Process Algebras and Concurrent Systems

Rocco De Nicola

Dipartimento di Sistemi ed Informatica
Università di Firenze

Process Algebras and Concurrent Systems
August 2006

Problems with Concurrent Programming

Developing a Concurrent Solution to a Simple Problem

Problem:

Write a program that

- terminates if the total function f has a (positive or negative) zero
- proceeds indefinitely otherwise.

Assume we have a program that looks for positive zeros:

```
S1 =   found := false; x := 0;
       while (not found)
             do x:= x+1; found := (f(x) = 0) od
```

Starting from the above, we can build the program looking for negative zeros.

```
S2 =    found := false; y := 0;
        while (not found)
              do y:= y-1; found := (f(y) = 0) od
```

Attempt 1

An obvious solution would be running `S1` and `S2` in parallel:

```
S1 || S2
```

Attempt 1

An obvious solution would be running S1 and S2 in parallel:

```
S1 || S2
```

However ...

If f has a positive zero and not a negative one, and S1 terminates before S2 starts, the latter sets found to false and looks indefinitely for a nonexisting zero.

Attempt 1

An obvious solution would be running `S1` and `S2` in parallel:

```
S1 || S2
```

However ...

If f has a positive zero and not a negative one, and `S1` terminates before `S2` starts, the latter sets found to false and looks indefinitely for a nonexisting zero.

The problem is due to the fact that found is initialized to false twice.

LESSON 1

Care is needed when handling shared variables.

Attempt 2

Let us consider a solution that initializes found only once.

```
found := false; (R1 || R2)          where
R1 =  x := 0; while (not found)
                  do x:= x+1; found := (f(x) = 0) od
R2 =  y := 0 while (not found)
                  do y:= y-1; found := (f(y) = 0) od
```

Attempt 2

Let us consider a solution that initializes found only once.

```
found := false; (R1 || R2)          where
R1 =  x := 0; while (not found)
                  do x:= x+1; found := (f(x) = 0) od
R2 =  y := 0 while (not found)
                  do y:= y-1; found := (f(y) = 0) od
```

If f has (again) only a positive zero assume that:

1. R2 proceeds up to the while body and is preempted by R1
2. R1 computes till it finds a zero
3. R2 gets the CPU back

When R2 restarts it executes the while body and sets found to false - found := (f(y) = 0). The program would not terminate because it would look for a non existing negative zero.

Attempt 2 ctd

The problem with the second attempt is due to the fact that found is (unnecessarily) set to false (via found := f(y) = 0) after it has already got the value true.

LESSON 2

No assumption can be made on the relative speed of processes.

Attempt 3

Let us see what happens if we do not perform "unnecessary" assignments and only assign true when we find a x or a y such that f(x) = 0 or f(y) = 0.

```
found := false; (T1 || T2) where
 T1 = x := 0; while not found
          do x:= x+1; if f(x) = 0 then found := true fi od
 T2 = y := 0; while not found
          do y:= y-1; if f(y) = 0 then found := true fi od
```

Attempt 3

Let us see what happens if we do not perform "unnecessary" assignments and only assign true when we find a x or a y such that f(x) = 0 or f(y) = 0.

```
found := false; (T1 || T2) where
 T1 = x := 0; while not found
          do x:= x+1; if f(x) = 0 then found := true fi od
 T2 = y := 0; while not found
          do y:= y-1; if f(y) = 0 then found := true fi od
```

However . . .

. . . if f has only a positive zero and that T2 gets the CPU and is scheduled to keep it until its termination; T1 will never get the chance to find its zero.

Attempt 3 - ctd

This problem is due to the considered scheduler of the CPU, to avoid problems we would need a *non fair* scheduler; but this is a too strong assumptions.

LESSON 3

No assumption can be made on the cpu scheduling policy chosen by the operating system.

Attempt 4

To avoid assumptions on the scheduler, we could think of adding control to the programs and let them "pass the baton" once they have got their "chance" to execute for a while.

```
turn:= 1; found := false; (P1 || P2) where
 P1 = x := 0; while not found do wait turn:= 1 then
      turn:= 2; x:= x+1; if f(x) = 0 then found := true fi od
 P2 = y := 0; while not found do wait turn:= 2 then
      turn:= 1; y:= y-1; if f(y) = 0 then found := true fi od
```

Attempt 4

To avoid assumptions on the scheduler, we could think of adding control to the programs and let them "pass the baton" once they have got their "chance" to execute for a while.

```
turn:= 1; found := false; (P1 || P2) where
 P1 = x := 0; while not found do wait turn:= 1 then
      turn:= 2; x:= x+1; if f(x) = 0 then found := true fi od
 P2 = y := 0; while not found do wait turn:= 2 then
      turn:= 1; y:= y-1; if f(y) = 0 then found := true fi od
```

However ...

...if P1 finds a zero and stops when P2 has already set turn:= 1, P2 would be blocked by the wait command because nobody can change the value of turn.

Attempt 4 - ctd.

The problem here is that one of the program does not terminate because it keeps waiting for an impossible event.

LESSON 4

When terminating processes should care of other processes counting on them.

A CORRECT Solution!

Idea ...

...pass (again) the baton just before terminating.

```
turn:= 1; found := false; (P1; turn:= 2  || P2; turn:= 1)

where
P1 = x := 0; while not found do
                        wait turn:= 1 then
                            turn:= 2; x:= x+1;
                            if f(x) = 0 then found := true fi
                                                        od
P2 = y := 0; while not found do
                        wait turn:= 2 then
                            turn:= 1; y:= y-1;
                            if f(y) = 0 then found := true fi
                                                        od
```

Sequential Programming vs Concurrent Programming

Classical Sequential Programming

1. Denotational semantics: the meaning of a program is a partial function from states to states

Sequential Programming vs Concurrent Programming

Classical Sequential Programming

1. Denotational semantics: the meaning of a program is a partial function from states to states
2. Nontermination is bad!

Sequential Programming vs Concurrent Programming

Classical Sequential Programming

1. Denotational semantics: the meaning of a program is a partial function from states to states
2. Nontermination is bad!
3. In case of termination, the result is unique.

Sequential Programming vs Concurrent Programming

Classical Sequential Programming

1. Denotational semantics: the meaning of a program is a partial function from states to states
2. Nontermination is bad!
3. In case of termination, the result is unique.

Concurrent - Interactive - Reactive Programming

1. Denotational semantics is very complicate due to nondeterminism

Sequential Programming vs Concurrent Programming

Classical Sequential Programming

1. Denotational semantics: the meaning of a program is a partial function from states to states
2. Nontermination is bad!
3. In case of termination, the result is unique.

Concurrent - Interactive - Reactive Programming

1. Denotational semantics is very complicate due to nondeterminism
2. Nontermination might be good!

Sequential Programming vs Concurrent Programming

Classical Sequential Programming

1. Denotational semantics: the meaning of a program is a partial function from states to states
2. Nontermination is bad!
3. In case of termination, the result is unique.

Concurrent - Interactive - Reactive Programming

1. Denotational semantics is very complicate due to nondeterminism
2. Nontermination might be good!
3. In case of termination, the result might not be unique.

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols
- Mobile phones

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols
- Mobile phones
- Vending machines

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols
- Mobile phones
- Vending machines

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols
- Mobile phones
- Vending machines

The above systems compute by reacting to stimuli from their environment and are known as Reactive Systems. Their distinguishing features are:

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols
- Mobile phones
- Vending machines

The above systems compute by reacting to stimuli from their environment and are known as Reactive Systems. Their distinguishing features are:

- Interaction (many parallel communicating processes)

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols
- Mobile phones
- Vending machines

The above systems compute by reacting to stimuli from their environment and are known as Reactive Systems. Their distinguishing features are:

- Interaction (many parallel communicating processes)
- Nondeterminism (results are not necessarily unique)

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols
- Mobile phones
- Vending machines

The above systems compute by reacting to stimuli from their environment and are known as Reactive Systems. Their distinguishing features are:

- Interaction (many parallel communicating processes)
- Nondeterminism (results are not necessarily unique)
- There may be no visible result (exchange of messages is used to coordinate progress)

Programming Reactive System

The classical denotational approach is not sufficient for modelling systems such as:

- Operating systems
- Communication protocols
- Mobile phones
- Vending machines

The above systems compute by reacting to stimuli from their environment and are known as Reactive Systems. Their distinguishing features are:

- Interaction (many parallel communicating processes)
- Nondeterminism (results are not necessarily unique)
- There may be no visible result (exchange of messages is used to coordinate progress)
- Nontermination is good (systems are expected to run continuously)

Analisys of Reactive Systems

Even short parallel programs may be hard to analyze, thus we need to face few questions:

Analisys of Reactive Systems

Even short parallel programs may be hard to analyze, thus we need to face few questions:

1. How can we develop (design) a system that works?

Analisys of Reactive Systems

Even short parallel programs may be hard to analyze, thus we need to face few questions:

1. How can we develop (design) a system that works?
2. How do we analyze (verify) such a system?

Analisys of Reactive Systems

Even short parallel programs may be hard to analyze, thus we need to face few questions:

1. How can we develop (design) a system that works?
2. How do we analyze (verify) such a system?

We need appropriate theories and formal methods and tools, otherwise we will experience again:

1. Intels Pentium-II bug in floating-point division unit

Analisys of Reactive Systems

Even short parallel programs may be hard to analyze, thus we need to face few questions:

1. How can we develop (design) a system that works?
2. How do we analyze (verify) such a system?

We need appropriate theories and formal methods and tools, otherwise we will experience again:

1. Intels Pentium-II bug in floating-point division unit
2. Ariane-5 crash due to a conversion of 64-bit real to 16-bit integer

Analisys of Reactive Systems

Even short parallel programs may be hard to analyze, thus we need to face few questions:

1. How can we develop (design) a system that works?
2. How do we analyze (verify) such a system?

We need appropriate theories and formal methods and tools, otherwise we will experience again:

1. Intels Pentium-II bug in floating-point division unit
2. Ariane-5 crash due to a conversion of 64-bit real to 16-bit integer
3. Mars Pathfinder problems

Formal Methods for Reactive Systems

To deal with reactive systems and guarantee their correct behavior in all possible environment, we need:

1. To study mathematical models for the formal description and analysis of concurrent programs.

Formal Methods for Reactive Systems

To deal with reactive systems and guarantee their correct behavior in all possible environment, we need:

1. To study mathematical models for the formal description and analysis of concurrent programs.
2. To devise formal languages for the specification of the possible behaviour of parallel and reactive systems.

Formal Methods for Reactive Systems

To deal with reactive systems and guarantee their correct behavior in all possible environment, we need:

1. To study mathematical models for the formal description and analysis of concurrent programs.
2. To devise formal languages for the specification of the possible behaviour of parallel and reactive systems.
3. To develop verification tools and implementation techniques underlying them.

This School

In this school you shall see different theories of special kind of reactive systems (Global Computers) and their applications.

This School

In this school you shall see different theories of special kind of reactive systems (Global Computers) and their applications.

The theories aim at supporting: Design, Specification and Verification (possibly automatic and compositional) of reactive (global) systems.

This School

In this school you shall see different theories of special kind of reactive systems (Global Computers) and their applications.

The theories aim at supporting: Design, Specification and Verification (possibly automatic and compositional) of reactive (global) systems.

Important Questions:

- What is the most abstract view of a reactive system (process)?
- Does it capture their relevant properties?
- Is it compositional?

This two lectures:

- The chosen abstraction for reactive systems is the notion of processes.
- Systems evolution is based on process transformation: A process performs an action and becomes another process.
- Everything is (or can be viewed as) a process. Buffers, shared memory, Linda tuple spaces, senders, receivers, . . . are all processes.
- Labelled Transition Systems (LTS) describe process behaviour, and permit modelling directly systems interaction.

Presentations of Labelled Transition Systems

Process Algebra as denotations of LTS

- LTS are represented by terms of process algebras.
- Terms are interpreted via operational semantics as LTS.

Process Algebra Basic Principles

1. Define a few elementary (atomic) processes modelling the simplest process behaviour;
2. Define appropriate composition operations to build more complex process behaviour from (existing) simpler ones.

Outline of the two lectures

1. Labelled Transition Systems as Concurrency Models

Outline of the two lectures

1. Labelled Transition Systems as Concurrency Models
2. Operators for Interaction, Nondeterminism and Concurrency

Outline of the two lectures

1. Labelled Transition Systems as Concurrency Models
2. Operators for Interaction, Nondeterminism and Concurrency
3. Process Calculi and their semantics

Outline of the two lectures

1. Labelled Transition Systems as Concurrency Models
2. Operators for Interaction, Nondeterminism and Concurrency
3. Process Calculi and their semantics
4. π-calculus and Klaim (if time permits)

Models for Concurrent Processes

Operational Semantics for Concurrent Processes

Systems behaviour is described by associating to each program a behaviour represented as a transition graph.

