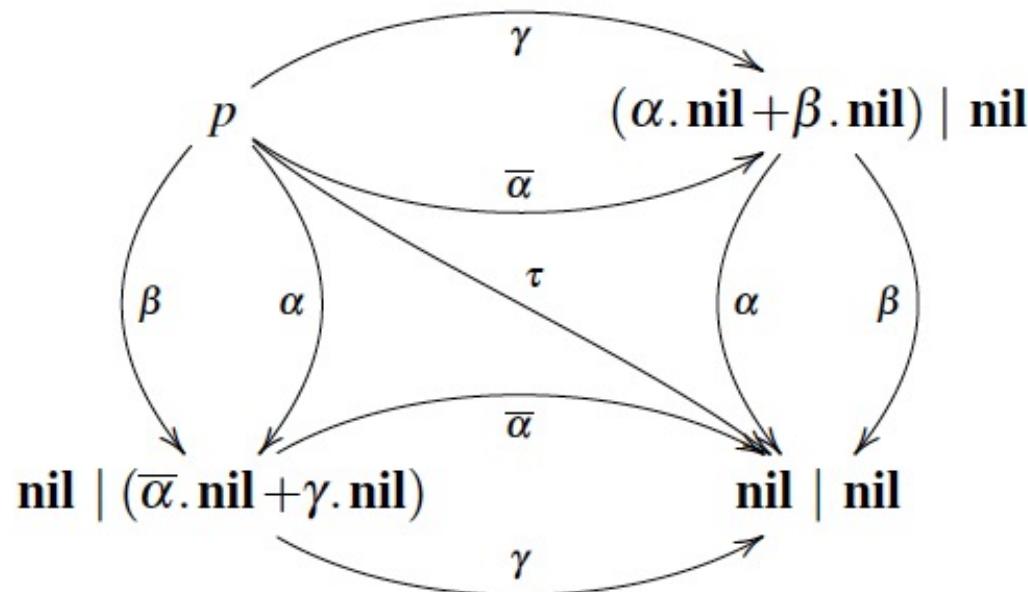
Derivations

- Derive

$$\begin{array}{c} \text{ACT} \quad \overline{\qquad\qquad\qquad} \\ \text{a.0} \xrightarrow{a} 0 \end{array} \quad \begin{array}{c} \overline{\qquad\qquad\qquad} \quad \text{ACT} \\ \bar{a}.0 \xrightarrow{\bar{a}} 0 \end{array}$$
$$\begin{array}{c} \text{SUM}_j \quad \overline{\qquad\qquad\qquad} \\ (a.0 + b.0) \xrightarrow{a} 0 \end{array} \quad \begin{array}{c} \overline{\qquad\qquad\qquad} \quad \text{SUM}_j \\ (\bar{a}.0 + c.0) \xrightarrow{\bar{a}} 0 \end{array}$$
$$\begin{array}{c} \text{COM3} \quad \overline{\qquad\qquad\qquad\qquad\qquad\qquad} \\ (a.0 + b.0) \mid (\bar{a}.0 + c.0) \xrightarrow{\tau} 0 \mid 0 \end{array}$$
$$\begin{array}{c} \text{COM1} \quad \overline{\qquad\qquad\qquad\qquad\qquad\qquad} \\ ((a.0 + b.0) \mid (\bar{a}.0 + c.0)) \mid Q \xrightarrow{\tau} 0 \mid 0 \mid Q \end{array}$$

