Process algebrae fundamentals

Distributed Systems / Technologies
Sistemi Distribuiti / Tecnologie

Giovanni Ciatto Andrea Omicini
giovanni.ciatto@unibo.it andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2019/2020

Outline

Outline

Lecture goals I

- Designing and implementing concurrent/distributed system (C/DS) is hard also because it is difficult to
  - precisely specify the behaviour of a C/DS
  - verify a protocol correctly works in all possible situations
- Here we will exemplify how process algebra-based modelling and reasoning may ease both a C/DS specification – making it clearer and more precise –, and its verification—enabling us to prove some properties always hold (e.g. termination, deadlock-freedom, etc.)
- Computer scientists are usually interested in proving general the correctness of a C/DS

  e.g. the PayPal payment protocol MUST be deadlock-free, and MUST guarantee a 0-or-1 semantics to payments, in ANY possibile scenario
- Software engineers are usually interested in precisely specifing the semantics of the software they are designing

  e.g. what do you exactly mean by "the `in` operation must suspend in case no tuple is available?"

Lecture goals I

- Designing and implementing concurrent/distributed system (C/DS) is hard also because it is difficult to
  - precisely specify the behaviour of a C/DS
  - verify a protocol correctly works in all possible situations
- Here we will exemplify how process algebra-based modelling and reasoning may ease both a C/DS specification – making it clearer and more precise –, and its verification—enabling us to prove some properties always hold (e.g. termination, deadlock-freedom, etc.)
- Computer scientists are usually interested in proving general the correctness of a C/DS
  - e.g. the PayPal payment protocol MUST be deadlock-free, and MUST guarantee a 0-or-1 semantics to payments, in ANY possibile scenario
- Software engineers are usually interested in precisely specifing the semantics of the software they are designing
  - e.g. what do you exactly mean by "the `in` operation must suspend in case no tuple is available?"

Lecture goals I

- Designing and implementing concurrent/distributed system (C/DS) is hard also because it is difficult to
  - precisely specify the behaviour of a C/DS
  - verify a protocol correctly works in all possible situations
- Here we will exemplify how process algebra-based modelling and reasoning may ease both a C/DS specification – making it clearer and more precise –, and its verification—enabling us to prove some properties always hold (e.g. termination, deadlock-freedom, etc.)
- Computer scientists are usually interested in proving general the correctness of a C/DS

  e.g. the PayPal payment protocol MUST be deadlock-free, and MUST guarantee a 0-or-1 semantics to payments, in ANY possibile scenario
- Software engineers are usually interested in precisely specifing the semantics of the software they are designing

  e.g. what do you exactly mean by "the `in` operation must suspend in case no tuple is available?"

Lecture goals I

- Designing and implementing concurrent/distributed system (C/DS) is hard also because it is difficult to
  - precisely specify the behaviour of a C/DS
  - verify a protocol correctly works in all possible situations
- Here we will exemplify how process algebra-based modelling and reasoning may ease both a C/DS specification – making it clearer and more precise –, and its verification—enabling us to prove some properties always hold (e.g. termination, deadlock-freedom, etc.)
- Computer scientists are usually interested in proving general the correctness of a C/DS

  e.g. the PayPal payment protocol MUST be deadlock-free, and MUST guarantee a 0-or-1 semantics to payments, in ANY possibile scenario
- Software engineers are usually interested in precisely specifing the semantics of the software they are designing

  e.g. what do you exactly mean by "the `in` operation must suspend in case no tuple is available?"

LINDA as a running example

- In giving you the LINDA specification during the previous Lab lessons, we deliberately used the natural language, often issuing ambiguous statements on purpose

- The aim was to let you understand that several arbitrary interpretations may be derived from an ambiguous specification
  - which is a nightmare for engineers

- You should already have implemented LINDA. We will now model (i.e. design) it by means of CCS, transition rules, and labelled transition systems

- We will then show how several semantics could be defined from the same natural language-based specification

LINDA as a running example

- In giving you the LINDA specification during the previous Lab lessons, we deliberately used the natural language, often issuing ambiguous statements on purpose

- The aim was to let you understand that several arbitrary interpretations may be derived from an ambiguous specification
  - which is a nightmare for engineers

- You should already have implemented LINDA. We will now model (i.e. design) it by means of CCS, transition rules, and labelled transition systems

- We will then show how several semantics could be defined from the same natural language-based specification

LINDA as a running example

- In giving you the LINDA specification during the previous Lab lessons, we deliberately used the natural language, often issuing ambiguous statements on purpose
- The aim was to let you understand that several arbitrary interpretations may be derived from an ambiguous specification
  - which is a nightmare for engineers
- You should already have implemented LINDA. We will now model (i.e. design) it by means of CCS, transition rules, and labelled transition systems
- We will then show how several semantics could be defined from the same natural language-based specification

LINDA as a running example

- In giving you the LINDA specification during the previous Lab lessons, we deliberately used the natural language, often issuing ambiguous statements on purpose
- The aim was to let you understand that several arbitrary interpretations may be derived from an ambiguous specification
  - which is a nightmare for engineers
- You should already have implemented LINDA. We will now model (i.e. design) it by means of CCS, transition rules, and labelled transition systems
- We will then show how several semantics could be defined from the same natural language-based specification

Outline

A workflow for semantics specification

In order to formally model a C/DS using the Calculus of Communicating Systems (CCS), and define its semantics, one usually need to perform the following steps:

1. Define a syntax for the C/DS system, covering all possible situations
   - this can be done by means of grammars, which is employed to define operators, actions, and processes
2. Instantiate a particular C/DS system as a word over the language generated by the grammar above
3. Define its semantics in terms of a Labelled Transition System (LST), which usually implies:
   - formalise the transition rules governing the system behaviour
   - adopting the aforementioned word as the initial state
   - generating the graph of possible states for the system by recursive applying all enabled transition rules to the initial state
4. Verify properties over the LTS by means of model-checkers and temporal logics (we won't do that)

A workflow for semantics specification

In order to formally model a C/DS using the Calculus of Communicating Systems (CCS), and define its semantics, one usually need to perform the following steps:

1. Define a syntax for the C/DS system, covering all possible situations
   - this can be done by means of grammars, which is employed to define operators, actions, and processes
2. Instantiate a particular C/DS system as a word over the language generated by the grammar above
3. Define its semantics in terms of a Labelled Transition System (LST), which usually implies:
   - formalise the transition rules governing the system behaviour
   - adopting the aforementioned word as the initial state
   - generating the graph of possible states for the system by recursive applying all enabled transition rules to the initial state
4. Verify properties over the LTS by means of model-checkers and temporal logics (we won't do that)

A workflow for semantics specification

In order to formally model a C/DS using the Calculus of Communicating Systems (CCS), and define its semantics, one usually need to perform the following steps:

1. Define a syntax for the C/DS system, covering all possible situations
   - this can be done by means of grammars, which is employed to define operators, actions, and processes
2. Instantiate a particular C/DS system as a word over the language generated by the grammar above
3. Define its semantics in terms of a Labelled Transition System (LST), which usually implies:
   - formalise the transition rules governing the system behaviour
   - adopting the aforementioned word as the initial state
   - generating the graph of possible states for the system by recursively applying all enabled transition rules to the initial state
4. Verify properties over the LTS by means of model-checkers and temporal logics (we won't do that)

A workflow for semantics specification

In order to formally model a C/DS using the Calculus of Communicating Systems (CCS), and define its semantics, one usually need to perform the following steps:

1. Define a syntax for the C/DS system, covering all possible situations
   - this can be done by means of grammars, which is employed to define operators, actions, and processes
2. Instantiate a particular C/DS system as a word over the language generated by the grammar above
3. Define its semantics in terms of a Labelled Transition System (LST), which usually implies:
   - formalise the transition rules governing the system behaviour
   - adopting the aforementioned word as the initial state
   - generating the graph of possible states for the system by recursively applying all enabled transition rules to the initial state
4. Verify properties over the LTS by means of model-checkers and temporal logics (we won't do that)

Outline

Formalising LINDA with process algebrae

We will now provide an example showing how process algebrae could be employed to formally describe the semantics of LINDA

1. We will model tuple spaces by means of CCS
2. We will then provide their formal semantics as standalone systems, by means of a LTS
3. We will model agents/users by means of CCS
4. We will then provide their formal semantics as standalone systems, by means of a LTS
5. Finally we will provide the syntax and the semantics of a coordinated system, i.e., the parallel composition of several agents/users interacting with and by means of a tuple space

Formalising LINDA with process algebrae

We will now provide an example showing how process algebrae could be employed to formally describe the semantics of LINDA

1. We will model tuple spaces by means of CCS
2. We will then provide their formal semantics as standalone systems, by means of a LTS
3. We will model agents/users by means of CCS
4. We will then provide their formal semantics as standalone systems, by means of a LTS
5. Finally we will provide the syntax and the semantics of a coordinated system, i.e., the parallel composition of several agents/users interacting with and by means of a tuple space

Formalising LINDA with process algebrae

We will now provide an example showing how process algebrae could be employed to formally describe the semantics of LINDA

1. We will model tuple spaces by means of CCS
2. We will then provide their formal semantics as standalone systems, by means of a LTS
3. We will model agents/users by means of CCS
4. We will then provide their formal semantics as standalone systems, by means of a LTS
5. Finally we will provide the syntax and the semantics of a coordinated system, i.e., the parallel composition of several agents/users interacting with and by means of a tuple space

Formalising LINDA with process algebrae

We will now provide an example showing how process algebrae could be employed to formally describe the semantics of LINDA

1. We will model tuple spaces by means of CCS
2. We will then provide their formal semantics as standalone systems, by means of a LTS
3. We will model agents/users by means of CCS
4. We will then provide their formal semantics as standalone systems, by means of a LTS
5. Finally we will provide the syntax and the semantics of a coordinated system, i.e., the parallel composition of several agents/users interacting with and by means of a tuple space

Formalising LINDA with process algebrae

We will now provide an example showing how process algebrae could be employed to formally describe the semantics of LINDA

1. We will model tuple spaces by means of CCS
2. We will then provide their formal semantics as standalone systems, by means of a LTS
3. We will model agents/users by means of CCS
4. We will then provide their formal semantics as standalone systems, by means of a LTS
5. Finally we will provide the syntax and the semantics of a coordinated system, i.e., the parallel composition of several agents/users interacting with and by means of a tuple space

Next in Line...