Two main models (or variants thereof) have been used:

Operational Semantics for Concurrent Processes

Systems behaviour is described by associating to each program a behaviour represented as a transition graph.

Two main models (or variants thereof) have been used:

Kripke Structures

State Labelled Graphs: States are labelled with the properties that are considered relevant (e.g. the value of - the relation between - some variables)

Operational Semantics for Concurrent Processes

Systems behaviour is described by associating to each program a behaviour represented as a transition graph.

Two main models (or variants thereof) have been used:

Kripke Structures

State Labelled Graphs: States are labelled with the properties that are considered relevant (e.g. the value of - the relation between - some variables)

Labelled Transition Systems

Transition Labelled Graph: Transition between states are labelled the action that induces the transition from one state to another.

Operational Semantics for Concurrent Processes

Systems behaviour is described by associating to each program a behaviour represented as a transition graph.

Two main models (or variants thereof) have been used:

Kripke Structures

State Labelled Graphs: States are labelled with the properties that are considered relevant (e.g. the value of - the relation between - some variables)

Labelled Transition Systems

Transition Labelled Graph: Transition between states are labelled the action that induces the transition from one state to another.

In this lectures, we shall mainly rely on **Labelled Transition Systems** and actions will play an important role

Finite State Automata

Definition

A *finite state automaton* M is a 5-tuple

$M = (Q, A, \rightarrow, q_0, F)$ *where*

- Q is a finite set of states
- A is the alphabet
- $\rightarrow \subseteq Q \times (A \cup \{\varepsilon\}) \times Q$ is the transition relation
- $q_0 \in Q$ is a special state called initial state,
- $F \subseteq Q$ is the set of (final states)

Finite State Automata

Definition

A *finite state automaton* M is a 5-tuple

$M = (Q, A, \rightarrow, q_0, F)$ *where*

- Q is a finite set of states
- A is the alphabet
- $\rightarrow \subseteq Q \times (A \cup \{\varepsilon\}) \times Q$ is the transition relation
- $q_0 \in Q$ is a special state called initial state,
- $F \subseteq Q$ is the set of (final states)

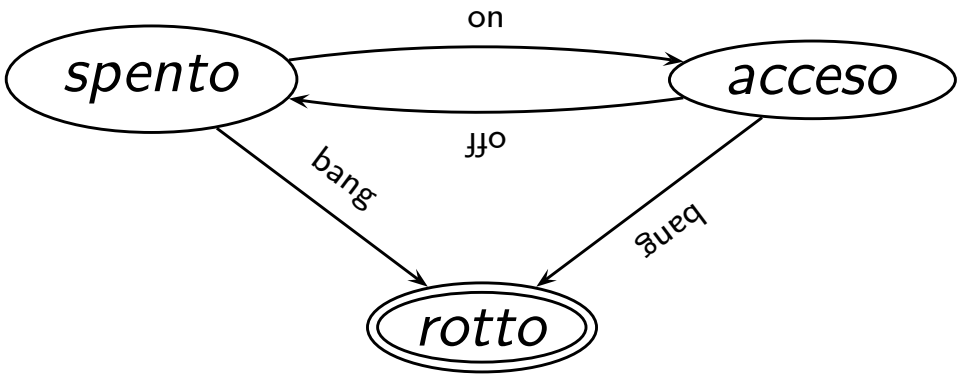


Figure: Finite state automaton

Labelled Transition Systems

Definition

A Labelled Transition System S is a 4-tuple $S = (Q, A, \rightarrow, q_0)$ *where* :

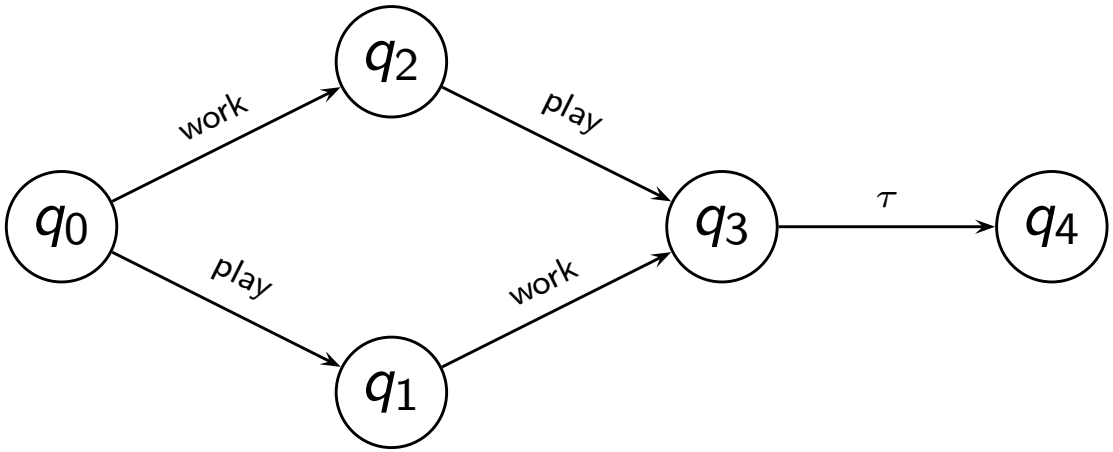
- Q is a set of states
- A is a finite set of actions
- $\rightarrow \subseteq Q \times A \times Q$ is a ternary relation called transition relation it is often written $q \xrightarrow{a} q'$ instead of $(q, a, q') \in \rightarrow$
- $q_0 \in Q$ is a special state called initial state.

Labelled Transition Systems

Definition

A Labelled Transition System S is a 4-tuple $S = (Q, A, \rightarrow, q_0)$ *where* :

- Q is a set of states
- A is a finite set of actions
- $\rightarrow \subseteq Q \times A \times Q$ is a ternary relation called transition relation it is often written $q \xrightarrow{a} q'$ instead of $(q, a, q') \in \rightarrow$
- $q_0 \in Q$ is a special state called initial state.

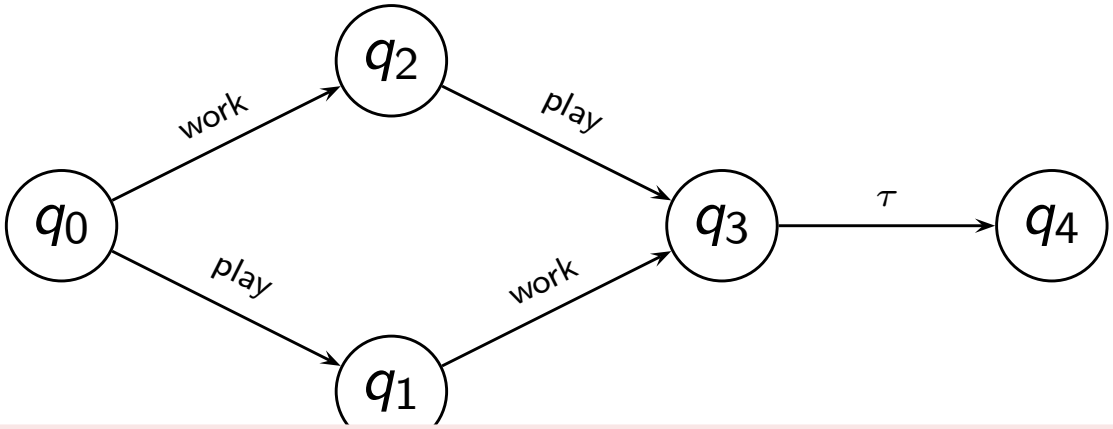


Labelled Transition Systems

Definition

A Labelled Transition System S is a 4-tuple $S = (Q, A, \rightarrow, q_0)$ *where* :

- Q is a set of states
- A is a finite set of actions
- $\rightarrow \subseteq Q \times A \times Q$ is a ternary relation called transition relation it is often written $q \xrightarrow{a} q'$ instead of $(q, a, q') \in \rightarrow$
- $q_0 \in Q$ is a special state called initial state.



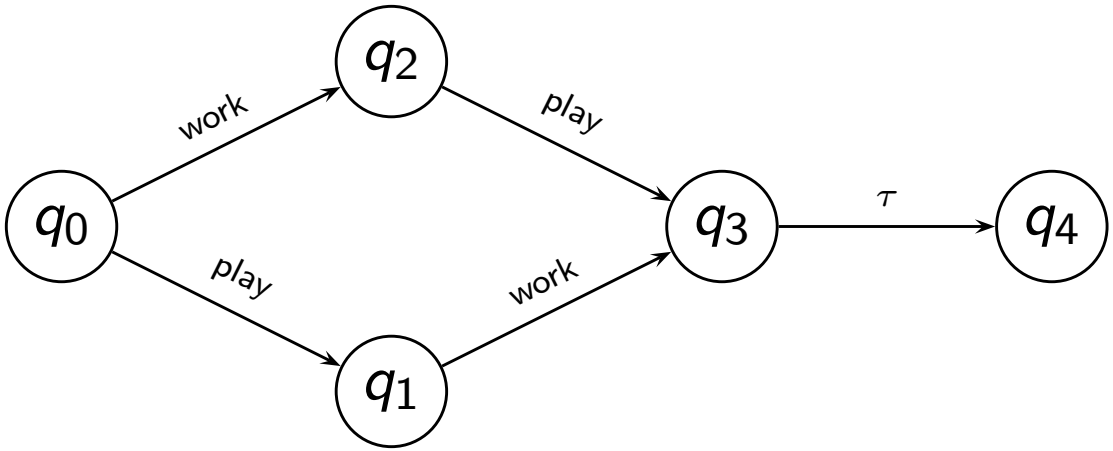
If initial states are not relevant (or known) LTSs are triples $(Q, A, \rightarrow)$ ...

A Simple Example

Example (Bill-Ben)

$S = (Q, A, \rightarrow)$ where:

- $Q = \{\ q_0,\ q_1,\ q_2,\ q_3,\ q_4\ \}$
- $A = \{\ play,\ work,\ \tau\ \}$
- $\rightarrow =$
  $\{(q_0, play, q_1),\ (q_0, work, q_2),\ (q_1, work, q_3),\ (q_2, play, q_3),\ (q_3, \tau, q_4)\}$



Internal and External Actions

An elementary action of a system represents the atomic (non-interruptible) abstract step of a computation that is performed by a system to move from one state to the other.

Actions represent various activities of concurrent systems:

1. Sending a message

Internal and External Actions

An elementary action of a system represents the atomic (non-interruptible) abstract step of a computation that is performed by a system to move from one state to the other.

Actions represent various activities of concurrent systems:

1. Sending a message
2. Receiving a message

Internal and External Actions

An elementary action of a system represents the atomic (non-interruptible) abstract step of a computation that is performed by a system to move from one state to the other.

Actions represent various activities of concurrent systems:

1. Sending a message
2. Receiving a message
3. Updating values

Internal and External Actions

An elementary action of a system represents the atomic (non-interruptible) abstract step of a computation that is performed by a system to move from one state to the other.

Actions represent various activities of concurrent systems:

1. Sending a message
2. Receiving a message
3. Updating values
4. Synchronizing with other processes

Internal and External Actions

An elementary action of a system represents the atomic (non-interruptible) abstract step of a computation that is performed by a system to move from one state to the other.

Actions represent various activities of concurrent systems:

1. Sending a message
2. Receiving a message
3. Updating values
4. Synchronizing with other processes
5. ...

Internal and External Actions

An elementary action of a system represents the atomic (non-interruptible) abstract step of a computation that is performed by a system to move from one state to the other.

Actions represent various activities of concurrent systems:

1. Sending a message
2. Receiving a message
3. Updating values
4. Synchronizing with other processes
5. ...

We have two main types of atomic actions:

Internal and External Actions

An elementary action of a system represents the atomic (non-interruptible) abstract step of a computation that is performed by a system to move from one state to the other.

Actions represent various activities of concurrent systems:

1. Sending a message
2. Receiving a message
3. Updating values
4. Synchronizing with other processes
5. ...

We have two main types of atomic actions:

- Visible Actions

Internal and External Actions

An elementary action of a system represents the atomic (non-interruptible) abstract step of a computation that is performed by a system to move from one state to the other.