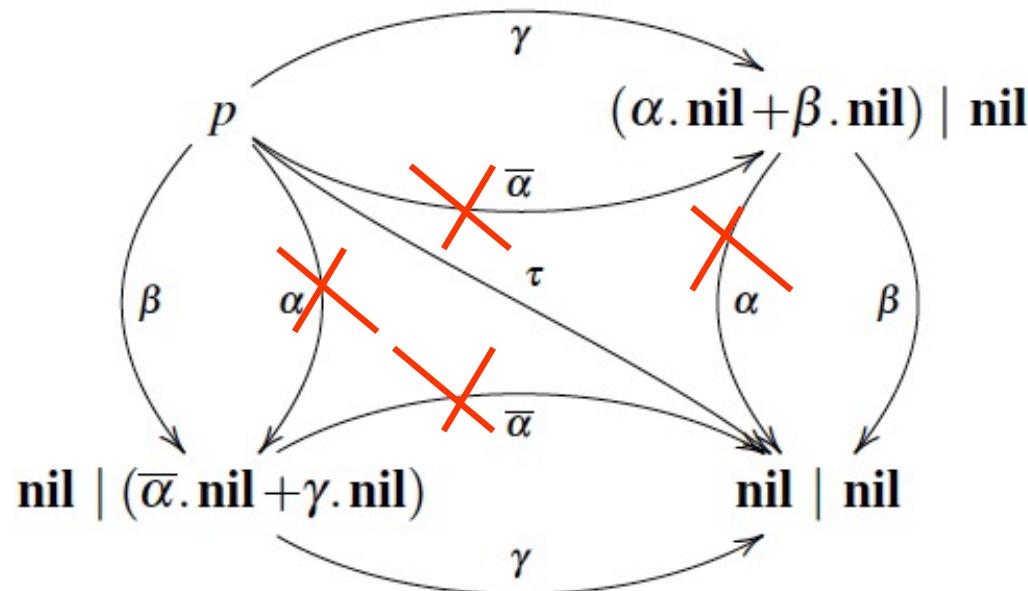
LTS: an example

$$p \stackrel{\text{def}}{=} (\alpha.\text{nil} + \beta.\text{nil}) \mid (\bar{\alpha}.\text{nil} + \gamma.\text{nil})$$



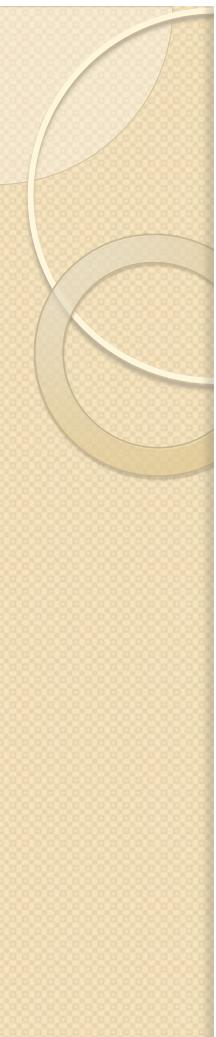
LTS: an example (cont.)

$$p \stackrel{\text{def}}{=} (\alpha.\text{nil} + \beta.\text{nil}) \mid (\bar{\alpha}.\text{nil} + \gamma.\text{nil})$$



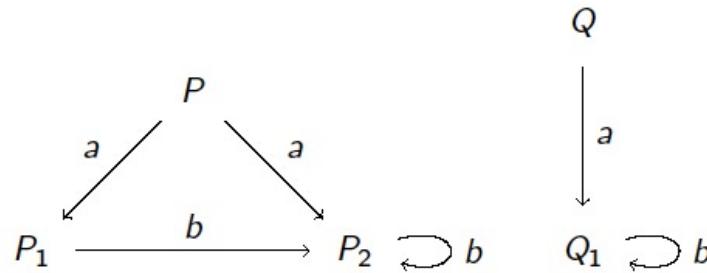
$$q \stackrel{\text{def}}{=} p \setminus \alpha$$

With **restrictions** we can produce **closed** systems:
restricted names are unaccessible to the environment



Behavioural equivalence

- Express the notions that two processes behave in the same way...
- ... accordingly to an external observer