Tuple Spaces – Syntax

A grammar for LINDA's tuple space processes

$$
\begin{array}{rcll}
TS & ::= & (T \cup TS) \mid (\langle T \rangle \cup TS) \mid \emptyset & \text{tuple spaces} \\
T & ::= & t \mid t' \mid t'' \mid \cdots \mid t_1 \mid t_1' \mid t_1'' \mid \cdots & \text{tuples}
\end{array}
$$

! $\langle t \rangle$ represents a tuple that is going to be inserted within the tuple space

Subject to the following axioms

$$
\begin{array}{rcll}
X \cup Y & \equiv & Y \cup X & \text{union is commutative} \\
X \cup (Y \cup Z) & \equiv & (X \cup Y) \cup Z & \text{parentheses are useless for unions} \\
X \cup \emptyset & \equiv & X & \text{neutral element for union}
\end{array}
$$

Tuple Spaces – Syntax

A grammar for LINDA's tuple space processes

$$TS ::= (T \cup TS) \mid (\langle T \rangle \cup TS) \mid \emptyset \qquad \text{tuple spaces}$$

$$T ::= t \mid t' \mid t'' \mid \cdots \mid t_1 \mid t_1' \mid t_1'' \mid \cdots \qquad \text{tuples}$$

! $\langle t \rangle$ represents a tuple that is going to be inserted within the tuple space

Subject to the following axioms

$$X \cup Y \equiv Y \cup X \qquad \text{union is commutative}$$

$$X \cup (Y \cup Z) \equiv (X \cup Y) \cup Z \qquad \text{parentheses are useless for unions}$$

$$X \cup \emptyset \equiv X \qquad \text{neutral element for union}$$

Tuple Spaces – Several possible processes

Several tuple space processes could be syntactically represented by means of this grammar:

e.g. $ts_0 = \emptyset$

e.g. $ts_1 = t \equiv t \cup \emptyset \equiv \emptyset \cup t \equiv t \cup \emptyset \cup \emptyset$

e.g. $ts_2 = t_1 \cup t_2 \equiv t_1 \cup t_2 \cup \emptyset \equiv t_2 \cup t_1 \equiv t_1 \cup \emptyset \cup t_2$

e.g. $ts_3 = t_1 \cup t_2 \cup t_3 \equiv t_1 \cup t_2 \cup t_3 \cup \emptyset$

e.g. $ts_3 = t_1 \cup \langle t_2 \rangle \cup t_3 \cup \langle t_2 \rangle$

We say that ts_0, ts_1, ts_2, ts_3 are words in $\mathcal{L}(TS)$, namely, the language generated by TS, i.e., the set of all possible tuples

Tuple Spaces – Several possible processes

Several tuple space processes could be syntactically represented by means of this grammar:

e.g. $ts_0 = \emptyset$

e.g. $ts_1 = t \equiv t \cup \emptyset \equiv \emptyset \cup t \equiv t \cup \emptyset \cup \emptyset$

e.g. $ts_2 = t_1 \cup t_2 \equiv t_1 \cup t_2 \cup \emptyset \equiv t_2 \cup t_1 \equiv t_1 \cup \emptyset \cup t_2$

e.g. $ts_3 = t_1 \cup t_2 \cup t_3 \equiv t_1 \cup t_2 \cup t_3 \cup \emptyset$

e.g. $ts_3 = t_1 \cup \langle t_2 \rangle \cup t_3 \cup \langle t_2 \rangle$

We say that ts_0, ts_1, ts_2, ts_3 are words in $\mathcal{L}(TS)$, namely, the language generated by TS, i.e., the set of all possible tuples

Tuple Spaces – Several possible processes

Several tuple space processes could be syntactically represented by means of this grammar:

e.g. $ts_0 = \emptyset$

e.g. $ts_1 = t \equiv t \cup \emptyset \equiv \emptyset \cup t \equiv t \cup \emptyset \cup \emptyset$

e.g. $ts_2 = t_1 \cup t_2 \equiv t_1 \cup t_2 \cup \emptyset \equiv t_2 \cup t_1 \equiv t_1 \cup \emptyset \cup t_2$

e.g. $ts_3 = t_1 \cup t_2 \cup t_3 \equiv t_1 \cup t_2 \cup t_3 \cup \emptyset$

e.g. $ts_3 = t_1 \cup \langle t_2 \rangle \cup t_3 \cup \langle t_2 \rangle$

We say that ts_0, ts_1, ts_2, ts_3 are words in $\mathcal{L}(TS)$, namely, the language generated by TS, i.e., the set of all possible tuples

Tuple Spaces – Several possible processes

Several tuple space processes could be syntactically represented by means of this grammar:

e.g. $ts_0 = \emptyset$

e.g. $ts_1 = t \equiv t \cup \emptyset \equiv \emptyset \cup t \equiv t \cup \emptyset \cup \emptyset$

e.g. $ts_2 = t_1 \cup t_2 \equiv t_1 \cup t_2 \cup \emptyset \equiv t_2 \cup t_1 \equiv t_1 \cup \emptyset \cup t_2$

e.g. $ts_3 = t_1 \cup t_2 \cup t_3 \equiv t_1 \cup t_2 \cup t_3 \cup \emptyset$

e.g. $ts_3 = t_1 \cup \langle t_2 \rangle \cup t_3 \cup \langle t_2 \rangle$

We say that ts_0, ts_1, ts_2, ts_3 are words in $\mathcal{L}(TS)$, namely, the language generated by TS, i.e., the set of all possible tuples

Tuple Spaces – Several possible processes

Several tuple space processes could be syntactically represented by means of this grammar:

e.g. $ts_0 = \emptyset$

e.g. $ts_1 = t \equiv t \cup \emptyset \equiv \emptyset \cup t \equiv t \cup \emptyset \cup \emptyset$

e.g. $ts_2 = t_1 \cup t_2 \equiv t_1 \cup t_2 \cup \emptyset \equiv t_2 \cup t_1 \equiv t_1 \cup \emptyset \cup t_2$

e.g. $ts_3 = t_1 \cup t_2 \cup t_3 \equiv t_1 \cup t_2 \cup t_3 \cup \emptyset$

e.g. $ts_3 = t_1 \cup \langle t_2 \rangle \cup t_3 \cup \langle t_2 \rangle$

We say that ts_0, ts_1, ts_2, ts_3 are words in $\mathcal{L}(TS)$, namely, the language generated by TS, i.e., the set of all possible tuples

Tuple Spaces – Semantics

A grammar for tuple space related **events**

$$
\begin{array}{rcll}
E_{TS} & ::= & !O_{TS} \mid ?I_{TS} \mid \tau & \text{event for tuple spaces} \\
O_{TS} & ::= & in(T) \mid rd(T) & \text{output events} \\
I_{TS} & ::= & out(T) & \text{input events}
\end{array}
$$

LINDA's tuple spaces as a Labelled Transition System

We define a tuple space $\mathcal{TS}$ as a LTS, i.e. a *quartet* $\langle S, s_0, \longrightarrow_{\mathcal{TS}}, E \rangle$ where:

- $S = \mathcal{L}(TS)$ is a set of possible states
- $s_0 \in S$ is the initial state
- $E = \mathcal{L}(E_{TS})$ is a set of event labels
- $\longrightarrow_{\mathcal{TS}} \subseteq (S \times E \times S)$ is the set of admissible transitions

Tuple Spaces – Semantics

A grammar for tuple space related **events**

$$
\begin{array}{rcll}
E_{TS} & ::= & !O_{TS} \mid ?I_{TS} \mid \tau & \text{event for tuple spaces} \\
O_{TS} & ::= & in(T) \mid rd(T) & \text{output events} \\
I_{TS} & ::= & out(T) & \text{input events}
\end{array}
$$

LINDA's tuple spaces as a Labelled Transition System

We define a tuple space $\mathcal{TS}$ as a LTS, i.e. a *quartet* $\langle S, s_0, \longrightarrow_{\mathcal{TS}}, E\rangle$ where:

- $S = \mathcal{L}(TS)$ is a set of possible states
- $s_0 \in S$ is the initial state
- $E = \mathcal{L}(E_{TS})$ is a set of event labels
- $\longrightarrow_{\mathcal{TS}} \subseteq (S \times E \times S)$ is the set of admissible transitions

Transition Rules

The set of admissible transitions $\longrightarrow_{\mathcal{TS}}$ is usually intensionally defined by means of transition rules matching the following pattern:

$$\frac{C_1,\ \ldots,\ C_N}{s \xrightarrow{e}_{\mathcal{TS}} s'} \qquad \text{[RULE-NAME]}$$

meaning that transition $(s, e, s') \in \longrightarrow_{\mathcal{TS}}$ only if preconditions $C_1,\ \ldots,\ C_n$ hold within the source state s

- If no precondition of interest need to be specified, the numerator and the fraction line are usually omitted
- The states s and s' are usually represented by means of non-ground formulas containing both terminal and non-terminal symbols, i.e. variables
  - you can then imagine a transition rule as a rewriting rule

Transition Rules

The set of admissible transitions $\longrightarrow_{\mathcal{TS}}$ is usually intensionally defined by means of transition rules matching the following pattern:

$$\frac{C_1,\ \ldots,\ C_N}{s \xrightarrow{e}_{\mathcal{TS}} s'} \qquad \text{[RULE-NAME]}$$

meaning that transition $(s, e, s') \in \longrightarrow_{\mathcal{TS}}$ only if preconditions $C_1, \ldots, C_n$ hold within the source state s

- If no precondition of interest need to be specified, the numerator and the fraction line are usually omitted
- The states s and s' are usually represented by means of non-ground formulas containing both terminal and non-terminal symbols, i.e. variables
  - you can then imagine a transition rule as a rewriting rule