Actions represent various activities of concurrent systems:

1. Sending a message
2. Receiving a message
3. Updating values
4. Synchronizing with other processes
5. ...

We have two main types of atomic actions:

- Visible Actions
- Internal Actions

Operators for Concurrency and Process Algebras

Operators for Processes Modelling

Processes are composed via a number of basic operators

1. Basic Processes
2. Action Prefixing
3. Sequentialization
4. Choice
5. Parallel Composition
6. Abstraction
7. Infinite Behaviours

Regular Expressions as Process Algebras

Syntax of Regular Expressions

$E ::= 0 \mid 1 \mid a \mid E + E \mid E; E \mid E^*$ *with* $a \in A$ the set of basic actions

Denotational Semantics of Regular Expressions

Regular Expression have a denotational semantics that associates to each expression the language (i.e. the set of strings) generated by it.

$$\mathcal{L}[\![0]\!] = \{\,\}$$

$$\mathcal{L}[\![1]\!] = \{\varepsilon\}$$

$$\mathcal{L}[\![a]\!] = \{a\} \quad (\text{per } a \in A)$$

$$\mathcal{L}[\![e + f]\!] = \mathcal{L}[\![e]\!] \cup \mathcal{L}[\![f]\!]$$

$$\mathcal{L}[\![e \cdot f]\!] = \mathcal{L}[\![e]\!] \cdot \mathcal{L}[\![f]\!]$$

$$\mathcal{L}[\![e^*]\!] = (\mathcal{L}[\![e]\!])^*$$

Regular Expressions as Process Algebras

Syntax of Regular Expressions

$$E ::= 0 \mid 1 \mid a \mid E + E \mid E; E \mid E^* \text{ with } a \in A \text{ and } -\text{below}- \ \mu \in A \cup \{\varepsilon\}$$

Operational Semantics of Regular Expressions

$$\text{(Tic)} \quad \frac{}{1 \xrightarrow{\varepsilon} 1} \qquad \text{(Atom)} \quad \frac{}{a \xrightarrow{a} 1}$$

$$\text{(Sum}_1\text{)} \quad \frac{e \xrightarrow{\mu} e'}{e + f \xrightarrow{\mu} e'} \qquad \text{(Sum}_2\text{)} \quad \frac{f \xrightarrow{\mu} f'}{e + f \xrightarrow{\mu} f'}$$

$$\text{(Seq}_1\text{)} \quad \frac{e \xrightarrow{\mu} e'}{e; f \xrightarrow{\mu} e'; f} \qquad \text{(Seq}_2\text{)} \quad \frac{e \xrightarrow{\varepsilon} 1}{e; f \xrightarrow{\varepsilon} f}$$

$$\text{(Star}_1\text{)} \quad \frac{}{e^* \xrightarrow{\varepsilon} 1} \qquad \text{(Star}_2\text{)} \quad \frac{e \xrightarrow{\mu} e'}{e^* \xrightarrow{\mu} e'; e^*}$$

Why operators for describing systems

How can we describe very large automata or LTSs?

Why operators for describing systems

How can we describe very large automata or LTSs?

As a table?

Rows and columns are labelled by states, entries are either empty or marked with a set of actions.

Why operators for describing systems

How can we describe very large automata or LTSs?

As a table?

Rows and columns are labelled by states, entries are either empty or marked with a set of actions.

As a listing of triples?

$\rightarrow = \{(q_0, a, q_1),\ (q_0, a, q_2),\ (q_1, b, q_3),\ (q_1, c, q_4),\ (q_2, \tau, q_3),\ (q_2, \tau, q_4)\}$.

Why operators for describing systems

How can we describe very large automata or LTSs?

As a table?

Rows and columns are labelled by states, entries are either empty or marked with a set of actions.

As a listing of triples?

$$\rightarrow = \{(q_0, a, q_1),\ (q_0, a, q_2),\ (q_1, b, q_3),\ (q_1, c, q_4),\ (q_2, \tau, q_3),\ (q_2, \tau, q_4)\}.$$

As a more compact listing of triples?

$$\rightarrow = \{(q_0, a, \{q_1, q_2\}),\ (q_1, b, q_3),\ (q_1, c, q_4),\ (q_2, \tau, \{q_3, q_4\})\}.$$

Why operators for describing systems

How can we describe very large automata or LTSs?

As a table?

Rows and columns are labelled by states, entries are either empty or marked with a set of actions.

As a listing of triples?

$\rightarrow = \{(q_0, a, q_1),\ (q_0, a, q_2),\ (q_1, b, q_3),\ (q_1, c, q_4),\ (q_2, \tau, q_3),\ (q_2, \tau, q_4)\}$.

As a more compact listing of triples?

$\rightarrow = \{(q_0, a, \{q_1, q_2\}),\ (q_1, b, q_3),\ (q_1, c, q_4),\ (q_2, \tau, \{q_3, q_4\})\}$.

As XML?

`<lts><ar><st>q0</st><lab>a</lab><st>q1</st></ar>...</lts>`.

Why operators for describing systems - ctd

Linguistic aspects are important!

The previous solutions are ok for machines . . . not for humans.

Why operators for describing systems - ctd

Linguistic aspects are important!

The previous solutions are ok for machines . . . not for humans.

Are prefix and sum operators sufficient?

They are ok to describe small finite systems:

- $p = a.b.(c + d)$
- $q = a.(b.c + b.d)$
- $r = a.b.c + a.c.d$

Why operators for describing systems - ctd

Linguistic aspects are important!

The previous solutions are ok for machines . . . not for humans.

Are prefix and sum operators sufficient?

They are ok to describe small finite systems:

- $p = a.b.(c + d)$
- $q = a.(b.c + b.d)$
- $r = a.b.c + a.c.d$

But additional operators are needed

- to design systems in a structured way (e.g. $p|q$)
- to model systems interaction
- to abstract from details
- to represent infinite systems

Operational Semantics

To each process, built using the above mentioned operators, an LTS is associated by relying on structural induction to define the meaning of each operator.

Inference Systems

An inference system is a set of inference rule of the form

$$\frac{p_1, \cdots, p_n}{q}$$

Transition Rules

For each operator op, we have a number of rules of the form below, where $\{i_1, \cdots, i_m\} \subseteq \{1, \cdots, n\}$.

$$\frac{E_{i_1} \xrightarrow{\alpha_1} E'_{i_1} \quad \cdots \quad E_{i_m} \xrightarrow{\alpha_m} E'_{i_m}}{op(E_1, \cdots, E_n) \xrightarrow{\alpha} op(E'_1, \cdots, E'_n)}$$

The Elegance of Operational Semantics

Automata as terms

Few SOS rules define all the automata that can ever be specified with the chosen operators. Given any term, the rules are used the derive the corresponding automaton. The set of rules is fixed once and for all.

The Elegance of Operational Semantics

Automata as terms

Few SOS rules define all the automata that can ever be specified with the chosen operators. Given any term, the rules are used the derive the corresponding automaton. The set of rules is fixed once and for all.

Structural induction

The interaction of complex systems is defined in terms of the behavior of their components.

The Elegance of Operational Semantics

Automata as terms

Few SOS rules define all the automata that can ever be specified with the chosen operators. Given any term, the rules are used the derive the corresponding automaton. The set of rules is fixed once and for all.

Structural induction

The interaction of complex systems is defined in terms of the behavior of their components.

A remark

The LTS is the least one satisfying the inference rules.

The Elegance of Operational Semantics

Automata as terms

Few SOS rules define all the automata that can ever be specified with the chosen operators. Given any term, the rules are used the derive the corresponding automaton. The set of rules is fixed once and for all.

Structural induction

The interaction of complex systems is defined in terms of the behavior of their components.

A remark

The LTS is the least one satisfying the inference rules.

Rule induction

A property is true for the whole LTS if whenever it holds for the premises of each rule, it holds also for the conclusion.

A few examples for Regular Expressions

$(a+b)^* \xrightarrow{a} 1;(a+b)^*$

$$\dfrac{\dfrac{\dfrac{}{a \xrightarrow{a} 1}\,(Atom)}{a+b \xrightarrow{a} 1}\,(Sum_1)}{(a+b)^* \xrightarrow{a} 1;(a+b)^*}\,(Star_2)$$

A few examples for Regular Expressions

$(a+b)^* \xrightarrow{a} 1;(a+b)^*$

$$\dfrac{\dfrac{\dfrac{}{a \xrightarrow{a} 1}\,(Atom)}{a+b \xrightarrow{a} 1}\,(Sum_1)}{(a+b)^* \xrightarrow{a} 1;(a+b)^*}\,(Star_2)$$

$1;(a+b)^* \xrightarrow{\varepsilon} (a+b)^*$

$$\dfrac{\dfrac{}{1 \xrightarrow{\varepsilon} 1}\,(Tic)}{1;(a+b)^* \xrightarrow{\varepsilon} (a+b)^*}\,(Seq_2)$$

Another Example On Regular Expressions

$(a^* + b^*)^* \xrightarrow{b} 1; b^*; (a^* + b^*)^*$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{b \xrightarrow{b} 1}(Atom)}{b^* \xrightarrow{b} 1; b^*}(Star_2)}{a^* + b^* \xrightarrow{b} 1; b^*}(Sum_2)}{(a^* + b^*)^* \xrightarrow{b} 1; b^*; (a^* + b^*)^*}(Star_2)$$

Another Example On Regular Expressions

$(a^* + b^*)^* \xrightarrow{b} 1; b^*; (a^* + b^*)^*$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{b \xrightarrow{b} 1}(Atom)}{b^* \xrightarrow{b} 1; b^*}(Star_2)}{a^* + b^* \xrightarrow{b} 1; b^*}(Sum_2)}{(a^* + b^*)^* \xrightarrow{b} 1; b^*; (a^* + b^*)^*}(Star_2)$$

A remark

$(a^* + b^*)^* \xrightarrow{c} 1$ would not contradict any rule, but it cannot be in the least LTS, because it cannot be inferred by using the rules we presented earlier.

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?
3. Proof by majority: When general agreement fails

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?
3. Proof by majority: When general agreement fails
4. Proof by plausibility: It sounds good

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?
3. Proof by majority: When general agreement fails
4. Proof by plausibility: It sounds good
5. Proof by intimidation: Trivial!

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?
3. Proof by majority: When general agreement fails
4. Proof by plausibility: It sounds good
5. Proof by intimidation: Trivial!
6. Proof by lost reference: I saw it somewhere

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?
3. Proof by majority: When general agreement fails
4. Proof by plausibility: It sounds good
5. Proof by intimidation: Trivial!
6. Proof by lost reference: I saw it somewhere
7. Proof by obscure reference: It appeared in the Annals of Polish Math. Soc. (1854, in polish)

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?
3. Proof by majority: When general agreement fails
4. Proof by plausibility: It sounds good
5. Proof by intimidation: Trivial!
6. Proof by lost reference: I saw it somewhere
7. Proof by obscure reference: It appeared in the Annals of Polish Math. Soc. (1854, in polish)
8. Proof by authority: Don Knuth said it was true

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?
3. Proof by majority: When general agreement fails
4. Proof by plausibility: It sounds good
5. Proof by intimidation: Trivial!
6. Proof by lost reference: I saw it somewhere
7. Proof by obscure reference: It appeared in the Annals of Polish Math. Soc. (1854, in polish)
8. Proof by authority: Don Knuth said it was true
9. Proof by intuition: I have this feeling...

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?
3. Proof by majority: When general agreement fails
4. Proof by plausibility: It sounds good
5. Proof by intimidation: Trivial!
6. Proof by lost reference: I saw it somewhere
7. Proof by obscure reference: It appeared in the Annals of Polish Math. Soc. (1854, in polish)
8. Proof by authority: Don Knuth said it was true
9. Proof by intuition: I have this feeling...
10. Proof by deception: Everybody please turn their backs...

Playful digression

Some advanced proof methods

1. Proof by obviousness: So evident it need not to be mentioned
2. Proof by general agreement: All in favor?
3. Proof by majority: When general agreement fails
4. Proof by plausibility: It sounds good
5. Proof by intimidation: Trivial!
6. Proof by lost reference: I saw it somewhere
7. Proof by obscure reference: It appeared in the Annals of Polish Math. Soc. (1854, in polish)
8. Proof by authority: Don Knuth said it was true
9. Proof by intuition: I have this feeling...
10. Proof by deception: Everybody please turn their backs...
11. Proof by logic: It is on the textbook, hence it must be true

Basic Processes

Inactive Process

Is usually denoted by

- *nil*
- 0
- *stop*

The semantics of this process is characterized by the fact that there is no rule to define its transition: it has no transition.

Basic Processes

Inactive Process

Is usually denoted by

- *nil*
- 0
- *stop*

The semantics of this process is characterized by the fact that there is no rule to define its transition: it has no transition.

A broken vending machine

$$nil$$

Does not accept coins and does not give any drink.

Basic Processes ctd

Termination

Termination is sometimes denoted by

- *exit*
- *skip*

that can only perform the special action $\surd$ ("tick") to indicate termination and become *nil*

$$\frac{}{exit \xrightarrow{\surd} stop}$$

Basic Processes ctd

Termination

Termination is sometimes denoted by

- *exit*
- *skip*

that can only perform the special action $\surd$ ("tick") to indicate termination and become *nil*

$$\frac{}{exit \xrightarrow{\surd} stop}$$

A gentle broken vending machine

$$exit$$

Does not accept coins, does not gives drinks but says that everything is ok.