Se riprendiamo la nostra sintassi vedo che ci sono vari modi per comporre processi e ci sono una serie di cose che mi fanno pensare che ci siano dei processi che si comportano in maniera molto simile addirittura ci sono dei processi che probabilmente più o meno si comportano nello stesso modo, ad esempio, alla slide 33, se io cambiassi la posizione di $a.0 + b.0$ con quella di $coa.0 + c.0$ diciamo il comportamento complessivo cambierebbe molto? Cioè se io invertissi proprio i due rami? Non dovrebbe. Così come se io inverta i due rami del più nella coffee machine cambia? Non molto, ci sono allora alcuni processi che sono diversi dal punto di vista sintattico ma che sono simili in qualche modo dal punto di vista dei comportamenti e ci arriviamo per gradi, nel senso che cominciate a poter immaginare che ci sono delle leggi algebriche come avevamo detto, e potete anche immaginare che due processi in parallelo si comportano in maniera analoga in qualunque modo si dispongano, cioè che p parallelo q alla fine si possa più o meno comportare come q parallelo p . Ma non è solo questo, cioè ci sono dei sistemi che mi faranno vedere quando due processi che sono anche sintatticamente molto diversi, anche non rearrangibili attraverso leggi algebriche che comunque si comportano in maniera simile. Ed è questa l'idea della teoria che sta dietro all'equivalenza comportamentale, cioè, detto in soldoni, si esprime la nozione che due processi si comportano più o meno nello stesso modo, o perlomeno lo facciano secondo l'occhio di un osservatore esterno ad esempio, questi due grafi che rappresentano il sistema di transizione di due processi p e q ci fanno riflettere, cioè questi due processi sono diversi, hanno un'evoluzione diversa, ma possiamo dire che più o meno fanno la stessa cosa? P può fare a ed andare in P oppure fare a e andare in P_2 da P_1 si va in P_2 con una b e da P_2 si cicla su P_1 . Invece il processo Q fa una singola azione a e va in Q_1 e su Q_1 cicla sull'azione b . Si può dire che questi due processi si comportano in maniera analoga? In entrambi i casi io come prima cosa che posso fare è una a e come seconda comincia a fare delle b . Se uno dovesse vederlo sotto forma di traccia, cioè sequenza di azioni, io posso sempre avere che prima si fa una a e poi un certo numero di b . Ora formalmente come si può rendere questa intuizione?



Behavioural equivalence

Strong Bisimilarity

Let $(Proc, Act, \{ \xrightarrow{\alpha} \mid \alpha \in Act \})$ be an LTS

Strong Bisimulation

A binary relation $R \subseteq Proc \times Proc$ is a **strong bisimulation** iff whenever $(P, Q) \in R$ then for each $\alpha \in Act$:

- ▶ if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ for some Q' such that $(P', Q') \in R$
- ▶ if $Q \xrightarrow{\alpha} Q'$ then $P \xrightarrow{\alpha} P'$ for some P' such that $(P', Q') \in R$

Strong Bisimilarity

Two processes $P_1, P_2 \in Proc$ are **strongly bisimilar** ($P_1 \sim P_2$) if and only if there exists a strong bisimulation R such that $(P_1, P_2) \in R$

$$\sim = \cup\{R \mid R \text{ is a strong bisimulation}\}$$

C'è proprio una nozione, ce ne sono varie in realtà, adesso vediamo quella che viene chiamata strong equivalence o strong similarity, che mi dice che si può creare una relazione che si chiama strong simulation, ogni volta io prendo due processi che stanno dentro questa relazione e succede questo: cioè se P si evolve con un'azione alfa in P' , allora anche Q si deve evolvere in Q' usando la stessa azione e arrivare entrambi in due stati che ancora appartengono alla relazione, cioè che ancora sono nella relazione di strong simulation e viceversa. Quello che fa uno può essere simulato dall'altro e viceversa. Quindi quand'è che due processi si possono dire strong bisimilar? Se e solo se esiste una relazione di questo tipo che fa sì che i due facciano parte della relazione. A questo punto, per vedere che due processi sono bisimili basta mostrare che esiste proprio una relazione che fa questo.

Behavioural equivalence

Strong Bisimilarity

Let $(Proc, Act, \{ \xrightarrow{\alpha} \mid \alpha \in Act \})$ be an LTS

Strong Bisimulation

A binary relation $R \subseteq Proc \times Proc$ is a **strong bisimulation** iff whenever $(P, Q) \in R$ then for each $\alpha \in Act$:

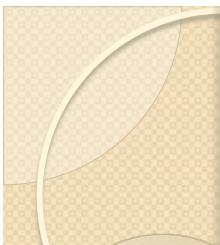
- ▶ if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ for some Q' such that $(P', Q') \in R$
- ▶ if $Q \xrightarrow{\alpha} Q'$ then $P \xrightarrow{\alpha} P'$ for some P' such that $(P', Q') \in R$