Transition Rules

The set of admissible transitions $\longrightarrow_{\mathcal{TS}}$ is usually intensionally defined by means of transition rules matching the following pattern:

$$\frac{C_1,\ \ldots,\ C_N}{s \xrightarrow{e}_{\mathcal{TS}} s'} \qquad \text{[RULE-NAME]}$$

meaning that transition $(s, e, s') \in \longrightarrow_{\mathcal{TS}}$ only if preconditions $C_1,\ \ldots,\ C_n$ hold within the source state s

- If no precondition of interest need to be specified, the numerator and the fraction line are usually omitted
- The states s and s' are usually represented by means of non-ground formulas containing both terminal and non-terminal symbols, i.e. variables
  - you can then imagine a transition rule as a rewriting rule

Tuple Spaces – Transition Rules

So, for what concerns Linda tuple spaces, we define the following transition rules:

$$TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup \langle t \rangle \quad [\text{SCHEDULE}]$$

$$TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{TS}} TS \cup t \quad [\text{INSERT}]$$

$$TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \quad [\text{CONSUME}]$$

$$TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \quad [\text{OBSERVE}]$$

Tuple Spaces – Transition Rules

So, for what concerns LINDA tuple spaces, we define the following transition rules:

$$TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup \langle t \rangle \quad \text{[SCHEDULE]}$$

$$TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{TS}} TS \cup t \quad \text{[INSERT]}$$

$$TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \quad \text{[CONSUME]}$$

$$TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \quad \text{[OBSERVE]}$$

Tuple Spaces – Transition Rules

So, for what concerns LINDA tuple spaces, we define the following transition rules:

$$TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup \langle t \rangle \quad \text{[SCHEDULE]}$$

$$TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{TS}} TS \cup t \quad \text{[INSERT]}$$

$$TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \quad \text{[CONSUME]}$$

$$TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \quad \text{[OBSERVE]}$$

Tuple Spaces – Transition Rules

So, for what concerns LINDA tuple spaces, we define the following transition rules:

$$TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup \langle t \rangle \quad \text{[SCHEDULE]}$$

$$TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{TS}} TS \cup t \quad \text{[INSERT]}$$

$$TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \quad \text{[CONSUME]}$$

$$TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \quad \text{[OBSERVE]}$$

Tuple Spaces – The state graph

Example where $s_0 = \emptyset$

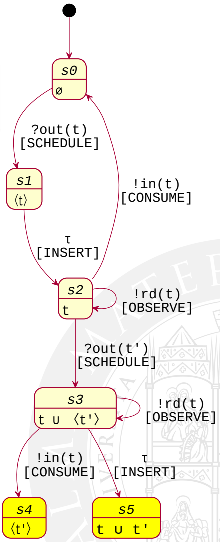
http://www.plantuml.com/plantuml/svg/RP1...0m00

$$TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup \langle t \rangle \quad \text{[SCHEDULE]}$$

$$TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{TS}} TS \cup t \quad \text{[INSERT]}$$

$$TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \quad \text{[CONSUME]}$$

$$TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \quad \text{[OBSERVE]}$$



Next in Line...

Users – Syntax

A grammar for User processes

$$
\begin{array}{rcll}
U & ::= & \mathtt{out}(T) \cdot U \mid \mathtt{in}(TT) \cdot U \mid \mathtt{rd}(TT) \cdot U & \text{users} \\
 & \mid & \langle T \rangle \mid (U + U) \mid 0 & \\
 & & & \\
TT & ::= & \bar{t} \mid \bar{t}' \mid \bar{t}'' \mid \cdots \mid \bar{t}_1 \mid \bar{t}_1' \mid \bar{t}_1'' \mid \cdots & \text{templates}
\end{array}
$$

! where $\langle t \rangle$ represent the result of an access action

Subject to the following axioms

$$
\begin{array}{rcll}
X \cdot Y & \not\equiv & Y \cdot X & \text{sequence is non-commutative} \\
X \cdot (Y \cdot Z) & \equiv & (X \cdot Y) \cdot Z & \text{parentheses are useless for sequences} \\
X \cdot 0 & \equiv & X & \text{neutral element for sequences} \\
\hline
X + Y & \equiv & Y + X & \text{choice is commutative} \\
X + (Y + Z) & \equiv & (X + Y) + Z & \text{parentheses are useless for choices} \\
(X + Y) \cdot Z & \equiv & X \cdot Z + Y \cdot Z & \text{choice is right-distributive w.r.t sequence}
\end{array}
$$

Users – Syntax

A grammar for User processes

$$
\begin{array}{rcll}
U & ::= & \mathtt{out}(T) \cdot U \mid \mathtt{in}(TT) \cdot U \mid \mathtt{rd}(TT) \cdot U & \text{users} \\
 & \mid & \langle T \rangle \mid (U + U) \mid 0 & \\
 & & & \\
TT & ::= & \bar{t} \mid \bar{t}' \mid \bar{t}'' \mid \cdots \mid \bar{t}_1 \mid \bar{t}'_1 \mid \bar{t}''_1 \mid \cdots & \text{templates}
\end{array}
$$

! where $\langle t \rangle$ represent the result of an access action

Subject to the following axioms

$$
\begin{array}{rcll}
X \cdot Y & \not\equiv & Y \cdot X & \text{sequence is non-commutative} \\
X \cdot (Y \cdot Z) & \equiv & (X \cdot Y) \cdot Z & \text{parentheses are useless for sequences} \\
X \cdot 0 & \equiv & X & \text{neutral element for sequences} \\
\hline
X + Y & \equiv & Y + X & \text{choice is commutative} \\
X + (Y + Z) & \equiv & (X + Y) + Z & \text{parentheses are useless for choices} \\
(X + Y) \cdot Z & \equiv & X \cdot Z + Y \cdot Z & \text{choice is right-distributive w.r.t sequence}
\end{array}
$$

Users – Several possible processes

Several user processes could be syntactically represented by means of this grammar (not always making sense):

e.g. $u_0 = \texttt{in}(\bar{t}) \cdot \langle t \rangle \cdot \texttt{out}(t') \cdot \texttt{rd}(\bar{t}'') \cdot \langle t'' \rangle \cdot 0$

e.g. $u_1 = \texttt{out}(t_1) \cdot \texttt{out}(t_2) \cdot \texttt{out}(t_3) \cdot \texttt{rd}(\bar{t}) \cdot \langle t \rangle \cdot 0$

e.g. $u_2 = (\texttt{in}(\bar{t}_1) + \texttt{in}(\bar{t}_2) + \texttt{in}(\bar{t}_3)) \cdot \langle t \rangle \cdot 0$

e.g. $u_3 = \langle t \rangle \cdot \texttt{in}(\bar{t}) \cdot \langle t' \rangle \cdot \texttt{rd}(\bar{t}') \cdot 0$! makes no sense

e.g. $u_4 = \texttt{in}(\bar{t}) \cdot \langle t \rangle \cdot \texttt{out}(t') \cdot u_4$! recursive

Users – Several possible processes

Several user processes could be syntactically represented by means of this grammar (not always making sense):

e.g. $u_0 = \mathtt{in}(\bar{t}) \cdot \langle t \rangle \cdot \mathtt{out}(t') \cdot \mathtt{rd}(\bar{t}'') \cdot \langle t'' \rangle \cdot 0$

e.g. $u_1 = \mathtt{out}(t_1) \cdot \mathtt{out}(t_2) \cdot \mathtt{out}(t_3) \cdot \mathtt{rd}(\bar{t}) \cdot \langle t \rangle \cdot 0$

e.g. $u_2 = (\mathtt{in}(\bar{t}_1) + \mathtt{in}(\bar{t}_2) + \mathtt{in}(\bar{t}_3)) \cdot \langle t \rangle \cdot 0$

e.g. $u_3 = \langle t \rangle \cdot \mathtt{in}(\bar{t}) \cdot \langle t' \rangle \cdot \mathtt{rd}(\bar{t}') \cdot 0$! makes no sense

e.g. $u_4 = \mathtt{in}(\bar{t}) \cdot \langle t \rangle \cdot \mathtt{out}(t') \cdot u_4$! recursive

Users – Several possible processes

Several user processes could be syntactically represented by means of this grammar (not always making sense):

e.g. $u_0 = \mathtt{in}(\bar{t}) \cdot \langle t \rangle \cdot \mathtt{out}(t') \cdot \mathtt{rd}(\bar{t}'') \cdot \langle t'' \rangle \cdot 0$

e.g. $u_1 = \mathtt{out}(t_1) \cdot \mathtt{out}(t_2) \cdot \mathtt{out}(t_3) \cdot \mathtt{rd}(\bar{t}) \cdot \langle t \rangle \cdot 0$

e.g. $u_2 = (\mathtt{in}(\bar{t}_1) + \mathtt{in}(\bar{t}_2) + \mathtt{in}(\bar{t}_3)) \cdot \langle t \rangle \cdot 0$

e.g. $u_3 = \langle t \rangle \cdot \mathtt{in}(\bar{t}) \cdot \langle t' \rangle \cdot \mathtt{rd}(\bar{t}') \cdot 0$! makes no sense

e.g. $u_4 = \mathtt{in}(\bar{t}) \cdot \langle t \rangle \cdot \mathtt{out}(t') \cdot u_4$! recursive

Users – Several possible processes

Several user processes could be syntactically represented by means of this grammar (not always making sense):

e.g. $u_0 = \mathtt{in}(\bar{t}) \cdot \langle t \rangle \cdot \mathtt{out}(t') \cdot \mathtt{rd}(\bar{t}'') \cdot \langle t'' \rangle \cdot 0$

e.g. $u_1 = \mathtt{out}(t_1) \cdot \mathtt{out}(t_2) \cdot \mathtt{out}(t_3) \cdot \mathtt{rd}(\bar{t}) \cdot \langle t \rangle \cdot 0$

e.g. $u_2 = (\mathtt{in}(\bar{t}_1) + \mathtt{in}(\bar{t}_2) + \mathtt{in}(\bar{t}_3)) \cdot \langle t \rangle \cdot 0$

e.g. $u_3 = \langle t \rangle \cdot \mathtt{in}(\bar{t}) \cdot \langle t' \rangle \cdot \mathtt{rd}(\bar{t}') \cdot 0$! makes no sense

e.g. $u_4 = \mathtt{in}(\bar{t}) \cdot \langle t \rangle \cdot \mathtt{out}(t') \cdot u_4$! recursive