Action Prefixing

Prefixing

For each action μ there is a unary operator

- $\mu.\cdot$
- $\mu \longrightarrow \cdot$

that builds from process E a new process $\mu.E$ that performs action μ and then behaves like E.

$$\frac{}{\mu.E \xrightarrow{\mu} E}$$

Action Prefixing

Prefixing

For each action μ there is a unary operator

- $\mu.\cdot$
- $\mu \rightarrow \cdot$

that builds from process E a new process $\mu.E$ that performs action μ and then behaves like E.

$$\frac{}{\mu.E \xrightarrow{\mu} E}$$

A "one shot" vending machine

$$coin \rightarrow choc \rightarrow stop$$

Accepts a coin and gives a chocolate, then stops.

Action Prefixing ctd

Action as processes

Instead of prefixing, some calculi rely on considering actions as basic processes.

$$\frac{}{a \xrightarrow{a} stop}$$

Action Prefixing ctd

Action as processes

Instead of prefixing, some calculi rely on considering actions as basic processes.

$$\frac{}{a \xrightarrow{a} stop}$$

A dishonest vending machine

$$coin$$

Accepts a coin and stops.

Sequential Composition

Sequentialization

The binary operator for sequential composition is denoted by

- $\cdot ; \cdot$
- $\cdot \gg \cdot$

If E ed F are processes, process $E; F$ executes E and then behaves like F

$$\frac{E \xrightarrow{\mu} E'}{E; F \xrightarrow{\mu} E'; F} \quad (\mu \neq \surd) \qquad \frac{E \xrightarrow{\surd} E'}{E; F \xrightarrow{\tau} F}$$

Sequential Composition

Sequentialization

The binary operator for sequential composition is denoted by

- $\cdot ; \cdot$
- $\cdot \gg \cdot$

If E ed F are processes, process $E; F$ executes E and then behaves like F

$$\frac{E \xrightarrow{\mu} E'}{E; F \xrightarrow{\mu} E'; F} \quad (\mu \neq \surd) \qquad \frac{E \xrightarrow{\surd} E'}{E; F \xrightarrow{\tau} F}$$

Another "one shot" vending machine

$$coin;\ choc$$

Sequential Composition ctd

Disabling Operator

The disabling binary operator

- $[>$

permits to interrupt some actions when specific events happen.

$$\frac{E \xrightarrow{\mu} E'}{E \,[> F \xrightarrow{\mu} E' \,[> F} \quad (\mu \neq \surd) \qquad \frac{E \xrightarrow{\surd} E'}{E \,[> F \xrightarrow{\tau} E'} \qquad \frac{F \xrightarrow{\mu} F'}{E \,[> F \xrightarrow{\mu} F'}$$

Sequential Composition ctd

Disabling Operator

The disabling binary operator

- $[>$

permits to interrupt some actions when specific events happen.

$$\frac{E \xrightarrow{\mu} E'}{E \,[> F \xrightarrow{\mu} E' \,[> F} \quad (\mu \neq \surd) \qquad \frac{E \xrightarrow{\surd} E'}{E \,[> F \xrightarrow{\tau} E'} \qquad \frac{F \xrightarrow{\mu} F'}{E \,[> F \xrightarrow{\mu} F'}$$

A cheating customer

$$(coin \rightarrow choc \rightarrow stop) \;[> \; (bang \rightarrow choc \rightarrow stop)$$

This describes a vending machine that when "banged" gives away a chocolate without getting the coin

Choice - 1

Nondeterministic Choice

$$\frac{E \xrightarrow{\mu} E'}{E + F \xrightarrow{\mu} E'} \qquad \frac{F \xrightarrow{\mu} F'}{E + F \xrightarrow{\mu} F'}$$

Choice - 1

Nondeterministic Choice

$$\frac{E \xrightarrow{\mu} E'}{E + F \xrightarrow{\mu} E'} \qquad \frac{F \xrightarrow{\mu} F'}{E + F \xrightarrow{\mu} F'}$$

User's Choice

$$coin \longrightarrow (choc \longrightarrow stop \ + \ water \longrightarrow stop)$$

Choice - 1

Nondeterministic Choice

$$\frac{E \xrightarrow{\mu} E'}{E + F \xrightarrow{\mu} E'} \qquad \frac{F \xrightarrow{\mu} F'}{E + F \xrightarrow{\mu} F'}$$

User's Choice

$$coin \longrightarrow (choc \longrightarrow stop \ + \ water \longrightarrow stop)$$

Machine's Choice

$$coin \longrightarrow choc \longrightarrow stop \ + \ coin \longrightarrow water \longrightarrow stop$$

Choice - 2

Internal Choice

$$\frac{}{E \oplus F \xrightarrow{\tau} E} \qquad \frac{}{E \oplus F \xrightarrow{\tau} F}$$

Choice - 2

Internal Choice

$$\frac{}{E \oplus F \xrightarrow{\tau} E} \qquad \frac{}{E \oplus F \xrightarrow{\tau} F}$$

Machine's Choice

$$coin \longrightarrow (choc \longrightarrow stop \ \oplus \ water \longrightarrow stop)$$

Choice - 3

External Choice

$$\frac{E \xrightarrow{\alpha} E'}{E \,\Box\, F \xrightarrow{\alpha} E'}\,(\alpha \neq \tau) \qquad \frac{F \xrightarrow{\alpha} F'}{E \,\Box\, F \xrightarrow{\alpha} F'}\,(\alpha \neq \tau)$$

$$\frac{E \xrightarrow{\tau} E'}{E \,\Box\, F \xrightarrow{\tau} E' \,\Box\, F} \qquad \frac{F \xrightarrow{\tau} F'}{E \,\Box\, F \xrightarrow{\tau} E \,\Box\, F'}$$

Choice - 3

External Choice

$$\frac{E \xrightarrow{\alpha} E'}{E \,\Box\, F \xrightarrow{\alpha} E'}\,(\alpha \neq \tau) \qquad \frac{F \xrightarrow{\alpha} F'}{E \,\Box\, F \xrightarrow{\alpha} F'}\,(\alpha \neq \tau)$$

$$\frac{E \xrightarrow{\tau} E'}{E \,\Box\, F \xrightarrow{\tau} E' \,\Box\, F} \qquad \frac{F \xrightarrow{\tau} F'}{E \,\Box\, F \xrightarrow{\tau} E \,\Box\, F'}$$

User's Choice

$$coin \rightarrow \big((choc \rightarrow stop \;\oplus\; water \rightarrow stop) \;\Box\; water \rightarrow stop\big)$$

Different Transitions

External Choice

$$coin \rightarrow \big((choc \rightarrow stop \ \oplus \ water \rightarrow stop) \ \Box \ water \rightarrow stop\big)$$

$$\xrightarrow{coin}$$

$$(choc \rightarrow stop \ \oplus \ water \rightarrow stop) \ \Box \ water \rightarrow stop$$

$$\xrightarrow{\tau}$$

$$(choc \rightarrow stop \ \Box \ water \rightarrow stop)$$

Different Transitions

External Choice

$$coin \rightarrow \big((choc \rightarrow stop \ \oplus \ water \rightarrow stop) \ \Box \ water \rightarrow stop\big)$$

$$\xrightarrow{coin}$$

$$(choc \rightarrow stop \ \oplus \ water \rightarrow stop) \ \Box \ water \rightarrow stop$$

$$\xrightarrow{\tau}$$

$$(choc \rightarrow stop \ \Box \ water \rightarrow stop)$$

Internal Choice

$$coin \rightarrow \big((choc \rightarrow stop \ \oplus \ water \rightarrow stop) \ \oplus \ water \rightarrow stop\big)$$

$$\xrightarrow{coin}$$

$$(choc \rightarrow stop \ \oplus \ water \rightarrow stop) \ \oplus \ water \rightarrow stop$$

$$\xrightarrow{\tau}$$

$$choc \rightarrow stop \ \oplus \ water \rightarrow stop$$

$$\xrightarrow{\tau}$$

$$choc \rightarrow stop$$

Parallel Composition - 1

Milner's Parallel

$$\frac{E \xrightarrow{\mu} E'}{E|F \xrightarrow{\mu} E'|F} \qquad \frac{F \xrightarrow{\mu} F'}{E|F \xrightarrow{\mu} E|F'} \qquad \frac{E \xrightarrow{\alpha} E' \qquad F \xrightarrow{\overline{\alpha}} F'}{E|F \xrightarrow{\tau} E'|F'}(\alpha \neq \tau)$$

Parallel Composition - 1

Milner's Parallel

$$\frac{E \xrightarrow{\mu} E'}{E|F \xrightarrow{\mu} E'|F} \qquad \frac{F \xrightarrow{\mu} F'}{E|F \xrightarrow{\mu} E|F'} \qquad \frac{E \xrightarrow{\alpha} E' \qquad F \xrightarrow{\overline{\alpha}} F'}{E|F \xrightarrow{\tau} E'|F'}(\alpha \neq \tau)$$

User-Machine interaction

$$(coin \rightarrow (\overline{choc} \rightarrow stop \oplus \overline{water} \rightarrow stop)) \mid (\overline{coin} \rightarrow choc \rightarrow stop)$$

We can have different interactions

Appropriate Interaction

$$(coin \rightarrow (\overline{choc} \rightarrow stop \ \oplus \ \overline{water} \rightarrow stop)) \ \mid \ (\overline{coin} \rightarrow choc \rightarrow stop)$$

$$\xrightarrow{\tau}$$

$$(\overline{choc} \rightarrow stop \ \oplus \ \overline{water} \rightarrow stop) \ \mid \ (choc \rightarrow stop)$$

$$\xrightarrow{\tau}$$

$$(\overline{choc} \rightarrow stop \ ) \ \mid \ (choc \rightarrow stop)$$

$$\xrightarrow{\tau}$$

$$stop \ \mid \ stop$$

We can have different interactions

Appropriate Interaction

$$(coin \rightarrow (\overline{choc} \rightarrow stop \oplus \overline{water} \rightarrow stop)) \mid (\overline{coin} \rightarrow choc \rightarrow stop)$$

$$\xrightarrow{\tau}$$

$$(\overline{choc} \rightarrow stop \oplus \overline{water} \rightarrow stop) \mid (choc \rightarrow stop)$$

$$\xrightarrow{\tau}$$

$$(\overline{choc} \rightarrow stop\ ) \mid (choc \rightarrow stop)$$

$$\xrightarrow{\tau}$$

$$stop \mid stop$$

Inappropriate Interaction - Coin thrown away

$$(coin \rightarrow (\overline{choc} \rightarrow stop \oplus \overline{water} \rightarrow stop)) \mid (\overline{coin} \rightarrow choc \rightarrow stop)$$

$$\xrightarrow{\tau}$$

$$(\overline{choc} \rightarrow stop \oplus \overline{water} \rightarrow stop) \mid (choc \rightarrow stop)$$

$$\xrightarrow{\tau}$$

$$(\overline{water} \rightarrow stop) \mid (choc \rightarrow stop)$$

Parallel Composition - 2

Merge Operator with Synchronization Function

$$\frac{E \xrightarrow{\mu} E'}{E \parallel F \xrightarrow{\mu} E' \parallel F} \qquad \frac{F \xrightarrow{\mu} F'}{E \parallel F \xrightarrow{\mu} E \parallel F'} \qquad \frac{E \xrightarrow{a} E' \qquad F \xrightarrow{b} F'}{E \parallel F \xrightarrow{\gamma(a,b)} E' \parallel F'}$$

with $\mu \in \Lambda \cup \{\tau\}$

Parallel Composition - 2

Merge Operator with Synchronization Function

$$\frac{E \xrightarrow{\mu} E'}{E \parallel F \xrightarrow{\mu} E' \parallel F} \qquad \frac{F \xrightarrow{\mu} F'}{E \parallel F \xrightarrow{\mu} E \parallel F'} \qquad \frac{E \xrightarrow{a} E' \qquad F \xrightarrow{b} F'}{E \parallel F \xrightarrow{\gamma(a,b)} E' \parallel F'}$$

with $\mu \in \Lambda \cup \{\tau\}$

Another interaction

$$getCoin.(giveChoc.nil \ + \ giveWater.nil) \ \parallel \ putCoin.getChoc.nil$$

with $\gamma(getCoin, putCoin) = ok$ e $\gamma(giveChoc, getChoc) = ok$.