Strong Bisimilarity

Two processes $P_1, P_2 \in Proc$ are **strongly bisimilar** ($P_1 \sim P_2$) if and only if there exists a strong bisimulation R such that $(P_1, P_2) \in R$

$$\sim = \cup\{R \mid R \text{ is a strong bisimulation}\}$$

- ▶ Two processes are bisimilar if there is a concrete strong bisimulation relation that relates them
- ▶ To **show** that two processes are bisimilar it suffices to exhibit such a concrete relation

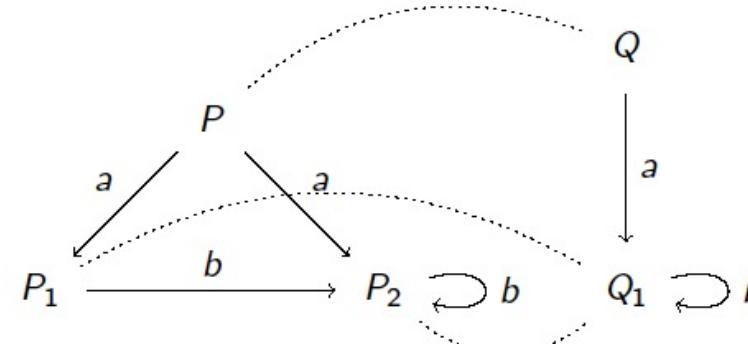


Behavioural equivalence

ad esempio il caso che abbiamo visto, vuol dire mettere insieme sostanzialmente P1 e P2 e metterli insieme tutti e due con Q1. Quindi in particolare possiamo vedere che se p e q stanno in relazione p fa un'azione alfa, è vero che anche q la può fare. Che la faccia di qui o di là non importa e in entrambi i casi si va a finire o in P1 o in P2 che però entrambi sono collegati dalla relazione con q1 e una volta che sono collegati o P1 o P2 con Q1 è vero che sia P1 che Q1 possono fare b sia P2 che Q1 possono fare b. Quindi l'idea è che io possa stabilire delle equivalenze osservazionali e comportamentali laddove processi sono scritti anche in maniera diversa, ma sostanzialmente poi possono fare le stesse cose. Non è sempre così, sembra ovvio, ma non è sempre così.

$$\mathcal{R} = \{(P, Q), (P_1, Q_1), (P_2, Q_1)\}$$

- ▶ (P, Q) is in \mathcal{R}
- ▶ \mathcal{R} is a bisimulation:
 - ▶ For each pair of states in \mathcal{R} , all possible transitions from the first can be matched by corresponding transitions from the second
 - ▶ For each pair of states in \mathcal{R} , all possible transitions from the second can be matched by corresponding transitions from the first



Behavioural equivalence

$$\begin{aligned} P &\stackrel{\text{def}}{=} a.(b.0 + c.0) \\ Q &\stackrel{\text{def}}{=} a.b.0 + a.c.0 \end{aligned}$$

$$P \not\sim Q$$

After the action a , in P there is still the choice between the two branches,
While in Q , there is not (the choice has been made)

Ad esempio qui non c'è il disegno, poi ve lo farò questi due processi non sono invece bisimili. Allora i processi sono P che fa prima a e dopo può scegliere tra fare b oppure c . Invece Q è fatto in maniera per cui prima fa a e poi fa b oppure fa a e poi fa c . Perché questi due processi non godono di questa equivalenza? In che cosa si distinguono? Quindi, sembra apparentemente che facciano la stessa cosa, ma in realtà nel caso di P io una volta che ho fatto a ho ancora la possibilità di scegliere se fare b oppure c , mentre se sono nel secondo caso, una volta che ho scelto a ho già scelto anche quale ramo faccio. Farò a del ramo di sinistra o a del ramo destra. Quindi se scelgo il ramo a sinistra ho già scelto di fare anche b oppure se scelgo il ramo destro ho già scelto di fare anche c , ora il disegno non ho fatto in tempo a farlo, ma se ci pensate dal punto di vista del disegno sono proprio diversi, perché c'ho una scelta che si apre dopo a nel primo caso è una scelta che si apre a monte nel caso di q . La cosa interessante da vedere qua è che però questa differenza la notate solo nel momento in cui avete un grafo, perché vedete le diramazioni, se invece uno facesse un'osservazione di tipo traccia, non se ne accorgerebbe, perché le tracce in entrambi i casi l'insieme delle tracce possibili sono $a-b$ oppure $a-c$. Il modello del transition system, è un modello che fa sì che vediate la differenza tra questi due processi, mentre se uno utilizzasse un processo a tracce che per sua natura non è ramificato, quindi è piatto e lineare, uno vedrebbe tracce dello stesso tipo.