Users – Several possible processes

Several user processes could be syntactically represented by means of this grammar (not always making sense):

e.g. $u_0 = \mathtt{in}(\bar{t}) \cdot \langle t \rangle \cdot \mathtt{out}(t') \cdot \mathtt{rd}(\bar{t}'') \cdot \langle t'' \rangle \cdot 0$

e.g. $u_1 = \mathtt{out}(t_1) \cdot \mathtt{out}(t_2) \cdot \mathtt{out}(t_3) \cdot \mathtt{rd}(\bar{t}) \cdot \langle t \rangle \cdot 0$

e.g. $u_2 = (\mathtt{in}(\bar{t}_1) + \mathtt{in}(\bar{t}_2) + \mathtt{in}(\bar{t}_3)) \cdot \langle t \rangle \cdot 0$

e.g. $u_3 = \langle t \rangle \cdot \mathtt{in}(\bar{t}) \cdot \langle t' \rangle \cdot \mathtt{rd}(\bar{t}') \cdot 0$! makes no sense

e.g. $u_4 = \mathtt{in}(\bar{t}) \cdot \langle t \rangle \cdot \mathtt{out}(t') \cdot u_4$! recursive

Users – Semantics

A grammar for tuple space related **events**

$$
\begin{array}{rcll}
E_U & ::= & !O_U \,|\, ?I_U \mid \tau & \text{events for users} \\
O_U & ::= & out(T) & \text{output events} \\
I_U & ::= & rd(TT) \mid in(TT) & \text{input events}
\end{array}
$$

LINDA's users as a Labelled Transition System

We define a user $\mathcal{U}$ as a LTS, i.e. a *quartet* $\langle S, s_0, \longrightarrow_{\mathcal{U}}, E \rangle$ where:

- $S = \mathcal{L}(U)$ is a set of possible states
- $s_0 \in S$ is the initial state
- $E = \mathcal{L}(E_U)$ is a set of event labels
- $\longrightarrow_{\mathcal{U}} \subseteq (S \times E \times S)$ is the set of admissible transitions

Users – Semantics

A grammar for tuple space related **events**

$$
\begin{array}{rcll}
E_U & ::= & !O_U \mid ?I_U \mid \tau & \text{events for users} \\
O_U & ::= & out(T) & \text{output events} \\
I_U & ::= & rd(TT) \mid in(TT) & \text{input events}
\end{array}
$$

LINDA's users as a Labelled Transition System

We define a user $\mathcal{U}$ as a LTS, i.e. a *quartet* $\langle S, s_0, \longrightarrow_{\mathcal{U}}, E \rangle$ where:

- $S = \mathcal{L}(U)$ is a set of possible states
- $s_0 \in S$ is the initial state
- $E = \mathcal{L}(E_U)$ is a set of event labels
- $\longrightarrow_{\mathcal{U}} \subseteq (S \times E \times S)$ is the set of admissible transitions

Users – Transition Rules

For what concerns LINDA users, we define the following transition rules:

$$\mathtt{out}(t) \cdot U \xrightarrow{!out(t)}_{\mathcal{U}} U \qquad \text{[WRITE]}$$

$$\mathtt{rd}(\bar{t}) \cdot U \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U \qquad \text{[READ]}$$

$$\mathtt{in}(\bar{t}) \cdot U \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U \qquad \text{[TAKE]}$$

$$\langle t \rangle \cdot U \xrightarrow{\tau}_{\mathcal{U}} U \qquad \text{[COMPUTE]}$$

$$\frac{U_1 \cdot U_1' \xrightarrow{E}_{\mathcal{U}} U_1'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_1'} \ \text{[CHOICE-L]} \qquad \frac{U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'} \ \text{[CHOICE-R]}$$

Notice that no rule exists matching state 0 since it represents termination

Users – Transition Rules

For what concerns LINDA users, we define the following transition rules:

$$\mathtt{out}(t) \cdot U \xrightarrow{!out(t)}_{\mathcal{U}} U \qquad [\text{WRITE}]$$

$$\mathtt{rd}(\bar{t}) \cdot U \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U \qquad [\text{READ}]$$

$$\mathtt{in}(\bar{t}) \cdot U \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U \qquad [\text{TAKE}]$$

$$\langle t \rangle \cdot U \xrightarrow{\tau}_{\mathcal{U}} U \qquad [\text{COMPUTE}]$$

$$\frac{U_1 \cdot U_1' \xrightarrow{E}_{\mathcal{U}} U_1'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_1'} \; [\text{CHOICE-L}] \qquad \frac{U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'} \; [\text{CHOICE-R}]$$

Notice that no rule exists matching state 0 since it represents termination

Users – Transition Rules

For what concerns LINDA users, we define the following transition rules:

$$\texttt{out}(t) \cdot U \xrightarrow{!out(t)}_{\mathcal{U}} U \qquad [\text{WRITE}]$$

$$\texttt{rd}(\bar{t}) \cdot U \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U \qquad [\text{READ}]$$

$$\texttt{in}(\bar{t}) \cdot U \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U \qquad [\text{TAKE}]$$

$$\langle t \rangle \cdot U \xrightarrow{\tau}_{\mathcal{U}} U \qquad [\text{COMPUTE}]$$

$$\frac{U_1 \cdot U_1' \xrightarrow{E}_{\mathcal{U}} U_1'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_1'} \; [\text{CHOICE-L}] \qquad \frac{U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'} \; [\text{CHOICE-R}]$$

Notice that no rule exists matching state 0 since it represents termination

Users – Transition Rules

For what concerns LINDA users, we define the following transition rules:

$$\texttt{out}(t) \cdot U \xrightarrow{!out(t)}_{\mathcal{U}} U \quad \text{[WRITE]}$$

$$\texttt{rd}(\bar{t}) \cdot U \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U \quad \text{[READ]}$$

$$\texttt{in}(\bar{t}) \cdot U \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U \quad \text{[TAKE]}$$

$$\langle t \rangle \cdot U \xrightarrow{\tau}_{\mathcal{U}} U \quad \text{[COMPUTE]}$$

$$\frac{U_1 \cdot U_1' \xrightarrow{E}_{\mathcal{U}} U_1'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_1'} \ \text{[CHOICE-L]} \qquad \frac{U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'} \ \text{[CHOICE-R]}$$

Notice that no rule exists matching state 0 since it represents termination

Users – Transition Rules

For what concerns LINDA users, we define the following transition rules:

$$\mathtt{out}(t) \cdot U \xrightarrow{!out(t)}_{\mathcal{U}} U \qquad [\text{WRITE}]$$

$$\mathtt{rd}(\bar{t}) \cdot U \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U \qquad [\text{READ}]$$

$$\mathtt{in}(\bar{t}) \cdot U \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U \qquad [\text{TAKE}]$$

$$\langle t \rangle \cdot U \xrightarrow{\tau}_{\mathcal{U}} U \qquad [\text{COMPUTE}]$$

$$\frac{U_1 \cdot U_1' \xrightarrow{E}_{\mathcal{U}} U_1'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_1'} \; [\text{CHOICE-L}] \qquad \frac{U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'} \; [\text{CHOICE-R}]$$

Notice that no rule exists matching state 0 since it represents termination

Users – Transition Rules

For what concerns LINDA users, we define the following transition rules:

$$\mathtt{out}(t) \cdot U \xrightarrow{!out(t)}_{\mathcal{U}} U \qquad \text{[WRITE]}$$

$$\mathtt{rd}(\bar{t}) \cdot U \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U \qquad \text{[READ]}$$

$$\mathtt{in}(\bar{t}) \cdot U \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U \qquad \text{[TAKE]}$$

$$\langle t \rangle \cdot U \xrightarrow{\tau}_{\mathcal{U}} U \qquad \text{[COMPUTE]}$$

$$\frac{U_1 \cdot U_1' \xrightarrow{E}_{\mathcal{U}} U_1'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_1'} \ \text{[CHOICE-L]} \qquad \frac{U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'} \ \text{[CHOICE-R]}$$

Notice that no rule exists matching state 0 since it represents termination

Users – The state graph

Example where $s_0 = \texttt{out}(t_1) \cdot (\texttt{rd}(\bar{t}_2) + \texttt{in}(\bar{t}_2)) \cdot \langle t_2 \rangle \cdot \texttt{in}(\bar{t}_3) \cdot \texttt{rd}(\bar{t}_1) \cdot 0$

$$\texttt{out}(t) \cdot U \xrightarrow{!out(t)}_{\mathcal{U}} U \qquad \text{[WRITE]}$$

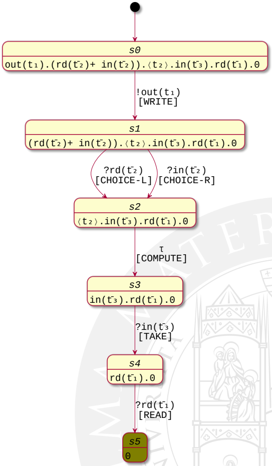
$$\texttt{rd}(\bar{t}) \cdot U \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U \qquad \text{[READ]}$$

$$\texttt{in}(\bar{t}) \cdot U \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U \qquad \text{[TAKE]}$$

$$\langle t \rangle \cdot U \xrightarrow{\tau}_{\mathcal{U}} U \qquad \text{[COMPUTE]}$$

$$\frac{U_1 \cdot U_1' \xrightarrow{E}_{\mathcal{U}} U_1'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_1'} \qquad \text{[CHOICE-L]}$$

$$\frac{U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'}{U_1 \cdot U_1' + U_2 \cdot U_2' \xrightarrow{E}_{\mathcal{U}} U_2'} \qquad \text{[CHOICE-R]}$$



! Zoomable image here

Next in Line...