Parallel Composition - 3

Communication Merge

$$\frac{E \xrightarrow{a} E' \qquad F \xrightarrow{b} F'}{E|_c F \xrightarrow{\gamma(a,b)} E' \parallel F'}$$

Parallel Composition - 3

Communication Merge

$$\frac{E \xrightarrow{a} E' \qquad F \xrightarrow{b} F'}{E|_c F \xrightarrow{\gamma(a,b)} E' \parallel F'}$$

Left Merge

$$\frac{E \xrightarrow{\mu} E'}{E \between F \xrightarrow{\mu} E' \parallel F}$$

Parallel Composition - 3

Communication Merge

$$\frac{E \xrightarrow{a} E' \qquad F \xrightarrow{b} F'}{E|_c F \xrightarrow{\gamma(a,b)} E' \parallel F'}$$

Left Merge

$$\frac{E \xrightarrow{\mu} E'}{E \between F \xrightarrow{\mu} E' \parallel F}$$

Interleaving

$$\frac{E \xrightarrow{\mu} E'}{E \lll F \xrightarrow{\mu} E' \lll F} \qquad \frac{F \xrightarrow{\mu} F'}{E \lll F \xrightarrow{\mu} E \lll F'}$$

Parallel Composition - 4

Hoare's Parallel

$$\frac{E \xrightarrow{\mu} E'}{E \,|[L]|\, F \xrightarrow{\mu} E' \,|[L]|\, F}\ (\mu \notin L) \qquad \frac{F \xrightarrow{\mu} F'}{E \,|[L]|\, F \xrightarrow{\mu} E \,|[L]|\, F'}\ (\mu \notin L)$$

$$\frac{E \xrightarrow{a} E' \qquad F \xrightarrow{a} F'}{E \,|[L]|\, F \xrightarrow{a} E' \,|[L]|\, F'}\ (a \in L)$$

Parallel Composition - 4

Hoare's Parallel

$$\frac{E \xrightarrow{\mu} E'}{E \;|[L]|\; F \xrightarrow{\mu} E' \;|[L]|\; F}\ (\mu \notin L) \qquad \frac{F \xrightarrow{\mu} F'}{E \;|[L]|\; F \xrightarrow{\mu} E \;|[L]|\; F'}\ (\mu \notin L)$$

$$\frac{E \xrightarrow{a} E' \qquad F \xrightarrow{a} F'}{E \;|[L]|\; F \xrightarrow{a} E' \;|[L]|\; F'}\ (a \in L)$$

The operator $|[L]|$ is strongly related with some of the operators seen before.

Parallel Composition - 4

Hoare's Parallel

$$\frac{E \xrightarrow{\mu} E'}{E \;\|[L]\|\; F \xrightarrow{\mu} E' \;\|[L]\|\; F}\ (\mu \notin L) \qquad \frac{F \xrightarrow{\mu} F'}{E \;\|[L]\|\; F \xrightarrow{\mu} E \;\|[L]\|\; F'}\ (\mu \notin L)$$

$$\frac{E \xrightarrow{a} E' \qquad F \xrightarrow{a} F'}{E \;\|[L]\|\; F \xrightarrow{a} E' \;\|[L]\|\; F'}\ (a \in L)$$

The operator $\|[L]\|$ is strongly related with some of the operators seen before.

1. $\|[L]\|$ and $\|$ are equivalent if $\gamma(a, a) = a$, $\forall a \in L$,

Parallel Composition - 4

Hoare's Parallel

$$\frac{E \xrightarrow{\mu} E'}{E \;|[L]|\; F \xrightarrow{\mu} E' \;|[L]|\; F}\ (\mu \notin L) \qquad \frac{F \xrightarrow{\mu} F'}{E \;|[L]|\; F \xrightarrow{\mu} E \;|[L]|\; F'}\ (\mu \notin L)$$

$$\frac{E \xrightarrow{a} E' \qquad F \xrightarrow{a} F'}{E \;|[L]|\; F \xrightarrow{a} E' \;|[L]|\; F'}\ (a \in L)$$

The operator $|[L]|$ is strongly related with some of the operators seen before.

1. $|[L]|$ and $\|$ are equivalent if $\gamma(a, a) = a$, $\forall a \in L$,
2. $|[L]|$ and $|||$ are equivalent if $L = \emptyset$,

Interaction via Synchronization Algebra

Most operators for parallel composition can be expressed in terms of suitable synchronization algebras (assume $E \xrightarrow{*} E$ for all E).

Definition

A Synchronization Algebra una 4-tuple $\langle \Lambda, *, 0, \bullet \rangle$ where

1. Λ is a set of labels containing the special labels $*$ e 0,
2. $\bullet$ is an associative and commutative binary operation over Λ (i.e. $\bullet : \Lambda \times \Lambda \longrightarrow \Lambda$) that satisfies:
   1. $a \bullet 0 = 0$ for all $a \in \Lambda$,

Interaction via Synchronization Algebra

Most operators for parallel composition can be expressed in terms of suitable synchronization algebras (assume $E \xrightarrow{*} E$ for all E).

Definition

A Synchronization Algebra una 4-tuple $\langle \Lambda, *, 0, \bullet \rangle$ where

1. Λ is a set of labels containing the special labels $*$ e 0,
2. $\bullet$ is an associative and commutative binary operation over Λ (i.e. $\bullet : \Lambda \times \Lambda \longrightarrow \Lambda$) that satisfies:
   1. $a \bullet 0 = 0$ for all $a \in \Lambda$,
   2. $* \bullet * = *$,

Interaction via Synchronization Algebra

Most operators for parallel composition can be expressed in terms of suitable synchronization algebras (assume $E \xrightarrow{*} E$ for all E).

Definition

A Synchronization Algebra una 4-tuple $\langle \Lambda, *, 0, \bullet \rangle$ where

1. Λ is a set of labels containing the special labels $*$ e 0,
2. $\bullet$ is an associative and commutative binary operation over Λ (i.e. $\bullet : \Lambda \times \Lambda \longrightarrow \Lambda$) that satisfies:
   1. $a \bullet 0 = 0$ for all $a \in \Lambda$,
   2. $* \bullet * = *$,
   3. $a \bullet b = *$ implies $a = b = *$, for all $a, b \in \Lambda$.

Interaction via Synchronization Algebra

Most operators for parallel composition can be expressed in terms of suitable synchronization algebras (assume $E \xrightarrow{*} E$ for all E).

Definition

A Synchronization Algebra una 4-tuple $\langle \Lambda, *, 0, \bullet \rangle$ where

1. Λ is a set of labels containing the special labels $*$ e 0,
2. $\bullet$ is an associative and commutative binary operation over Λ (i.e. $\bullet : \Lambda \times \Lambda \to \Lambda$) that satisfies:
   1. $a \bullet 0 = 0$ for all $a \in \Lambda$,
   2. $* \bullet * = *$,
   3. $a \bullet b = *$ implies $a = b = *$, for all $a, b \in \Lambda$.

$$\frac{E \xrightarrow{\alpha} E' \quad F \xrightarrow{\beta} F'}{E \bullet F \xrightarrow{\alpha \bullet \beta} E' \bullet F'} \quad (\alpha \bullet \beta \neq 0)$$

$\bullet$	$*$	α	0
$*$	$*$	α	0
α	α	0	0
0	0	0	0

Interaction with Value Passing

Single Evolutions

$$\frac{}{a(x).E \xrightarrow{a(v)} E\{v/x\}}\ (v \text{ is a value}) \qquad \frac{}{\overline{a}\, e.E \xrightarrow{\overline{a}\ val(e)} E}$$

Interaction with Value Passing

Single Evolutions

$$\frac{}{a(x).E \xrightarrow{a(v)} E\{v/x\}}\ (v \text{ is a value}) \qquad \frac{}{\overline{a}\, e.E \xrightarrow{\overline{a}\ val(e)} E}$$

Interaction

$$\frac{E \xrightarrow{\overline{a}\ v} E' \quad F \xrightarrow{a(v)} F'}{E|F \xrightarrow{\tau} E'|F'} \qquad \frac{E \xrightarrow{a(v)} E' \quad F \xrightarrow{\overline{a}\ v} F'}{E|F \xrightarrow{\tau} E'|F'}$$

Conditional Execution

$$\frac{val(e) = true \quad E \xrightarrow{\mu} E'}{if\ e\ then\ E\ else\ F \xrightarrow{\mu} E'} \qquad \frac{val(e) = false \quad F \xrightarrow{\mu} F'}{if\ e\ then\ E\ else\ F \xrightarrow{\mu} F'}$$

Conditional Execution

$$\frac{val(e) = true \quad E \xrightarrow{\mu} E'}{if\ e\ then\ E\ else\ F \xrightarrow{\mu} E'} \qquad \frac{val(e) = false \quad F \xrightarrow{\mu} F'}{if\ e\ then\ E\ else\ F \xrightarrow{\mu} F'}$$

Let us consider a vending machine that accept 20 cents coins (or higher) and offers a chocolate:

$$coin(x).\ if\ x \geq 20\ then\ choc.nil\ else\ nil$$

The user interacts with the machine as follows:

$$coin(x).\ if\ x \geq 20\ then\ \overline{choc}.nil\ else\ nil \ \mid\ \overline{coin}\,40.choc.nil$$

$$\xrightarrow{\tau}$$

$$if\ 40 \geq 20\ then\ \overline{choc}.nil\ else\ nil \ \mid\ choc.nil$$

$$\xrightarrow{\tau}$$

$$nil \ \mid\ nil$$

Abstraction - 1

Restriction

$$\frac{E \xrightarrow{\alpha} E'}{E \setminus L \xrightarrow{\alpha} E' \setminus L} \quad (\alpha\ ,\ \overline{\alpha}\ \notin\ L)$$

Abstraction - 1

Restriction

$$\frac{E \xrightarrow{\alpha} E'}{E \setminus L \xrightarrow{\alpha} E' \setminus L} \ (\alpha \, , \, \overline{\alpha} \notin L)$$

Forcing Interaction

$$\begin{array}{c}
\left(\ (coin.\overline{ok}.nil) \mid ok.(\overline{choc}.nil + \overline{water}.nil) \ \right) \setminus ok \quad \mid \quad \overline{coin}.choc.nil \\
\xrightarrow{\tau} \\
\left(\ (\overline{ok}.nil) \mid ok.(\overline{choc}.nil + \overline{water}.nil) \ \right) \setminus ok \quad \mid \quad choc.nil \\
\xrightarrow{\tau} \\
\left(\ nil \mid (\overline{choc}.nil + \overline{water}.nil) \ \right) \setminus ok \quad \mid \quad choc.nil \\
\xrightarrow{\tau} \\
\left(\ nil \mid nil \ \right) \setminus ok \quad \mid \quad nil
\end{array}$$

A malicious user executing $\overline{ok}.choc.nil$ would be stopped.

Abstraction - 2

Hiding

$$\frac{E \xrightarrow{\alpha} E'}{E/L \xrightarrow{\alpha} E'/L}\,(\alpha \notin L) \qquad \frac{E \xrightarrow{\alpha} E'}{E/L \xrightarrow{\tau} E'/L}\,(\alpha \in L)$$

Avoiding Interaction

$$\left((coin.ok.nil) \;\; |[ok]| \;\; ok.(choc.nil + water.nil) \right) / ok$$

The *ok* signal is internalized thus it cannot be used by a dishonest user.

Abstraction - 3

Renaming

$$\frac{E \xrightarrow{\mu} E'}{E[f] \xrightarrow{f(\mu)} E'[f]}$$

Multilingual Interaction

An Italian user

$$\overline{soldo}.\, acqua.\, nil$$

can interact with the machine with English indication by applying:

$$(\,\overline{soldo}.\, acqua.\, nil\,)\,[coin/soldo,\ water/acqua]$$

Infinite Behaviour - 1

Recursion

$$\frac{E\{rec\, X.E/X\} \xrightarrow{\mu} E'}{rec\, X.E \xrightarrow{\mu} E'}$$

Long Lasting Vending Machine

$$rec\, D.\, coin.\, (\overline{choc}.\, D \;+\; \overline{water}.\, D)$$

$$\left.\begin{array}{l} rec\, D.\, coin.\, (\overline{choc}.\, D \;+\; \overline{water}.\, D) \end{array}\right\} \xrightarrow{coin}$$

$$\left.\begin{array}{l} \overline{choc}.\, rec\, D.\, coin.\, (\overline{choc}.\, D \;+\; \overline{water}.\, D) \\ + \\ \overline{water}.\, rec\, D.\, coin.\, (\overline{choc}.\, D \;+\; \overline{water}.\, D) \end{array}\right\} \xrightarrow{\overline{choc}}$$

$$\left.\begin{array}{l} rec\, D.\, coin.\, (\overline{choc}.\, D \;+\; \overline{water}.\, D) \end{array}\right\} \xrightarrow{coin} \ldots$$

Infinite Behaviour - 2

Replication

$$\frac{E \xrightarrow{\mu} E'}{!\,E \xrightarrow{\mu} E' \mid !\,E} \qquad \text{or, equivalently} \qquad \frac{E \mid !\,E \xrightarrow{\mu} E'}{!\,E \xrightarrow{\mu} E'}$$

The replication operator can be defined by the following equation $!\,E \triangleq E|!\,E$ that can be expressed in terms of rec as follows: $recX.(E|X)$

Chocolate ad libitum

$$!\,coin.\overline{choc}.nil \xrightarrow{coin}$$
$$\overline{choc}.nil \mid !\,coin.\overline{choc}.nil \xrightarrow{coin}$$
$$\overline{choc}.nil \mid \overline{choc}.nil \mid !\,coin.\overline{choc}.nil \xrightarrow{\overline{choc}}$$
$$nil \mid \overline{choc}.nil \mid !\,coin.\overline{choc}.nil \xrightarrow{\overline{choc}}$$
$$nil \mid nil \mid !\,coin.\overline{choc}.nil$$

Infinite Behaviour - 3

Iteration

$$\frac{}{E^* \xrightarrow{\epsilon} \surd} \qquad \text{and} \qquad \frac{E \xrightarrow{\mu} E'}{E^* \xrightarrow{\mu} E'; E^*}$$

This iteration operator is the classical one of regular expressions.