Outline

- Motivation
- Key notion
- CCS
- **From CCS to Pi calculus**
- Pi calculus: syntax and semantics
- Conclusions and what next



CCS vs Pi Calculus

CCS

$$a.P \mid \bar{a}.Q \xrightarrow{\tau} P \mid Q$$

Value-passing CCS

can be encoded in pure CCS

$$a(x).P \mid \bar{a}\langle v \rangle.Q \xrightarrow{\tau} P\{v/x\} \mid Q$$

x is bound in P, v is a value

Pi calculus

y is bound in P, z can be a channel name

$$x(y).P \mid \bar{x}\langle z \rangle.Q \xrightarrow{\tau} P\{z/y\} \mid Q$$



Towards Value-passing CCS

Suppose C wants to buy a pizza from P*

$$C \triangleq \overline{\text{askPizza}}.\overline{\text{pay}}.\text{pizza}$$

$$P \triangleq \text{askPizza}.\text{pay}.\overline{\text{pizza}}.P$$

CCS

$$C|P \xrightarrow{\tau} \overline{\text{pay}}.\text{pizza}|\text{pay}.\overline{\text{pizza}}.P \xrightarrow{\tau} \text{pizza}|\overline{\text{pizza}}.P \xrightarrow{\tau} 0|P \equiv P$$

Clear in a while

abbiamo che il cliente può chiedere una pizza, pagarla e poi aspettarla e quindi corrispondentemente, adesso vi dovrebbe essere chiaro il meccanismo, il pizzaiolo aspetta una richiesta per la pizza, aspetta per il pagamento e quindi procede al delivery. Questo è quanto potevo ottenere con la modellazione del ccs puro. L'evoluzione è quella che uno si aspetta, a questo punto dovreste essere esperti, si passa a fare la sincronizzazione sulla richiesta per poi fare la sincronizzazione sul pagamento e quindi la transizione della consegna della pizza. Il ccs ci offre la modellazione di quella che viene chiamata sincronizzazione handshake, cioè stretta di mano non c'è passaggio di parametri a questo poi,

Handshake synchronization

* [R. De Nicola, R. Bruni]

Towards Value-passing CCS

Suppose C wants to buy a pizza from P*

$$C \triangleq \overline{\text{askPizza}}.\overline{\text{pay}}.\text{pizza}$$

$$P \triangleq \text{askPizza}.\text{pay}.\overline{\text{pizza}}.P$$

CCS

$$C|P \xrightarrow{\tau} \overline{\text{pay}}.\text{pizza}|\text{pay}.\overline{\text{pizza}}.P \xrightarrow{\tau} \text{pizza}|\overline{\text{pizza}}.P \xrightarrow{\tau} 0|P \equiv P$$

Clear in a while

$$C \triangleq \overline{\text{askPizza}}(\text{margherita}).\overline{\text{pay}}(5 \text{ Euro}).\text{pizza}$$

$$\begin{aligned} P \triangleq & \text{askPizza}(x).\text{pay}(y).\text{if } y = \text{price}(x) \text{ then } \overline{\text{pizza}}.P \\ & \text{else if } y > \text{price}(x) \text{ then } \overline{\text{pizza}}.\overline{\text{output}}(y - \text{price}(x)).P \\ & \text{else } \overline{\text{askMoney}}... \end{aligned}$$

Value
passing
CCS

Handshake synchronization

+

the possibility to exchange data

* [R. De Nicola, R. Bruni]

Towards Pi Calculus

Home delivery!



$$\begin{aligned} C &\triangleq \overline{\text{askPizza}}(\text{myHome}).\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \\ P &\triangleq \text{askPizza}(y).\text{pay}.(\nu \text{pizza})\overline{y}(\text{pizza}).P \end{aligned}$$

Pi Calculus

$$\overline{\text{askPizza}}(\text{myHome}).\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \mid \\ \text{askPizza}(y).\text{pay}.(\nu \text{pizza})\overline{y}(\text{pizza}).P$$