Coordinated Systems – Syntax

A grammar for coordinated systems

$$
\begin{array}{rcll}
CS & ::= & US \parallel TS & \text{coordinated system} \\
US & ::= & (U \parallel US) \mid 0 & \text{list of users}
\end{array}
$$

Subject to the following axioms

$$
\begin{array}{rcll}
X \parallel Y & \neq & Y \parallel X & \text{parallel is not commutative} \\
X \parallel (Y \parallel Z) & \equiv & (X \parallel Y) \parallel Z & \text{parentheses are useless for unions} \\
X \parallel 0 & \equiv & X & \text{neutral element for parallel}
\end{array}
$$

! A non-commutative parallel operator makes the processes identifiable by means of their *index*

Notice that the states of a coordinated systems must match the patterns:

$$U_1 \parallel \cdots \parallel U_i \parallel \cdots \parallel U_n \parallel TS$$
$$US \parallel U_i \parallel US' \parallel TS$$

Coordinated Systems – Syntax

A grammar for coordinated systems

$$
\begin{array}{rcll}
CS & ::= & US \parallel TS & \text{coordinated system} \\
US & ::= & (U \parallel US) \mid 0 & \text{list of users}
\end{array}
$$

Subject to the following axioms

$$
\begin{array}{rcll}
X \parallel Y & \not\equiv & Y \parallel X & \text{parallel is not commutative} \\
X \parallel (Y \parallel Z) & \equiv & (X \parallel Y) \parallel Z & \text{parentheses are useless for unions} \\
X \parallel 0 & \equiv & X & \text{neutral element for parallel}
\end{array}
$$

! A non-commutative parallel operator makes the processes identifiable by means of their *index*

Notice that the states of a coordinated systems must match the patterns:

$$
U_1 \parallel \cdots \parallel U_i \parallel \cdots \parallel U_n \parallel TS
$$
$$
US \parallel U_i \parallel US' \parallel TS
$$

Coordinated Systems – Syntax

A grammar for coordinated systems

$$
\begin{array}{rcll}
CS & ::= & US \parallel TS & \text{coordinated system} \\
US & ::= & (U \parallel US) \mid 0 & \text{list of users}
\end{array}
$$

Subject to the following axioms

$$
\begin{array}{rcll}
X \parallel Y & \not\equiv & Y \parallel X & \text{parallel is not commutative} \\
X \parallel (Y \parallel Z) & \equiv & (X \parallel Y) \parallel Z & \text{parentheses are useless for unions} \\
X \parallel 0 & \equiv & X & \text{neutral element for parallel}
\end{array}
$$

! A non-commutative parallel operator makes the processes identifiable by means of their *index*

Notice that the states of a coordinated systems must match the patterns:

$$
U_1 \parallel \cdots \parallel U_i \parallel \cdots \parallel U_n \parallel TS
$$
$$
US \parallel U_i \parallel US' \parallel TS
$$

Coordinated Systems – Semantics

A grammar for coordinated systems related **events**

$$E_{CS} \quad ::= \quad out(T) \mid in(T) \mid rd(T) \mid \tau \quad \text{events}$$

Coordinated systems as a Labelled Transition System

We define a user $\mathcal{CS}$ as a LTS, i.e. a *quartet* $\langle S, s_0, \longrightarrow_{\mathcal{CS}}, E \rangle$ where:

- $S = \mathcal{L}(CS)$ is a set of possible states
- $s_0 \in S$ is the initial state
- $E = \mathcal{L}(E_{CS})$ is a set of event labels
- $\longrightarrow_{\mathcal{CS}} \subseteq (S \times E \times S)$ is the set of admissible transitions

Coordinated Systems – Semantics

A grammar for coordinated systems related **events**

$$E_{CS} \quad ::= \quad out(T) \mid in(T) \mid rd(T) \mid \tau \qquad \text{events}$$

Coordinated systems as a Labelled Transition System

We define a user $\mathcal{CS}$ as a LTS, i.e. a *quartet* $\langle S, s_0, \longrightarrow_{\mathcal{CS}}, E \rangle$ where:

- $S = \mathcal{L}(CS)$ is a set of possible states
- $s_0 \in S$ is the initial state
- $E = \mathcal{L}(E_{CS})$ is a set of event labels
- $\longrightarrow_{\mathcal{CS}} \subseteq (S \times E \times S)$ is the set of admissible transitions

Coordinated Systems – Transition Rules I

For what concerns coordinated systems, we define the following transition rules (pay attention to the indexes in rules names):

$$\frac{\texttt{out}(t) \cdot U_i \xrightarrow{!out(t)}_{\mathcal{U}} U_i \qquad TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup \langle t \rangle}{US \parallel \texttt{out}(t) \cdot U_i \parallel US' \parallel TS \xrightarrow{out(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup \langle t \rangle} \; [\text{OUT}_i]$$

$$\frac{\texttt{in}(\bar{t}) \cdot U_i \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \qquad t \in \bar{t}}{US \parallel \texttt{in}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{in(t)}_{\mathcal{CS}} US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS} \; [\text{IN}_i]$$

$$\frac{\texttt{rd}(\bar{t}) \cdot U_i \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \qquad t \in \bar{t}}{US \parallel \texttt{rd}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{rd(t)}_{\mathcal{CS}} US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS \cup t} \; [\text{RD}_i]$$

Coordinated Systems – Transition Rules I

For what concerns coordinated systems, we define the following transition rules (pay attention to the indexes in rules names):

$$\frac{\mathtt{out}(t) \cdot U_i \xrightarrow{!out(t)}_{\mathcal{U}} U_i \qquad TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup \langle t \rangle}{US \parallel \mathtt{out}(t) \cdot U_i \parallel US' \parallel TS \xrightarrow{out(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup \langle t \rangle} \; [\text{OUT}_i]$$

$$\frac{\mathtt{in}(\bar{t}) \cdot U_i \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \qquad t \in \bar{t}}{US \parallel \mathtt{in}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{in(t)}_{\mathcal{CS}} US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS} \; [\text{IN}_i]$$

$$\frac{\mathtt{rd}(\bar{t}) \cdot U_i \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \qquad t \in \bar{t}}{US \parallel \mathtt{rd}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{rd(t)}_{\mathcal{CS}} US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS \cup t} \; [\text{RD}_i]$$

Coordinated Systems – Transition Rules I

For what concerns coordinated systems, we define the following transition rules (pay attention to the indexes in rules names):

$$\frac{\mathtt{out}(t) \cdot U_i \xrightarrow{!out(t)}_{\mathcal{U}} U_i \qquad TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup \langle t \rangle}{US \parallel \mathtt{out}(t) \cdot U_i \parallel US' \parallel TS \xrightarrow{out(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup \langle t \rangle} \; [\mathrm{OUT}_i]$$

$$\frac{\mathtt{in}(\bar{t}) \cdot U_i \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \qquad t \in \bar{t}}{US \parallel \mathtt{in}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{in(t)}_{\mathcal{CS}} US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS} \; [\mathrm{IN}_i]$$

$$\frac{\mathtt{rd}(\bar{t}) \cdot U_i \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \qquad t \in \bar{t}}{US \parallel \mathtt{rd}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{rd(t)}_{\mathcal{CS}} US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS \cup t} \; [\mathrm{RD}_i]$$

Coordinated Systems – Transition Rules II

$$\frac{\langle t \rangle \cdot U_i \xrightarrow{\tau}_{\mathcal{U}} U_i}{US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS \xrightarrow{\tau}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS} \quad [\text{COMPUTE}_i]$$

$$\frac{TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{TS}} TS \cup t}{US \parallel TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{CS}} US \parallel TS \cup t} \quad [\text{INSERT}]$$

$$\frac{U_i + U_i' \xrightarrow{E'}_{\mathcal{U}} U_i'' \qquad TS \xrightarrow{E''}_{\mathcal{TS}} TS' \qquad E = \gamma(E', E'')}{US \parallel U_i + U_i' \parallel US' \parallel TS \xrightarrow{E}_{\mathcal{CS}} US \parallel U_i'' \parallel US' \parallel TS'} \quad [\text{CHOICE}_i]$$

where the partial funtion $\gamma(\cdot, \cdot)$ is defined as follows:

- $\gamma(!out(t), ?out(t)) = out(t)$
- $\gamma(?rd(\bar{t}), !rd(t)) = rd(t)$
- $\gamma(?in(\bar{t}), !in(t)) = in(t)$

Coordinated Systems – Transition Rules II

$$\frac{\langle t \rangle \cdot U_i \xrightarrow{\tau}_{\mathcal{U}} U_i}{US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS \xrightarrow{\tau}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS} \quad [\text{COMPUTE}_i]$$

$$\frac{TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{TS}} TS \cup t}{US \parallel TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{CS}} US \parallel TS \cup t} \quad [\text{INSERT}]$$

$$\frac{U_i + U_i' \xrightarrow{E'}_{\mathcal{U}} U_i'' \qquad TS \xrightarrow{E''}_{\mathcal{TS}} TS' \qquad E = \gamma(E', E'')}{US \parallel U_i + U_i' \parallel US' \parallel TS \xrightarrow{E}_{\mathcal{CS}} US \parallel U_i'' \parallel US' \parallel TS'} \quad [\text{CHOICE}_i]$$

where the partial funtion $\gamma(\cdot, \cdot)$ is defined as follows:

- $\gamma(!out(t), ?out(t)) = out(t)$
- $\gamma(?in(\bar{t}), !in(t)) = in(t)$
- $\gamma(?rd(\bar{t}), !rd(t)) = rd(t)$

Coordinated Systems – Transition Rules II

$$\frac{\langle t \rangle \cdot U_i \xrightarrow{\tau}_{\mathcal{U}} U_i}{US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS \xrightarrow{\tau}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS} \quad [\text{COMPUTE}_i]$$

$$\frac{TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{TS}} TS \cup t}{US \parallel TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{CS}} US \parallel TS \cup t} \quad [\text{INSERT}]$$

$$\frac{U_i + U_i' \xrightarrow{E'}_{\mathcal{U}} U_i'' \qquad TS \xrightarrow{E''}_{\mathcal{TS}} TS' \qquad E = \gamma(E', E'')}{US \parallel U_i + U_i' \parallel US' \parallel TS \xrightarrow{E}_{\mathcal{CS}} US \parallel U_i'' \parallel US' \parallel TS'} \quad [\text{CHOICE}_i]$$

where the partial funtion $\gamma(\cdot, \cdot)$ is defined as follows:

- $\gamma(!out(t), ?out(t)) = out(t)$
- $\gamma(?in(\bar{t}), !in(t)) = in(t)$
- $\gamma(?rd(\bar{t}), !rd(t)) = rd(t)$

Coordinated Systems – Transition Rules II

$$\frac{\langle t \rangle \cdot U_i \xrightarrow{\tau}_{\mathcal{U}} U_i}{US \parallel \langle t \rangle \cdot U_i \parallel US' \parallel TS \xrightarrow{\tau}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS} \; [\text{COMPUTE}_i]$$

$$\frac{TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{TS}} TS \cup t}{US \parallel TS \cup \langle t \rangle \xrightarrow{\tau}_{\mathcal{CS}} US \parallel TS \cup t} \; [\text{INSERT}]$$

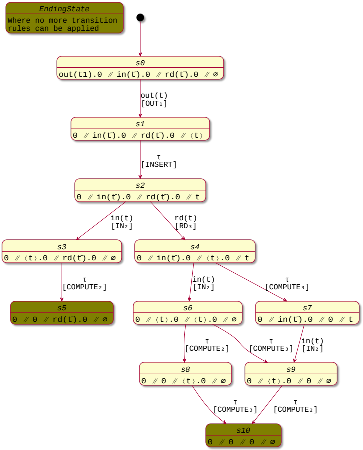
$$\frac{U_i + U_i' \xrightarrow{E'}_{\mathcal{U}} U_i'' \qquad TS \xrightarrow{E''}_{\mathcal{TS}} TS' \qquad E = \gamma(E', E'')}{US \parallel U_i + U_i' \parallel US' \parallel TS \xrightarrow{E}_{\mathcal{CS}} US \parallel U_i'' \parallel US' \parallel TS'} \; [\text{CHOICE}_i]$$

where the partial funtion $\gamma(\cdot, \cdot)$ is defined as follows:

- $\gamma(!out(t), ?out(t)) = out(t)$
- $\gamma(?in(\bar{t}), !in(t)) = in(t)$
- $\gamma(?rd(\bar{t}), !rd(t)) = rd(t)$

Example: Out-In-Rd, *unordered*

Example where $s_0 = \texttt{out}(t) \cdot 0 \parallel \texttt{in}(\bar{t}) \cdot 0 \parallel \texttt{rd}(t) \cdot 0 \parallel \emptyset$



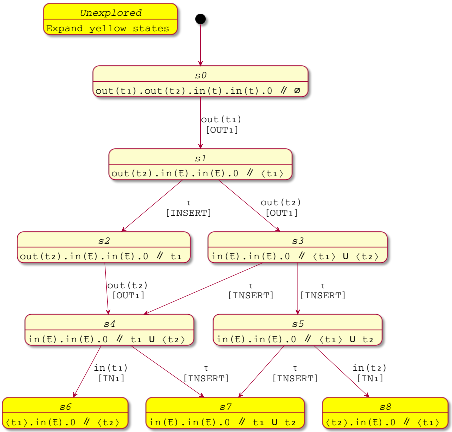
! Zoomable image here

Next in Line...

Exercise 5-1: Out-Out-In-In, *unordered* I

1. Clone the Lab-5 GitLab repository:
   `https://gitlab.com/pika-lab/courses/ds/aa1819/lab-5`
2. Consider the system with $s_0 = \mathtt{out}(t_1) \cdot \mathtt{out}(t_2) \cdot \mathtt{in}(\overline{t}) \cdot \mathtt{in}(\overline{t}) \cdot 0 \parallel \emptyset$, where $t_1, t_2 \in \overline{t}$, s.t. the LINDA semantics defined before
3. Complete the state graph according to the transition ruled defined before and include the state graph image in your `README.md` file
   - web editor here: http://www.plantuml.com/plantuml/uml/ZP...SDad
4. On the state graph, edges' labels should show both:
   - the event raised by the transition
   - the transition rule justifying the edge
5. You can rely on the tools listed on slide 56 for your exercise
6. Commit & push your `README.md` file

Exercise 5-1: Out-Out-In-In, *unordered*



! Zoomable image here

Outline

Ordered VS unordered semantics for LINDA

- The tuple spaces semantics defined so far is known as the unordered semantics of LINDA
  - long story short: if n tuples are orderly `out`'d by an agent, you cannot assume them to be inserted into the tuple space in the same order
  - agents cannot use tuple spaces as counters $\Longrightarrow$ no Turing equivalence
- The unordered semantics essentially states the `out` operation is *asynchronous*
- We will now provide an ordered semantics where the `out` primitive is *synchronous*, making the resulting coordinated systems Turing equivalent

Ordered VS unordered semantics for Linda

- The tuple spaces semantics defined so far is known as the unordered semantics of Linda
    - long story short: if n tuples are orderly `out`'d by an agent, you cannot assume them to be inserted into the tuple space in the same order
    - agents cannot use tuple spaces as counters $\Longrightarrow$ no Turing equivalence
- The unordered semantics essentially states the `out` operation is *asynchronous*
- We will now provide an ordered semantics where the `out` primitive is *synchronous*, making the resulting coordinated systems Turing equivalent

Ordered VS unordered semantics for LINDA

- The tuple spaces semantics defined so far is known as the unordered semantics of LINDA
  - long story short: if n tuples are orderly `out`'d by an agent, you cannot assume them to be inserted into the tuple space in the same order
  - agents cannot use tuple spaces as counters $\Longrightarrow$ no Turing equivalence

- The unordered semantics essentially states the `out` operation is *asynchronous*

- We will now provide an ordered semantics where the `out` primitive is *synchronous*, making the resulting coordinated systems Turing equivalent

Ordered VS unordered semantics for LINDA

- The tuple spaces semantics defined so far is known as the unordered semantics of LINDA
  - long story short: if n tuples are orderly `out`'d by an agent, you cannot assume them to be inserted into the tuple space in the same order
  - agents cannot use tuple spaces as counters $\Longrightarrow$ no Turing equivalence

- The unordered semantics essentially states the `out` operation is *asynchronous*

- We will now provide an ordered semantics where the `out` primitive is *synchronous*, making the resulting coordinated systems Turing equivalent

Ordered VS unordered semantics for LINDA

- The tuple spaces semantics defined so far is known as the unordered semantics of LINDA
  - long story short: if n tuples are orderly `out`'d by an agent, you cannot assume them to be inserted into the tuple space in the same order
  - agents cannot use tuple spaces as counters $\Longrightarrow$ no Turing equivalence

- The unordered semantics essentially states the `out` operation is *asynchronous*

- We will now provide an ordered semantics where the `out` primitive is *synchronous*, making the resulting coordinated systems Turing equivalent

Next in Line. . .

Ordered semantics for Linda

1. Imagine the syntax for tuple spaces was defined without a means for expressing pending tuples to be inserted:

$$TS \quad ::= \quad (T \cup TS) \mid \emptyset$$

2. Then, imagine transition rule [INSERT] was never defined, neither for $\mathcal{TS}$ nor for $\mathcal{CS}$
3. Then, imagine transition rule [SCHEDULE] was defined in the following way for $\mathcal{TS}$:

$$TS \quad \xrightarrow{?out(t)}_{\mathcal{TS}} \quad (TS \cup t)$$

4. Finally, imagine transition rule [OUT$_i$] was defined in the following way for $\mathcal{CS}$:

$$\frac{\mathtt{out}(t) \cdot U_i \xrightarrow{!out(t)}_{\mathcal{U}} U_i \qquad TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup t}{US \parallel \mathtt{out}(t) \cdot U_i \parallel US' \parallel TS \xrightarrow{out(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup t}$$

Ordered semantics for LINDA

1. Imagine the syntax for tuple spaces was defined without a means for expressing pending tuples to be inserted:

$$TS \quad ::= \quad (T \cup TS) \mid \emptyset$$

2. Then, imagine transition rule [INSERT] was never defined, neither for $\mathcal{TS}$ nor for $\mathcal{CS}$

3. Then, imagine transition rule [SCHEDULE] was defined in the following way for $\mathcal{TS}$:

$$TS \quad \xrightarrow{?out(t)}_{\mathcal{TS}} \quad (TS \cup t)$$

4. Finally, imagine transition rule [OUT$_i$] was defined in the following way for $\mathcal{CS}$:

$$\frac{\mathtt{out}(t) \cdot U_i \xrightarrow{!out(t)}_{\mathcal{U}} U_i \qquad TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup t}{US \parallel \mathtt{out}(t) \cdot U_i \parallel US' \parallel TS \xrightarrow{out(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup t}$$

Ordered semantics for LINDA

1. Imagine the syntax for tuple spaces was defined without a means for expressing pending tuples to be inserted:

$$TS \quad ::= \quad (T \cup TS) \mid \emptyset$$

2. Then, imagine transition rule [INSERT] was never defined, neither for $\mathcal{TS}$ nor for $\mathcal{CS}$

3. Then, imagine transition rule [SCHEDULE] was defined in the following way for $\mathcal{TS}$:

$$TS \quad \xrightarrow{?out(t)}_{\mathcal{TS}} \quad (TS \cup t)$$

4. Finally, imagine transition rule [OUT$_i$] was defined in the following way for $\mathcal{CS}$:

$$\frac{\mathrm{out}(t) \cdot U_i \xrightarrow{!out(t)}_{\mathcal{U}} U_i \qquad TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup t}{US \parallel \mathrm{out}(t) \cdot U_i \parallel US' \parallel TS \xrightarrow{out(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup t}$$

Ordered semantics for LINDA

1. Imagine the syntax for tuple spaces was defined without a means for expressing pending tuples to be inserted:

$$TS \quad ::= \quad (T \cup TS) \mid \emptyset$$

2. Then, imagine transition rule [INSERT] was never defined, neither for $\mathcal{TS}$ nor for $\mathcal{CS}$
3. Then, imagine transition rule [SCHEDULE] was defined in the following way for $\mathcal{TS}$:

$$TS \quad \xrightarrow{?out(t)}_{\mathcal{TS}} \quad (TS \cup t)$$

4. Finally, imagine transition rule [OUT_i] was defined in the following way for $\mathcal{CS}$:

$$\frac{\text{out}(t) \cdot U_i \xrightarrow{!out(t)}_{\mathcal{U}} U_i \qquad TS \xrightarrow{?out(t)}_{\mathcal{TS}} TS \cup t}{US \parallel \text{out}(t) \cdot U_i \parallel US' \parallel TS \xrightarrow{out(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup t}$$

Next in Line...