A few Process Description Languages

CCS: Calculus of Communicating Processes

Milner - 1980

The set of actions Act_τ consists of a set of labels Λ, of the set $\overline{\Lambda}$ of complementary labels and of the distinct action τ, the syntax is

$$E ::= nil \mid \mu.E \mid E\backslash L \mid E[f] \mid E_1 + E_2 \mid E_1|E_2 \mid recX.E$$

Moreover we have:

- $\mu \in Act_\tau$;
- $L \subseteq \Lambda$;
- $f : Act_\tau \to Act_\tau$;
- $f(\bar{\alpha}) = \overline{f(\alpha)}$ and $f(\tau) = \tau$.

CCS has been studied with Bisimulation and Testing Semantics

SCCS: Synchronous Calculus of Communicating Processes

Milner - 1983

The set of actions Act is an Abelian group containing a set of labels Λ, and of complementary actions $\overline{\Lambda}$ with over-dashed actions, the neutral element is 1, the syntax is

$$E ::= nil \mid \mu : E \mid E \upharpoonright L \mid E_1 + E_2 \mid E_1 \times E_2 \mid recX.E$$

where

- $\mu \in Act \cup \{1\}$,
- $L \subseteq \Lambda$,
- : denotes action prefixing

There is no relabelling operator, it is expressible via the other operators.

SCCS has been studied with Bisimulation Semantics

TCSP: Theoretical Communicating Sequential Processes

Brookes-Hoare-Roscoe - 1984

The set of actions is a set Λ, and the syntax is

$$E ::= stop \mid skip \mid a \rightarrow E \mid E\backslash L \mid E[f] \mid E_1; E_2 \mid E_1 \sqcap E_2 \mid E_1 \Box E_2 \mid E_1 \parallel E_2 \mid E_1 \lll E_2 \mid E_1 \|[L]\| E_2 \mid A$$

where

- $a \in \Lambda$, $L \subseteq \Lambda$, $f : \Lambda \rightarrow \Lambda$,
- the operators $\sqcap$ and $\Box$ denote internal and external choice respectively;
- the operator $\rightarrow$ denotes action prefixing
- A is a process constant

CSP has been studied with Failure Semantics - a variant of Testing Sem.

ACP: Algebra of Communicating Processes

Bergstra-Klop - 1984

The set of actions Λ_τ consists of a finite set of labels Λ and of special action τ, the syntax is

$$E ::= \sqrt{} \mid a \mid E\backslash L \mid E/L \mid E[f] \mid E_1 \cdot E_2 \mid E_1 + E_2 \mid E_1 \parallel E_2 \mid E_1 \between E_2 \mid E_1 |_c E_2 \mid A$$

- $a \in \Lambda_\tau$, $L \subseteq \Lambda$, $f : \Lambda \to \Lambda$;
- the operator $\cdot$ denotes sequential composition;
- A is a process constant.
- The original notation for operators $\cdot\backslash L$, $\cdot/L$ e $\cdot[f]$ are $\delta_L(\cdot)$, $\tau_L(\cdot)$ and $\rho_f(\cdot)$) respectively.

ACP has been studied with Bisimulation and Branching Bis. Semantics

LOTOS: Language of Temporal Order Specification

Standard ISO - 1988

The set of actions Λ_i contains a set of labels Λ and the distinct label i, the syntax is

$$E ::= stop \mid exit \mid \mu; E \mid E/L \mid E[f] \mid E_1 \gg E_2 \mid E_1 [> E_2 \mid E_1 + E_2 \mid E_1 \parallel E_2 \mid E_1 \lll E_2 \mid E_1 \|[L]\| E_2 \mid A$$

- $\mu \in \Lambda_i$, $L \subseteq \Lambda$, $f : \Lambda \to \Lambda$;
- the operator ; denotes action prefixing;
- the operator $\gg$ denotes parallel composition;
- A is a process constant.

LOTOS has been studied with Bisimulation and Testing Semantics

A gentle introduction to π-calculus

From CCS to π-calculus - 1

Consider a scenario of somebody willing to buy a pizza.

In CCS, we can model this situation by composing in parallel the client C, and the "pizzaiolo" P.

From CCS to π-calculus - 1

Consider a scenario of somebody willing to buy a pizza.

In CCS, we can model this situation by composing in parallel the client C, and the "pizzaiolo" P.

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

The client C asks for a pizza, pays for it and takes it away.

From CCS to π-calculus - 1

Consider a scenario of somebody willing to buy a pizza.

In CCS, we can model this situation by composing in parallel the client C, and the "pizzaiolo" P.

$$C \triangleq \overline{askPizza}.\overline{pay}.pizza \qquad P \triangleq askPizza.pay.\overline{pizza}$$

The client C asks for a pizza, pays for it and takes it away.
The "pizzaiolo" P receives the request for the pizza, gets the money and delivers the pizza.

From CCS to π-calculus - 2

If we use values, i.e. CCS with value passing, we can add further details to our system.

From CCS to π-calculus - 2

If we use values, i.e. CCS with value passing, we can add further details to our system.

$$C \triangleq \overline{askPizza}\langle margherita\rangle.\overline{pay}\langle 5\ Euro\rangle.pizza$$

The client asks for a Margherita, pays the due amount and eats the pizza.

From CCS to π-calculus - 2

If we use values, i.e. CCS with value passing, we can add further details to our system.

$$C \triangleq \overline{askPizza}\langle margherita\rangle.\overline{pay}\langle 5\ Euro\rangle.pizza$$

$$P \triangleq askPizza(x).pay(y).\textbf{if } y = price(x) \textbf{ then } pizza \textbf{ else } \textbf{if } y > price(x) \textbf{ then } \overline{pizza}.\overline{output}\langle y - price(x)\rangle \textbf{ else } \overline{askMoney}$$

The client asks for a Margherita, pays the due amount and eats the pizza. The "pizzaiolo" receives the request for the pizza, gets the money then checks the received amount and gives back the requested pizza and possibly the change.

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

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

The client can communicate the address where he wants the pizza be delivered.

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

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

The client can communicate the address where he wants the pizza be delivered.

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

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

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

The client can communicate the address where he wants the pizza be delivered.

$$\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P \xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu pizza)\overline{myHome}\langle pizza\rangle.P$$

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

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

The client can communicate the address where he wants the pizza be delivered.

$$\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P \xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu pizza)\overline{myHome}\langle pizza\rangle.P$$

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

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

The client can communicate the address where he wants the pizza be delivered.

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

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

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

The client can communicate the address where he wants the pizza be delivered.

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

Syntax of π-calculus

We assume a countably infinite set of names $\mathcal{N}$ is defined.

(Processes) P	$::=$	S	sum
	$\mid$	$P_1 \mid P_2$	parallel composition
	$\mid$	$(\nu x)P$	name restriction
	$\mid$	$!P$	replication
(Sums) S	$::=$	$\mathbf{0}$	inactive process (nil)
	$\mid$	$\pi.P$	prefix
	$\mid$	$S_1 + S_2$	choice
(Prefixes) π	$::=$	$\overline{x}\langle y \rangle$	sends y on x
	$\mid$	$x(z)$	replaces z with the name received on x
	$\mid$	τ	internal action
	$\mid$	$[x = y]\pi$	matching: tests equality of x and y

Notation, Comments and Remarks

- $(\nu z)P$ is alike CCS restriction $P\backslash z$.
- $!P$ models replication and denotes the parallel composition of an arbitrary number of copies of P.
- $[x = y]\pi.P$ is known as name matching: it is equivalent to **if** $x = y$ **then** $\pi.P$.
- Occurrences of $\mathbf{0}$ will sometimes be omitted, thus, e.g., $\overline{x}\langle y\rangle.\mathbf{0}$ will be written $\overline{x}\langle y\rangle$.

Notation, Comments and Remarks

- $(\nu z)P$ is alike CCS restriction $P\backslash z$.
- $!P$ models replication and denotes the parallel composition of an arbitrary number of copies of P.
- $[x = y]\pi.P$ is known as name matching: it is equivalent to **if** $x = y$ **then** $\pi.P$.
- Occurrences of $\mathbf{0}$ will sometimes be omitted, thus, e.g., $\overline{x}\langle y\rangle.\mathbf{0}$ will be written $\overline{x}\langle y\rangle$.
- $x(z)$ indicates input while $\overline{x}\langle y\rangle$ indicates output.

Notation, Comments and Remarks

- $(\nu z)P$ is alike CCS restriction $P\backslash z$.
- $!P$ models replication and denotes the parallel composition of an arbitrary number of copies of P.
- $[x = y]\pi.P$ is known as name matching: it is equivalent to **if** $x = y$ **then** $\pi.P$.
- Occurrences of $\mathbf{0}$ will sometimes be omitted, thus, e.g., $\overline{x}\langle y\rangle.\mathbf{0}$ will be written $\overline{x}\langle y\rangle$.
- $x(z)$ indicates input while $\overline{x}\langle y\rangle$ indicates output.
- In $x(z).P$ e $(\nu z)P$, the name z is *bound* in P (i.e., P is the scope of such name). A name that is not bound is called *free*.
- $fn(P)$ e $bn(P)$ are the sets of all free, resp. bound, names of P.
- We take processes up to *alpha-conversion*, denoted by $=_\alpha$, which permits renaming of a bound name with a *fresh* name that is not already used.

An LTS for π-calculus

$(IN)\ a(x).P \xrightarrow{ab} P[b/x]$

$(OUT)\ \bar{a}\langle b\rangle.P \xrightarrow{\bar{a}b} P$

$$(COM)\ \frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\bar{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

Symmetric Com Rule

$$(RES)\ \frac{P \xrightarrow{\alpha} P'}{(\nu b)P \xrightarrow{\alpha} (\nu b)P'} \quad b \notin n(\alpha)$$

$$(OP)\ \frac{P \xrightarrow{\bar{a}b} P'}{(\nu b)P \xrightarrow{\bar{a}(b)} P'} \quad a \neq b$$

$$(CLO)\ \frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\bar{a}(b)} Q'}{P \mid Q \xrightarrow{\tau} (\nu b)(P' \mid Q')} \quad b \notin fn(P)$$

Symmetric Close Rule

$$(PAR)\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad bn(\alpha) \cap fn(Q) = \emptyset$$

Symmetric Par Rule

$$(EQ)\ \frac{P \xrightarrow{\alpha} P'}{[a = a]P \xrightarrow{\alpha} P'}$$

$$(REP)\ \frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

Structural Congruence for π-calculus - I

Why Structural Congruence?

The syntax of π-calculus processes is to some extent *too concrete* (even when taken up-to α-conversion):

1. The order processes are composed in parallel should not matter.
2. The order processes "summed" should not matter.
3. The order names are restricted should not matter.

In fact, we shall make sure that processes differing only for the above aspects are always equivalent.

Structural Congruence for π-calculus - I

Why Structural Congruence?

The syntax of π-calculus processes is to some extent *too concrete* (even when taken up-to α-conversion):

1. The order processes are composed in parallel should not matter.
2. The order processes "summed" should not matter.
3. The order names are restricted should not matter.

In fact, we shall make sure that processes differing only for the above aspects are always equivalent.