... + the possibility to create and send channel names

Towards Pi Calculus

Home delivery

$$\begin{aligned} C &\triangleq \overline{\text{askPizza}}(\text{myHome}).\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \\ P &\triangleq \text{askPizza}(y).\text{pay}.(\nu \text{pizza})\overline{y}(\text{pizza}).P \end{aligned}$$

Pi Calculus

$$\begin{aligned} &\overline{\text{askPizza}}(\text{myHome}).\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \mid \\ &\quad \text{askPizza}(y).\text{pay}.(\nu \text{pizza})\overline{y}(\text{pizza}).P \\ &\xrightarrow{\tau} \overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \quad \mid \quad \text{pay}.(\nu \text{pizza})\overline{\text{myHome}}(\text{pizza}).P \end{aligned}$$

... + the possibility to create and send channel names

Towards Pi Calculus

Home delivery

$$\begin{aligned} C &\triangleq \overline{\text{askPizza}}(\text{myHome}).\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \\ P &\triangleq \text{askPizza}(y).\text{pay}.(\nu \text{pizza})\overline{y}(\text{pizza}).P \end{aligned}$$

Pi Calculus

$$\begin{aligned} &\overline{\text{askPizza}}(\text{myHome}).\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \mid \\ &\quad \text{askPizza}(y).\text{pay}.(\nu \text{pizza})\overline{y}(\text{pizza}).P \\ \xrightarrow{\tau} &\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \mid \text{pay}.(\nu \text{pizza})\overline{\text{myHome}}(\text{pizza}).P \end{aligned}$$

... + the possibility to exchange channel names

Towards Pi Calculus

Home delivery

$$\begin{aligned} C &\triangleq \overline{\text{askPizza}}(\text{myHome}).\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \\ P &\triangleq \text{askPizza}(y).\text{pay}.(\nu \text{pizza})\overline{y}(\text{pizza}).P \end{aligned}$$

Pi Calculus

$$\begin{array}{c} \overline{\text{askPizza}}(\text{myHome}).\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \mid \\ \hspace{10em} \text{askPizza}(y).\text{pay}.(\nu \text{pizza})\overline{y}(\text{pizza}).P \\ \xrightarrow{\tau} \overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}(x) \quad \mid \quad \text{pay}.(\nu \text{pizza})\overline{\text{myHome}}(\text{pizza}).P \\ \xrightarrow{\tau} \text{myHome}(x).\overline{\text{eat}}(x) \quad \mid \quad (\nu \text{pizza})\overline{\text{myHome}}(\text{pizza}).P \end{array}$$

... + the possibility to exchange channel names

a questo punto avremo un'azione per il pagamento che questa è ccs like like e alla fine avremo la comunicazione su my home e quindi il pizzaiolo potrà mandare la pizza nuova appena fatta sull'indirizzo indicato e quindi si potrà avere l'ultimo pezzettino e naturalmente il cliente, a questo punto potrà mangiarla

Towards Pi Calculus

Home delivery

$$\begin{aligned} C &\triangleq \overline{\text{askPizza}}\langle\text{myHome}\rangle.\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}\langle x\rangle \\ P &\triangleq \text{askPizza}(y).\text{pay}.(\nu\text{pizza})\overline{y}\langle\text{pizza}\rangle.P \end{aligned}$$

$$\begin{array}{c} \overline{\text{askPizza}}\langle\text{myHome}\rangle.\overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}\langle x\rangle \mid \\ \hspace{10em} \text{askPizza}(y).\text{pay}.(\nu\text{pizza})\overline{y}\langle\text{pizza}\rangle.P \\ \xrightarrow{\tau} \overline{\text{pay}}.\text{myHome}(x).\overline{\text{eat}}\langle x\rangle \quad \mid \quad \text{pay}.(\nu\text{pizza})\overline{\text{myHome}}\langle\text{pizza}\rangle.P \\ \xrightarrow{\tau} \text{myHome}(x).\overline{\text{eat}}\langle x\rangle \quad \mid \quad (\nu\text{pizza})\overline{\text{myHome}}\langle\text{pizza}\rangle.P \\ \xrightarrow{\tau} (\nu\text{pizza})(\overline{\text{eat}}\langle\text{pizza}\rangle) \quad \mid \quad P \end{array}$$