Exercise 5-2: Out-Out-In-In, *ordered* I

1. Go on working on your local clone of the Lab-5 GitLab repository

2. Consider the system with $s_0 = \texttt{out}(t_1) \cdot \texttt{out}(t_2) \cdot \texttt{in}(\bar{t}) \cdot \texttt{in}(\bar{t}) \cdot 0 \parallel \emptyset$, where $t_1, t_2 \in \bar{t}$, s.t. the ordered LINDA semantics defined before

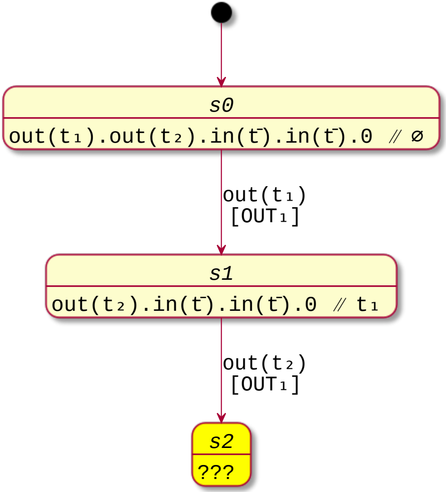
3. Complete the state graph according to the transition ruled defined before and include the state graph image in your `README.md` file
   - web editor here: http://www.plantuml.com/plantuml/uml/XO...q0

4. On the state graph, edges' labels should show both:
   - the event raised by the transition
   - the transition rule justifying the edge

5. You can rely on the tools listed on slide 56 for your exercise

6. Commit & push your `README.md` file

Exercise 5-2: Out-Out-In-In, *ordered* II

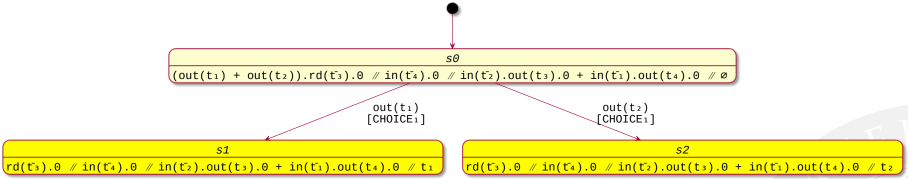


Next in Line...

Exercise 5-3: Choice, *ordered* I

1. Go on working on your local clone of the Lab-5 GitLab repository
2. Consider the system with $s_0 = (\mathtt{out}(t_1) + \mathtt{out}(t_2)) \cdot \mathtt{rd}(\bar{t}_3) \cdot 0 \parallel \mathtt{in}(\bar{t}_4) \cdot 0 \parallel (\mathtt{in}(\bar{t}_2) \cdot \mathtt{out}(t_3) + \mathtt{in}(\bar{t}_1) \cdot \mathtt{out}(t_4)) \cdot 0 \parallel \emptyset$, where $t_i \in \bar{t}_i$ for all $i = 1, \ldots, 4$, s.t. the ordered LINDA semantics defined before
3. Complete the state graph according to the transition ruled defined before and include the state graph image in your `README.md` file
   - web editor here: http://www.plantuml.com/plantuml/uml/IO...m00
4. On the state graph, edges' labels should show both:
   - the event raised by the transition
   - the transition rule justifying the edge
5. You can rely on the tools listed on slide 56 for your exercise
6. Commit & push your `README.md` file

Exercise 5-3: Choice, *ordered* II



! Zoomable image here

Let's further simplify Users: abstracting [COMPUTE] away

1. Imagine the syntax for users was simply defined as follows:

$$\begin{array}{rcl} U & ::= & \mathtt{out}(T) \cdot U \mid \mathtt{in}(TT) \cdot U \\ & \mid & \mathtt{rd}(TT) \cdot U \mid (U + U) \mid 0 \end{array}$$

2. Then, imagine transition rule [COMPUTE] and [COMPUTE$_i$] where never defined, neither for $\mathcal{TS}$ nor for $\mathcal{CS}$
3. Then, imagine rule [IN$_i$] was defined in the following way for $\mathcal{CS}$:

$$\frac{\mathtt{in}(\bar{t}) \cdot U_i \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \qquad t \in \bar{t}}{US \parallel \mathtt{in}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{in(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS}$$

4. Finally, imagine rule [RD$_i$] was defined in the following way for $\mathcal{CS}$

$$\frac{\mathtt{rd}(\bar{t}) \cdot U_i \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \qquad t \in \bar{t}}{US \parallel \mathtt{rd}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{rd(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup t}$$

Let's further simplify Users: abstracting [COMPUTE] away

1. Imagine the syntax for users was simply defined as follows:

$$
\begin{array}{rcl}
U & ::= & \mathtt{out}(T) \cdot U \mid \mathtt{in}(TT) \cdot U \\
 & \mid & \mathtt{rd}(TT) \cdot U \mid (U + U) \mid 0
\end{array}
$$

2. Then, imagine transition rule [COMPUTE] and [COMPUTE$_i$] where never defined, neither for $\mathcal{TS}$ nor for $\mathcal{CS}$

3. Then, imagine rule [IN$_i$] was defined in the following way for $\mathcal{CS}$:

$$
\frac{\mathtt{in}(\bar{t}) \cdot U_i \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \qquad t \in \bar{t}}{US \parallel \mathtt{in}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{in(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS}
$$

4. Finally, imagine rule [RD$_i$] was defined in the following way for $\mathcal{CS}$:

$$
\frac{\mathtt{rd}(\bar{t}) \cdot U_i \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \qquad t \in \bar{t}}{US \parallel \mathtt{rd}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{rd(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup t}
$$

Let's further simplify Users: abstracting [COMPUTE] away

1. Imagine the syntax for users was simply defined as follows:

$$\begin{array}{rcl} U & ::= & \mathtt{out}(T) \cdot U \mid \mathtt{in}(TT) \cdot U \\ & \mid & \mathtt{rd}(TT) \cdot U \mid (U + U) \mid 0 \end{array}$$

2. Then, imagine transition rule [COMPUTE] and [COMPUTE$_i$] where never defined, neither for $\mathcal{TS}$ nor for $\mathcal{CS}$
3. Then, imagine rule [IN$_i$] was defined in the following way for $\mathcal{CS}$:

$$\frac{\mathtt{in}(\bar{t}) \cdot U_i \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \qquad t \in \bar{t}}{US \parallel \mathtt{in}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{in(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS}$$

4. Finally, imagine rule [RD$_i$] was defined in the following way for $\mathcal{CS}$:

$$\frac{\mathtt{rd}(\bar{t}) \cdot U_i \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \qquad t \in \bar{t}}{US \parallel \mathtt{rd}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{rd(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup t}$$

Let's further simplify Users: abstracting [COMPUTE] away

1. Imagine the syntax for users was simply defined as follows:

$$\begin{array}{rcl} U & ::= & \texttt{out}(T) \cdot U \mid \texttt{in}(TT) \cdot U \\ & \mid & \texttt{rd}(TT) \cdot U \mid (U + U) \mid 0 \end{array}$$

2. Then, imagine transition rule [COMPUTE] and [COMPUTE$_i$] where never defined, neither for $\mathcal{TS}$ nor for $\mathcal{CS}$
3. Then, imagine rule [IN$_i$] was defined in the following way for $\mathcal{CS}$:

$$\frac{\texttt{in}(\bar{t}) \cdot U_i \xrightarrow{?in(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!in(t)}_{\mathcal{TS}} TS \qquad t \in \bar{t}}{US \parallel \texttt{in}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{in(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS}$$

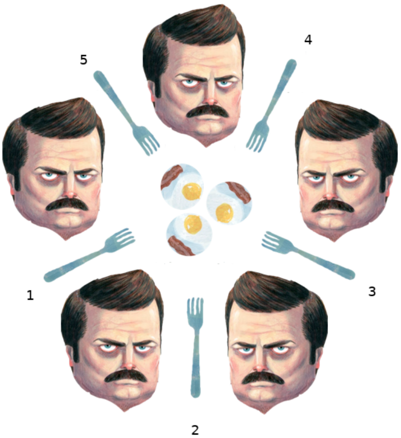
4. Finally, imagine rule [RD$_i$] was defined in the following way for $\mathcal{CS}$:

$$\frac{\texttt{rd}(\bar{t}) \cdot U_i \xrightarrow{?rd(\bar{t})}_{\mathcal{U}} U_i \qquad TS \cup t \xrightarrow{!rd(t)}_{\mathcal{TS}} TS \cup t \qquad t \in \bar{t}}{US \parallel \texttt{rd}(\bar{t}) \cdot U_i \parallel US' \parallel TS \cup t \xrightarrow{rd(t)}_{\mathcal{CS}} US \parallel U_i \parallel US' \parallel TS \cup t}$$

Next in Line...