By taking processes up to a suitable structural congruence we can:

1. Write processes in a canonical form.
2. Represent all possible interactions with fewer rules.

Structural Congruence for π-calculus - II

$$P \mid \mathbf{0} \equiv P \qquad P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$$

$$S + \mathbf{0} \equiv S \qquad S_1 + S_2 \equiv S_2 + S_1 \qquad S_1 + (S_2 + S_3) \equiv (S_1 + S_2) + S_3$$

Structural Congruence for π-calculus - II

$$P \mid \mathbf{0} \equiv P \qquad P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$$

$$S + \mathbf{0} \equiv S \qquad S_1 + S_2 \equiv S_2 + S_1 \qquad S_1 + (S_2 + S_3) \equiv (S_1 + S_2) + S_3$$

$$!P \equiv P \mid !P \qquad [a = a]\pi.P \equiv \pi.P$$

$$(\nu a)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad \frac{a \notin fn(P)}{P \mid (\nu a)Q \equiv (\nu a)(P \mid Q)}$$

Structural Congruence for π-calculus - II

$$P \mid \mathbf{0} \equiv P \qquad P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$$

$$S + \mathbf{0} \equiv S \qquad S_1 + S_2 \equiv S_2 + S_1 \qquad S_1 + (S_2 + S_3) \equiv (S_1 + S_2) + S_3$$

$$!P \equiv P \mid !P \qquad [a = a]\pi.P \equiv \pi.P$$

$$(\nu a)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad \frac{a \notin fn(P)}{P \mid (\nu a)Q \equiv (\nu a)(P \mid Q)}$$

$$P \equiv P \qquad \frac{P \equiv Q}{Q \equiv P} \qquad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R} \text{ (equivalence)}$$

Structural Congruence for π-calculus - II

$P \mid \mathbf{0} \equiv P$ $\quad P_1 \mid P_2 \equiv P_2 \mid P_1$ $\quad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$

$S + \mathbf{0} \equiv S$ $\quad S_1 + S_2 \equiv S_2 + S_1$ $\quad S_1 + (S_2 + S_3) \equiv (S_1 + S_2) + S_3$

$!P \equiv P \mid !P$ $\quad [a = a]\pi.P \equiv \pi.P$

$(\nu a)\mathbf{0} \equiv \mathbf{0}$ $\quad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$ $\quad \dfrac{a \notin fn(P)}{P \mid (\nu a)Q \equiv (\nu a)(P \mid Q)}$

$P \equiv P$ $\quad \dfrac{P \equiv Q}{Q \equiv P}$ $\quad \dfrac{P \equiv Q \quad Q \equiv R}{P \equiv R}$ (equivalence)

$\dfrac{P =_\alpha P'}{P \equiv P'}$ $\quad \dfrac{P \equiv P'}{\mathbb{C}[P] \equiv \mathbb{C}[P']}$ (congruence)

Canonical Form

For each π-calculus process P there exist:

1. a finite number of names $x_1, ..., x_k$,
2. a finite number of sums $S_1, ..., S_n$, and
3. a finite number of processes $P_1, ..., P_m$ such that

$$P \equiv (\nu x_1)...(\nu x_k)\big(S_1|...|S_n|!P_1|...|!P_m\big)$$

The structural congruence permits rearranging the terms describing π-calculus processes so that any two possibly interacting subterms (composed in parallel) can be put side by side.

All interactions can then be expressed by considering only a very small number of cases.

Reduction Semantics: An alternative sem. for π-calculus

The so-called *reduction semantics* focuses on *internal* moves $P \longmapsto Q$ only and takes significantly advantage of structural congruence.

$$(RTAU) \quad \frac{}{(\tau.P + S) \longmapsto P}$$

$$(RCOM) \quad \frac{}{(a(x).P_1 + S_1)|(\overline{a}\langle b\rangle.P_2 + S_2) \longmapsto P_1[^b/_x]|P_2}$$

$$(RPAR) \quad \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q}$$

$$(RRES) \quad \frac{P \longmapsto P'}{(\nu a)P \longmapsto (\nu a)P'}$$

$$(RSTRUCT) \quad \frac{P \equiv Q \quad Q \longmapsto Q' \quad Q' \equiv P'}{P \longmapsto P'}$$

Harmony Lemma

The reduction semantics and the LTS semantics can be tightly reconciled.

Notation

Given two relations $\mathcal{R}$ and $\mathcal{S}$ on processes, we write $P\ \mathcal{RS}\ Q$ if there exists a process R such that $P\ \mathcal{R}\ R$ and $R\ \mathcal{S}\ Q$.

Theorem (Harmony Lemma)

For any π-calculus process P we have:

1. $P \equiv \xrightarrow{\alpha} P'$ *implies* $P \xrightarrow{\alpha} \equiv P'$
2. $P \longmapsto P'$ *if and only if* $P \xrightarrow{\tau} \equiv P'$

Electoral Propaganda

We shall see how the use of restricted channels can prevent intrusions.

Assume we want to campaign for Romano and have set up the following scenario :

Naive Campaigning

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{\mathit{air}}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq \mathit{air}(x).\overline{\mathit{wire}}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq \mathit{wire}(y).\overline{\mathit{highvolume}}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

Electoral Propaganda

We shall see how the use of restricted channels can prevent intrusions.

Assume we want to campaign for Romano and have set up the following scenario :

Naive Campaigning

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{\mathit{air}}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq \mathit{air}(x).\overline{\mathit{wire}}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq \mathit{wire}(y).\overline{\mathit{highvolume}}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

This system will evolve as follows:

$$
\begin{aligned}
\mathit{Ad} &\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Romano} \rangle \mid \mathit{Loudspeaker} \\
&\longmapsto \overline{\mathit{highvolume}}\langle \mathit{vote\ for\ Romano} \rangle
\end{aligned}
$$

Electoral Propaganda and Intrusions

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{\mathit{air}}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq \mathit{air}(x).\overline{\mathit{wire}}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq \mathit{wire}(y).\overline{\mathit{highvolume}}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

Electoral Propaganda and Intrusions

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{\mathit{air}}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq \mathit{air}(x).\overline{\mathit{wire}}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq \mathit{wire}(y).\overline{\mathit{highvolume}}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

Let $\mathit{Rival} \triangleq \mathit{wire}(z).\overline{\mathit{wire}}\langle \mathit{vote\ for\ Silvio} \rangle$

$$
\begin{aligned}
\mathit{Ad} \mid \mathit{Rival} &\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Romano} \rangle \mid \mathit{Loudspeaker} \mid \mathit{Rival} \\
&\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Silvio} \rangle \mid \mathit{Loudspeaker} \\
&\longmapsto \overline{\mathit{highvolume}}\langle \mathit{vote\ for\ Silvio} \rangle
\end{aligned}
$$

Electoral Propaganda and Intrusions

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{\mathit{air}}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq \mathit{air}(x).\overline{\mathit{wire}}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq \mathit{wire}(y).\overline{\mathit{highvolume}}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

Let $\mathit{Rival} \triangleq \mathit{wire}(z).\overline{\mathit{wire}}\langle \mathit{vote\ for\ Silvio} \rangle$

$$
\begin{aligned}
\mathit{Ad} \mid \mathit{Rival} &\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Romano} \rangle \mid \mathit{Loudspeaker} \mid \mathit{Rival} \\
&\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Silvio} \rangle \mid \mathit{Loudspeaker} \\
&\longmapsto \overline{\mathit{highvolume}}\langle \mathit{vote\ for\ Silvio} \rangle
\end{aligned}
$$

Rival could use *wire* because it is a public channel.

A secure propaganda would be:

$$\mathit{SecureAd} \triangleq (\nu\, \mathit{air}, \mathit{wire})(\mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker})$$

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server
- c_{BS} is the communication channel between Bob and the server

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server
- c_{BS} is the communication channel between Bob and the server
- c_{AB} is the new (secure) channel that Alice and Bob want to establish to communicate.

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server
- c_{BS} is the communication channel between Bob and the server
- c_{AB} is the new (secure) channel that Alice and Bob want to establish to communicate.

We can code Alice, Bob and the Server as follows:

$$
\begin{array}{rcl}
A & \triangleq & (\nu c_{AB})\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle mess\rangle \\
S & \triangleq & !c_{AS}(x).\overline{c_{BS}}\langle x\rangle \mid !c_{BS}(y).\overline{c_{AS}}\langle y\rangle \\
B & \triangleq & c_{BS}(z).z(w). < use\ z >
\end{array}
$$

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server
- c_{BS} is the communication channel between Bob and the server
- c_{AB} is the new (secure) channel that Alice and Bob want to establish to communicate.

We can code Alice, Bob and the Server as follows:

$$\begin{array}{rcl} A & \triangleq & (\nu c_{AB})\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle mess\rangle \\ S & \triangleq & !c_{AS}(x).\overline{c_{BS}}\langle x\rangle \mid !c_{BS}(y).\overline{c_{AS}}\langle y\rangle \\ B & \triangleq & c_{BS}(z).z(w).\ < use\ z > \end{array}$$

$$(\nu c_{AS}, c_{BS})(A|S|B) \longmapsto\longmapsto\longmapsto (\nu c_{AS}, c_{BS}, c_{AB})(S \mid\ < use\ mess >)$$

And now . . . Global Computing with Klaim

Global Computers

Global Computers can be seen as Distributed Systems with a number of distinguishing features:

- Wide area distribution

Global Computers

Global Computers can be seen as Distributed Systems with a number of distinguishing features:

- Wide area distribution
- Variability of interconnection structures

Global Computers

Global Computers can be seen as Distributed Systems with a number of distinguishing features:

- Wide area distribution
- Variability of interconnection structures
- Physical (and Logical) Mobility

Global Computers

Global Computers can be seen as Distributed Systems with a number of distinguishing features:

- Wide area distribution
- Variability of interconnection structures
- Physical (and Logical) Mobility
- Possibility of Disconnections

Global Computers

Global Computers can be seen as Distributed Systems with a number of distinguishing features:

- Wide area distribution
- Variability of interconnection structures
- Physical (and Logical) Mobility
- Possibility of Disconnections
- Indistinguishability of Failures from Slow Reactions

Global Computers

Global Computers can be seen as Distributed Systems with a number of distinguishing features:

- Wide area distribution
- Variability of interconnection structures
- Physical (and Logical) Mobility
- Possibility of Disconnections
- Indistinguishability of Failures from Slow Reactions
- Importance of Quality of Service

Programming Global Computers

Programming Languages would definitely benefit from explicit primitives for

- **Distribution**
  - computing over different (explicit) localities
- **Mobility**
  - moving agents and computations over localities
- **Concurrency**
  - considering parallel and non-deterministic computations
- **Access Rights**
  - maintaining privacy and integrity of data

Languages for Global Computing would of course benefit from formal semantics and associated logics for reasoning on programs behaviour.

Our Research Goals

Developing a simple *programming language* and associated tools for network aware and migrating applications with a tractable semantic theory that permits programs (services) verification.

Our Starting Points (1980 — . . .)

- Process Algebras
  - CCS, CSP, . . .
- Calculi and languages for Mobility
  - Pi-calculus, Obliq, Ambients, . . .
- Tuple Based Interaction Models
  - Linda
- Modal and Temporal Logics
  - HML, CTL, ACTL, μ-calculus, . . .

Kernel Language for Agent Interaction and Mobility

Process Calculus Flavored

- Small set of basic combinator;
- Clean operational semantics.

Linda based communication model

- Asynchronous communication;
- Shared tuple spaces;
- Pattern Matching

Explicit use of localities

- Multiple distributed tuple spaces;
- Code and Process mobility.

Tuples and Templates

Tuples

$$(\text{"}foo\text{"}, 10 + 5, true)$$

contain only Actual Fields

Templates

$$(\text{"}foo\text{"}, 10 + 5, !u)$$

contain both Actual Fields and Formal Fields

Pattern Matching

- Formal fields match any field of the same type
- Actual fields match if identical
- tuple $(\text{"}foo\text{"}, 10 + 5, true)$ matches template $(!s, 15, !b)$

From Linda and Process Algebras to KLAIM

Explicit Localities to model distribution

- *Physical Locality* (sites)
- *Logical Locality* (names for sites)
- A distinct name *self* (or *here*) indicates the site a process is on.

Allocation environment to associate sites to logical localities

- This avoids the programmers to know the exact physical structure.