... + the possibility to exchange channel names

questa quindi alla
diciamo un esempio che
ci fa capire la
motivazione che dicevo
prima e cioè che
abbiamo bisogno di
scambiare i valori e
scambiare anche la
possibilità di comunicare
quindi sostanzialmente
la mobilità di cui
parlavamo proprio
all'inizio della lezione che
è quella scelta dal pi
calcolo è quella relativa
ai link, cioè quello che
viene modificato, quello
che è mobile è la trama
delle possibili
comunicazioni, è lo
schema delle capability
delle comunicazioni,
quindi un processo
all'inizio può comunicare
su un certo numero di
canali con altri processi.
Dopo ogni transizione
questo insieme potrebbe
essere modificato quindi
quello che cambia è la
mobilità della capability
ad accedere ad un certo
canale.

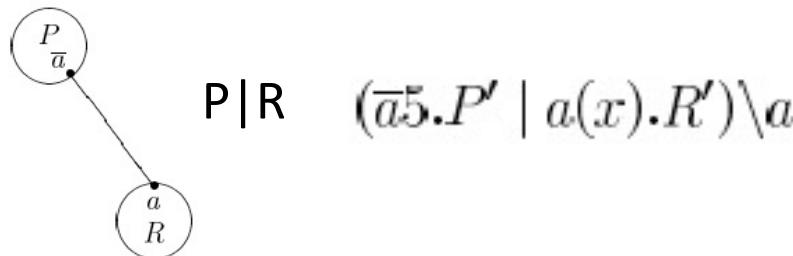
Towards the Pi Calculus

It is not difficult to extend the calculus (CCS) to allow for the **exchange of values** such as integers over the channel during a synchronization. Doing so does not fundamentally change the character of the calculus. A variation on this, however, does change the calculus in a highly nontrivial way, and that is to **allow for the communication of channel names during synchronization**. This yields the pi-calculus. To see why such an extension might be useful, consider the following scenario. It shows that passing channel names around can be used to model process mobility. Intuitively, a process is characterized by the channels it exposes to the world, that can be used to communicate with it. These channels act as an interface to the process. [...] A process that sends x to another process in some sense sends the capability to access x to that process. This captures the mobility of process P , although more accurately it captures the **mobility of the capability to access P** .

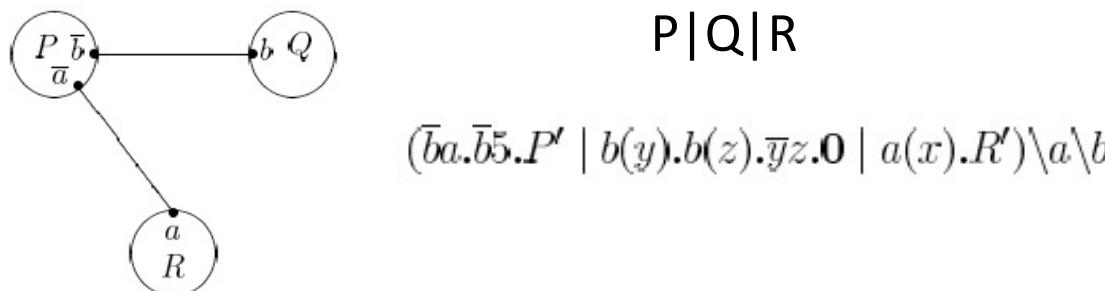
Review of **The pi-calculus: A Theory of Mobile Processes**
by D. Sangiorgi and D. Walker Riccardo Pucella

Mobility

- Suppose P wishes to send 5 to R, along the private channel **a**



- Now, suppose that P wish to delegate to Q the transmission of 5 (P is connected with Q via **b**)



Mobility: name extrusion



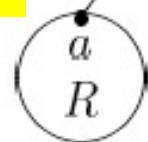
- After two communications, both along **b**:

$$(P' \mid \bar{a}5.0 \mid a(x).R') \setminus a \setminus b$$
$$P \mid R$$

Restriction is not a static operator



If a does not appear in P'



- Note that a, b, x, y, 5 are all just names
- The other class of entities are agents

- P's link **a** has moved to Q



as a reference



Conclusions and what next

- Process algebras: models for concurrent systems
- Pi calculus is an evolution of CCS

Next

- We will see Pi calculus more in detail