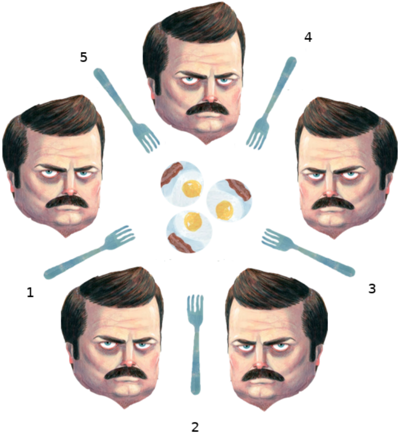
Dining Philosophers

- N philosophers are sitting on a round table spending their time eating and thinking
- N forks are on the table: philosopher i has fork i on his left and form $(i+1)\%N$ on his right
- In order to eat, philosopher i must be holding both fork i and fork $(i+1)\%N$
- There is no way for a philosopher to take more than one fork at a time



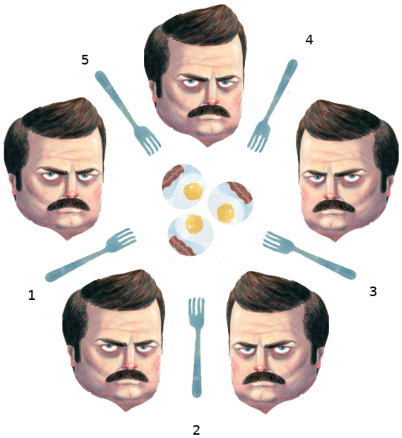
Dining Philosophers

- N philosophers are sitting on a round table spending their time eating and thinking
- N forks are on the table: philosopher i has fork i on his left and form $(i+1)\%N$ on his right
- In order to eat, philosopher i must be holding both fork i and fork $(i+1)\%N$
- There is no way for a philosopher to take more than one fork at a time



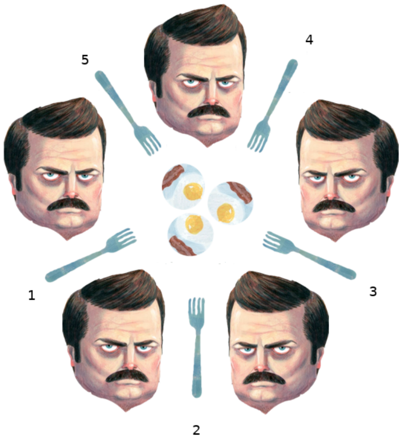
Dining Philosophers

- N philosophers are sitting on a round table spending their time eating and thinking
- N forks are on the table: philosopher i has fork i on his left and form $(i+1)\%N$ on his right
- In order to eat, philosopher i must be holding both fork i and fork $(i+1)\%N$
- There is no way for a philosopher to take more than one fork at a time



Dining Philosophers

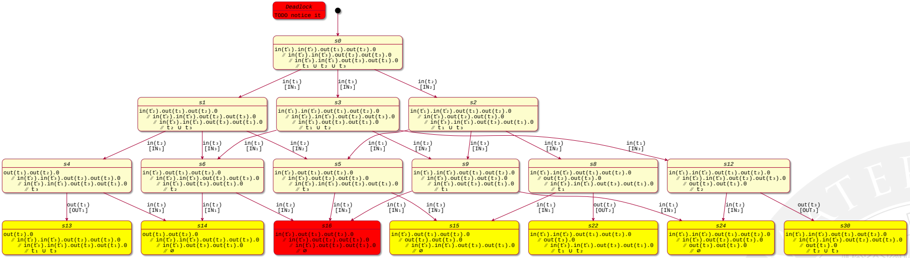
- N philosophers are sitting on a round table spending their time eating and thinking
- N forks are on the table: philosopher i has fork i on his left and form $(i+1)\%N$ on his right
- In order to eat, philosopher i must be holding both fork i and fork $(i+1)\%N$
- There is no way for a philosopher to take more than one fork at a time



Exercise 5-4: Three Dining Philosophers, *ordered* I

1. Go on working on your local clone of the Lab-5 GitLab repository
2. Consider the system with
   $\mathtt{in}(\bar{t}_1) \cdot \mathtt{in}(\bar{t}_2) \cdot \mathtt{out}(t_1) \cdot \mathtt{out}(t_2) \cdot 0 \parallel \mathtt{in}(\bar{t}_2) \cdot \mathtt{in}(\bar{t}_3) \cdot \mathtt{out}(t_2) \cdot \mathtt{out}(t_3) \cdot 0 \parallel \mathtt{in}(\bar{t}_3) \cdot \mathtt{in}(\bar{t}_1) \cdot \mathtt{out}(t_3) \cdot \mathtt{out}(t_1) \cdot 0 \parallel t_1 \cup t_2 \cup t_3$, where $t_i \in \bar{t}_i$ for all $i = 1, \ldots, 3$, s.t. the ordered LINDA semantics defined before
3. Complete the state graph according to the transition ruled defined before and include the state graph image in your `README.md` file
   - web editor here: http://www.plantuml.com/plantuml/uml/R8...jy0
4. On the state graph, edges' labels should show both:
   - the event raised by the transition
   - the transition rule justifying the edge
5. You can rely on the tools listed on slide 56 for your exercise
6. Commit & push your `README.md` file

Exercise 5-4: Three Dining Philosophers, *ordered* II



! SVG zoomable image available here

Deadlocks

Deadlock

A situation where N processes must access some shared resources in a mutually exclusive way and all of them get stuck, waiting for some other process to release a resource

Deadlock on a state graph

Deadlocks are those states on a state graph having no outgoing edge, i.e., those states where no transition rule can be applied (red states on the image)

Deadlocks

Deadlock

A situation where N processes must access some shared resources in a mutually exclusive way and all of them get stuck, waiting for some other process to release a resource

Deadlock on a state graph

Deadlocks are those states on a state graph having no outgoing edge, i.e., those states where no transition rule can be applied (red states on the image)

Next in Line...

Formalising other primitives

1. Imagine the syntax for users was extended as follows:

$$
\begin{array}{rcl}
U & ::= & \mathtt{out}(T) \cdot U \\
& | & \mathtt{in}(TT) \cdot U \\
& | & \mathtt{rd}(TT) \cdot U \\
& | & \mathtt{no}(TT) \cdot U \\
& | & \mathtt{inp}(TT) \ ?\ U : U \\
& | & \mathtt{rdp}(TT) \ ?\ U : U \\
& | & \mathtt{nop}(TT) \ ?\ U : U \\
& | & (U + U) \mid 0
\end{array}
$$

2. Subject to the following axioms:

$$(X \ ?\ T : F) \cdot Y \ \equiv \ X \ ?\ (T \cdot Y) : (F \cdot Y)$$

Formalising other primitives

1. Imagine the syntax for users was extended as follows:

$$
\begin{array}{rcl}
U & ::= & \mathtt{out}(T) \cdot U \\
 & | & \mathtt{in}(TT) \cdot U \\
 & | & \mathtt{rd}(TT) \cdot U \\
 & | & \mathtt{no}(TT) \cdot U \\
 & | & \mathtt{inp}(TT) \; ? \; U : U \\
 & | & \mathtt{rdp}(TT) \; ? \; U : U \\
 & | & \mathtt{nop}(TT) \; ? \; U : U \\
 & | & (U + U) \mid 0
\end{array}
$$

2. Subject to the following axioms:

$$(X \; ? \; T : F) \cdot Y \quad \equiv \quad X \; ? \; (T \cdot Y) : (F \cdot Y)$$

Exercise 5-5: Writing transition rules

1. Go on working on your local clone of the Lab-5 GitLab repository

2. Try extending the $\mathcal{CS}$ definition with new transition rules formally specifying the semantics of the `no`, `inp`, `rdp`, and `nop` primitives

3. You can rely on the tools listed on slide 56 for your exercise

4. Commit & push your `README.md` file

Exercise 5-5: Example for the nop primitive

For instance, the nop primitive could be defined by means of the following transition rules:

$$\frac{\forall t \in \bar{t} : TS \neq TS' \cup t}{US \parallel \mathtt{nop}(\bar{t}) \ ? \ U : U' \parallel US' \parallel TS \xrightarrow{nop(\bar{t},\top)}_{\mathcal{CS}} US \parallel U \parallel US' \parallel TS} \ [\text{NOP-T}_i]$$

$$\frac{\exists t \in \bar{t} : TS = TS' \cup t}{US \parallel \mathtt{nop}(\bar{t}) \ ? \ U : U' \parallel US' \parallel TS \xrightarrow{nop(\bar{t},\bot)}_{\mathcal{CS}} US \parallel U' \parallel US' \parallel TS} \ [\text{NOP-F}_i]$$

Outline

Exercise 5-6: Coffee Machine I

1. You must perform and end-to-end formalisation of a C/DS composed by a coffee machine and the user interacting with it
2. The system must take into account the following requirements:
   - Any coffee machine simply performs the following sort of actions:
     - it initially waits for coins to be inserted
     - it then checks if coins are sufficient
     - it may optionally give change back to the user
     - it serves the coffee to the user
     - it finally waits for the user to take the coffee
   - In turn, any user can perform the following actions:
     - he/she can walk around
     - he/she can chat with some friends
     - he/she can insert coins into the coffee machine in order to pay
     - it can take the coffee the machine has eventually served

Exercise 5-6: Coffee Machine II

- Of course, coffee machines can stop waiting for money only if some user pays Similarly, they can stop waiting for the coffee to be taken only if some user takes it

3. Your formalisation must provide an interpretation and a semantics for the following formula:

$$s_0 = (\mathtt{chat} + \mathtt{walk}) \cdot \mathtt{insert} \cdot (\mathtt{walk} + \mathtt{chat}) \cdot \mathtt{take} \cdot 0$$
$$\parallel \mathtt{waitCoin} \cdot \mathtt{check} \cdot (\mathtt{change} \cdot \mathtt{coffee} + \mathtt{coffee}) \cdot \mathtt{waitUser} \cdot 0$$

4. Draw the state graph of the system having s_0 as initial state, according to your semantics
5. You can rely on the tools listed on slide 56 for your exercise
6. Commit & push your `README.md` file

Outline

Useful tools

- Web-based markdown editor supporting LaTeX syntax for formulas
  - https://upmath.me
- Web-based PlantUML editor for designing State Charts (and other UML diagrams)
  - http://plantuml.com/plantuml
- Web-based tool for converting LaTeX formulas into Unicode strings
  - http://vikhyat.net/projects/latex_to_unicode

Process algebrae fundamentals

Distributed Systems / Technologies
Sistemi Distribuiti / Tecnologie

Giovanni Ciatto Andrea Omicini
giovanni.ciatto@unibo.it andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2019/2020