Process Algebras Operators to compose programs

- Sequentialization
- Parallel composition
- Creation of new names

KLAIM Nodes and KLAIM Nets

KLAIM Nodes

consist of:

- a site
- a tuple space
- a set of parallel processes
- an allocation environment

KLAIM Nets

are:

- a set of KLAIM nodes linked via the allocation environment

KLAIM Syntax

$$
\begin{array}{lcll}
P & ::= & \mathbf{nil} & \text{(null process)} \\
 & | & a.P & \text{(action prefixing)} \\
 & | & P_1 \mid P_2 & \text{(parallel composition)} \\
 & | & P_1 + P_2 & \text{(choice)} \\
 & | & X & \text{(process variable)} \\
 & | & A\langle \widetilde{P}, \widetilde{\ell}, \widetilde{e} \rangle & \text{(process invocation)} \\
a & ::= & \mathbf{out}(t)@\ell \mid \mathbf{in}(t)@\ell \mid \mathbf{read}(t)@\ell \mid \mathbf{eval}(P)@\ell \mid \mathbf{newloc}(u) & \\
t & ::= & e \mid P \mid \ell \mid !\,x \mid !\,X \mid !\,u \mid t_1, t_2 & \\
N & ::= & s ::_\rho P & \text{(node)} \\
 & | & N_1 \parallel N_2 & \text{(net composition)}
\end{array}
$$

Dining Philosophers in KLAIM

$$
\begin{aligned}
P_i = \;& \\
& \#\ \textit{think...} \\
& \mathbf{in}(\text{“chopstick”})@c_i. \\
& \mathbf{in}(\text{“chopstick”})@c_{(i+1)\mathbf{mod}\ n}. \\
& \#\ \textit{eat...} \\
& \mathbf{out}(\text{“chopstick”})@c_i. \\
& \mathbf{out}(\text{“chopstick”})@c_{(i+1)\mathbf{mod}\ n}. \\
& P_i
\end{aligned}
$$

$$
\begin{aligned}
& c_0 :: (\text{“chopstick”}) \parallel p_0 :: P_0 \parallel c_1 :: (\text{“chopstick”}) \parallel p_1 :: P_1 \parallel \\
& c_2 :: (\text{“chopstick”}) \parallel p_2 :: P_2 \parallel c_3 :: (\text{“chopstick”}) \parallel p_3 :: P_3 \parallel \\
& c_4 :: (\text{“chopstick”}) \parallel p_4 :: P_4
\end{aligned}
$$

ATTENTION

This system may deadlock! One of the well-known solutions can be devised to avoid problems.

Structural Congruence

Monoid Laws for $\|$

$N \parallel \mathbf{0} \equiv N$

$N_1 \parallel N_2 \equiv N_2 \parallel N_1$

$(N_1 \parallel N_2) \parallel N_3 \equiv N_1 \parallel (N_2 \parallel N_3)$

Congruence Laws

(Alpha)	N	$\equiv$	N'	if $N =_\alpha N'$
(RCom)	$(\nu l_1)(\nu l_2)N$	$\equiv$	$(\nu l_2)(\nu l_1)N$	
(Ext)	$N_1 \parallel (\nu l)N_2$	$\equiv$	$(\nu l)(N_1 \parallel N_2)$	if $l \notin fn(N_1)$
(Abs)	$l ::_\rho C$	$\equiv$	$l ::_\rho (C \mid \mathbf{nil})$	
(Clone)	$l ::_\rho C_1 \mid C_2$	$\equiv$	$l ::_\rho C_1 \parallel l ::_\rho C_2$	
(Rec)	$l ::_\rho \mathbf{rec}X.P$	$\equiv$	$l ::_\rho P[\mathbf{rec}X.P/X]$	

Matching Function

$$match(l, l) = \epsilon$$

$$match(!x, l) = [^{l}/_{x}]$$

$$match(!X, P) = [^{P}/_{X}]$$

$$\frac{match(T_1, t_1) = \sigma_1 \qquad match(T_2, t_2) = \sigma_2}{match(\ T_1, T_2\ ,\ t_1, t_2\ ) = \sigma_1 \circ \sigma_2}$$

Process Reduction Rules

$$\text{(Red-Out)}\quad \frac{\rho(u) = l' \qquad \mathcal{E}[\![\, t \,]\!]_\rho = t'}{l ::_\rho \overline{u}\langle t\rangle.P \;\|\; l' ::_{\rho'} \mathbf{nil} \longmapsto l ::_\rho P \,\|\, l' ::_{\rho'} \langle t'\rangle}$$

$$\text{(Red-Eval)}\quad \frac{\rho(u) = l'}{l ::_\rho \mathbf{eval}(P_2)@u.P_1 \;\|\; l' ::_{\rho'} \mathbf{nil} \longmapsto l ::_\rho P_1 \,\|\, l' ::_{\rho'} P_2}$$

$$\text{(Red-In)}\quad \frac{\rho(u) = l' \qquad match(\mathcal{E}[\![\, T \,]\!]_\rho, t) = \sigma}{l ::_\rho \mathbf{in}(T)@u.P \;\|\; l' ::_{\rho'} \langle t\rangle \longmapsto l ::_\rho P\sigma \,\|\, l' ::_{\rho'} \mathbf{nil}}$$

$$\text{(Red-Read)}\quad \frac{\rho(u) = l' \qquad match(\mathcal{E}[\![\, T \,]\!]_\rho, t) = \sigma}{l ::_\rho \mathbf{read}(T)@u.P \;\|\; l' ::_{\rho'} \langle t\rangle \longmapsto l ::_\rho P\sigma \,\|\, l' ::_{\rho'} \langle t\rangle}$$

$$\text{(Red-New)}\quad l ::_\rho \mathbf{newloc}(l').P \longmapsto (\nu l')(l ::_\rho P \,\|\, l' ::_{\rho[l'/\mathtt{self}]} \mathbf{nil})$$

Nets Reduction Rules

$$\text{(Red-Par)} \quad \frac{N_1 \longmapsto N_1'}{N_1 \parallel N_2 \longmapsto N_1' \parallel N_2}$$

$$\text{(Red-Res)} \quad \frac{N \longmapsto N'}{(\nu l)N \longmapsto (\nu l)N'}$$

$$\text{(Red-Struct)} \quad \frac{N \equiv M \longmapsto M' \equiv N'}{N \longmapsto N'}$$

μKLAIM: A core calculus for KLAIM

We take away from KLAIM:

- distinction between logical and physical localities/addresses: No allocation environment
- higher order communication: No process in tuples

μKLAIM Syntax

$$
\begin{array}{lcl}
N & ::= & l :: R \mid N_1 \parallel N_2 \\
R & ::= & P \mid \langle et \rangle \\
P & ::= & \mathbf{nil} \mid act.P \mid P_1 \mid P_2 \mid X \mid \mathbf{rec}X.P \\
act & ::= & \bar{l}\langle t \rangle \mid \mathbf{in}(T)@l \mid \mathbf{read}(T)@l \mid \\
 & & \mathbf{eval}(P)@l \mid \mathbf{newloc}(u)
\end{array}
$$

$$
t ::= f \mid f, t \quad \text{where} \quad f ::= e \mid l \mid u
$$

$$
T ::= F \mid F, T \quad \text{where} \quad F ::= f \mid !\,x \mid !\,u
$$

Matching predicate

Matching μKLAIM:

$$(\mathrm{M}_1)\quad match(V, V) = [] \qquad (\mathrm{M}_2)\quad match(!\,x, V) = [V/x]$$

$$(\mathrm{M}_3)\quad match(l, l) = [] \qquad (\mathrm{M}_4)\quad match(!\,u, l) = [l/u]$$

$$(\mathrm{M}_5)\quad \frac{match(f, t) = \sigma_1 \qquad match(F, T) = \sigma_2}{match(\ (f, t)\ ,\ (F, T)\ ) = \sigma_1 \circ \sigma_2}$$

Matching examples:

- $match((\text{"}Foo\text{"}, l), (!x, !u)) = [\text{"}Foo\text{"}/x, l/u]$
- $match((\text{"}Foo\text{"}, 5), (\text{"}Foo\text{"}, !x)) = [5/x]$

Structural congruence

(Commutativity) $N_1 \parallel N_2 \equiv N_2 \parallel N_1$

(Associativity) $(N_1 \parallel N_2) \parallel N_3 \equiv N_1 \parallel (N_2 \parallel N_3)$

(Absorbtion) $l :: P \equiv l :: (P|\mathbf{nil})$

(Unfolding) $l :: \mathbf{rec}X.P \equiv l :: P[\mathbf{rec}X.P/X]$

(Cloning) $l :: (P_1|P_2) \equiv l :: P_1 \parallel l :: P_2$

Labelled Operational Semantics

Actions and Transitions

- The set of transition labels λ is: $\lambda ::= l : act \mid \tau$
- the operational semantics of μKLAIM nets is defined using relation $\stackrel{\cdot}{\longrightarrow}$.

Some Rules

- $l_1 :: \overline{l_2}\langle t \rangle.P \xrightarrow{l_1 : \overline{l_2}\langle \mathcal{E}[\![t]\!] \rangle} l_1 :: P$
- $$\frac{N_1 \xrightarrow{l_1 : \overline{l_2}\langle et \rangle} N_2}{N_1 \parallel l_2 :: P \xrightarrow{\tau} N_2 \parallel l_2 :: P \parallel l_2 :: \langle et \rangle}$$
- $l_1 :: \mathbf{in}(T)@l_2.P \xrightarrow{l_1 : \mathbf{in}(T)@l_2} l :: P$
- $$\frac{N_1 \xrightarrow{l_1 : \mathbf{in}(T)@l_2} N_2 \ \sigma = match(T, et)}{N_1 \parallel l_2 :: \langle et \rangle \xrightarrow{\tau} N_2\sigma \parallel l_2 :: \mathbf{nil}}$$

Transition Rules - 1

$$l :: act.P \xrightarrow{l:[\![act]\!]} l :: P \ (\textsc{Act})$$

$$\frac{N_1 \xrightarrow{l_1:\overline{l_2}\langle et\rangle} N_2}{N_1 \parallel l_2 :: P \xrightarrow{\tau} N_2 \parallel l_2 :: P \parallel l_2 :: \langle et\rangle} \ (\textsc{Out})$$

$$\frac{N_1 \xrightarrow{l_1:\mathbf{eval}(Q)@l_2} N_2}{N_1 \parallel l_2 :: P \xrightarrow{\tau} N_2 \parallel l_2 :: P \parallel l_2 :: Q} \ (\textsc{Eval})$$

$$\frac{N_1 \xrightarrow{l_1:\mathbf{in}(T)@l_2} N_2 \quad \sigma = match(T, et)}{N_1 \parallel l_2 :: \langle et\rangle \xrightarrow{\tau} N_2\sigma \parallel l_2 :: \mathbf{nil}} \ (\textsc{In})$$

Transition Rules - 2

$$\frac{N_1 \xrightarrow{l_1:\mathbf{read}(T)@l_2} N_2 \quad \sigma = match(T, et)}{N_1 \parallel l_2 :: \langle et \rangle \xrightarrow{\tau} N_2\sigma \parallel l_2 :: \langle et \rangle} \text{ (READ)}$$

$$\frac{N_1 \xrightarrow{l_1:\mathbf{newloc}(u)} N_2 \quad l \notin N_2}{N_1 \parallel N_2[l/u] \parallel l :: \mathbf{nil}} \text{ (NEW)}$$

$$\frac{N_1 \xrightarrow{l:act} N_2}{N_1 \parallel N \xrightarrow{l:act} N_2 \parallel N} \text{ (PAR)}$$

$$\frac{N_1 \equiv N_1' \quad N_1' \xrightarrow{\lambda} N_2' \quad N_2' \equiv N_2}{N_1 \xrightarrow{\lambda} N_2} \text{ (STRUCT)}$$

Conclusions?

It is just a start! No conclusions ....

For work going on in Firenze, please visit our web site
`http://music.dsi.unifi.it`

Most Importantly see:
`http://rap.dsi.unifi.it/tapas`

Many Thanks for your attention

Bibliography

- Apt K.R., Olderog E.-R., *Verification of Sequential and Concurrent Programs*, Springer-Verlag, 1997.
  I took from here the first example of these lectures.
- Fokkink Wan, *Introduction to Process Algebra*, Springer, 2000.
  A gentle introduction to ACP.
- Milner R., *Communication and Concurrency*, Prentice Hall, 1989.
  The classical book on CCS and Bisimulation.
- Roscoe A.W., *The Theory and Practice of Concurrency*, Prentice Hall, 1998.
  A good book on TCSP and the failure Model.
- Bowman H. and Gomez R., *Concurrency Theory: Calculi and Automata for Modelling Untimed and Timed Concurrent Systems*, Springer, 2006.
  A new book on concurrency theory based on LOTOS.

Bibliography ctd.

- Baeten J.C.M. and Weijland W.P., *Process Algebra*, Cambridge University Press, 1990.
  The first book on ACP and Branching Bisimulation.
- Hennessy M., *Algebraic theory of processes*, Springer-Verlag, 2001.
  A simple introduction to Algebraic, Denotational and Operational Semantics of processes based on Testing Equivalence.
- Van Glabbeek R.J., *The Linear Time - Branching Time Spectrum I*. The Semantics of Concrete, Sequential Processes*, Handbook on Process Algebras, North Holland, 2001.
  A good overview of behavioral equivalences over LTS.
- Sangiorgi D. and Walker D., *PI-Calculus: A Theory of Mobile Processes*, Cambridge University Press, 2001.
  THE book on π-calculus.
- For Klaim see: `http://music.dsi.unifi.it`