Overview

- Overview of problems and approaches
- Preliminaries: Basic concepts and models
  - Labelled transition systems (LTS)
  - Input-Output Automata (IOA)
  - Communication Finite State Machines (CFSM)
- Avoiding unspecified receptions
- Deriving submodule specifications
- Deriving protocol specifications from service specifications
Designing distributed and communication systems:

Problems

- Problems common to system design in general:
  - How to specify the requirements
  - Take architectural constraints and existing components into account during design
  - Interface documentation, especially dynamic aspects
  - Validation and verification

- Specific to distributed systems:
  - Flexibility for the physical distribution of functions
  - To define a protocol that provides a specified service
  - Complexity due to the parallel operation of the physically distributed components (“state space explosion”)
  - Need for compatibility at interfaces and at the levels of protocols

General orientation of this tutorial

- Validation and verification require precise specification of requirements and designs
- Emphasis on the automation of certain software development steps (Note: This requires formally defined meaning of specification languages)
- Emphasis on requirements and design phases
- Presentation of very simple specification languages, in order to explain the concepts that are important for the design of distributed systems (Note: For practical purposes, one requires more powerful languages, especially concerning interaction parameters and data types)
Topics covered in this tutorial

- Review of the basic concepts of LTS, IOA and CFSM with emphasis on behavior comparison in view of conformance
- A simple protocol design method for avoiding unspecified receptions
- How to derive the specification of a system component in such a way that its composition with a given component has certain given properties (Note: This is relevant for design of communication gateways and system controllers, and for the design with existing components)
- How to derive the specification of a protocol that provides the service described by a given expression of distributed sequencing rules (e.g. distributed LTS, Petri net, or similar)

Overview

- Overview of problems and approaches
- Preliminaries: Basic concepts and models
  - Labelled transition systems (LTS)
  - Input-Output Automata (IOA)
  - Communication Finite State Machines (CFSM)
- Avoiding unspecified receptions
- Deriving submodule specifications
- Deriving protocol specifications from service specifications
**Labelled Transition Systems**

**Examples**

- **A**
  - State 1
  - Transition: a → 2
    - Label: Open connection
  - Transition: b → 2
    - Label: Close connection
  - State 2
- **B**
  - State 1
  - Transition: a → 2
    - Label: Open connection
  - Transition: b → 2
    - Label: Close connection
  - State 2
- **A ∞ B**
  - State 1
    - Transition: a → 1
      - Label: Open connection
    - Transition: b → 3
      - Label: Busy
  - State 2
    - Transition: c → 2
      - Label: Close connection
  - State 3
    - Transition: d → 1
      - Label: Send data

**Composition:**
- Rendezvous for equally named actions

---

**Interesting concepts**

- Final state
- Home state
- Deadlock
- Partial deadlock
- Loops: useful – not useful (livelock)
- Operations on machines:
  - Composition
  - Abstraction (action hiding, projection)
  - Comparison of behaviors
Composition and interfaces

**Interface**
- An interface is a port where a subset of the interactions of the machine take place
- “compatible” ports of different machines can be used for interconnection (composition)

```
A
| c |
- a, b

B
| d |
- c, f

A
| e |
- a, b

B
| f |
- c, f
```

Coupling of
- equally named actions, or
- explicitly mapped actions

**Abstraction**

**Example:** hide \{a, b\} in A \(\bowtie\) B

```
Architecture

A
| c |
- a, b

B
| d |
- e, f

Behavior

1.1

\(\bowtie\)

2.2

\(\bowtie\)

\(\bowtie\)
```

equivalent to … ?
Behavior comparisons

- Different criteria for comparison
  - Possible sequences of interactions (execution traces)
    - trace equivalence, trace inclusion
  - As above, plus possible blocking behaviors
    - testing equivalence, observational equivalence, reduction, extension
  - Comparison based on states and transitions
    - (bi-) simulation
  - Consider also timing/performance behavior

Behavior based on trace semantics

- Basic paradigm:
  
Behavior of a machine = set of its possible execution traces

- Note: A set of traces $T$ can be characterized by a constraint $C_T$ that is satisfied by each trace in the set (e.g. satisfy a specific regular expression)

- Note: Given $T_1$ and $T_2$, $T_2 \subseteq (is\ subset\ of)\ T_1$ iff $C_{T_2} \Rightarrow (implies)\ C_{T_1}$
Comparison of specifications: Conformance

- **Question:** Given a system design containing a module A which is specified to have properties $P_A$, is it possible to replace this module by a module A’ with properties $P_{A'}$?

- **Answer:** Yes, iff $P_{A'} \Rightarrow P_A$

Then we say that A’ conforms to A

- In the case of trace semantics:
  - P is of the form: “each trace satisfies C “
  - A’ conforms to A iff $C_{A'} \Rightarrow C_A$ iff the set of traces of A’ is included in the set of traces of A

Overview

- Overview of problems and approaches
- Preliminaries: Basic concepts and models
  - Labelled transition systems (LTS)
  - Input-Output Automata (IOA)
  - Communication Finite State Machines (CFSM)
- Avoiding unspecified receptions
- Deriving submodule specifications
- Deriving protocol specifications from service specifications
Specification with assumptions and guarantees (A/G)

- **Form of a specification of a component C:** If the environment satisfies the assumption $A_C$ then the component C will guarantee the property $G_C$.

  Note: For the case that the assumption is not satisfied, the specification does not say anything about the desired behavior.

- **This means:** the property that must be satisfied by C has the form “$A_C \implies G_C$”

---

**Example (an electrical component)**

Example: **Electricity counter (component C)**
- I1: electricity input (assumption: less than 100 Amps, less than 300 Volts)
- I2: electricity output (guarantee: same current and voltage as at I1)
- I3: read-out interface (guarantee: the value provided is equal to the energy (i.e. product of current*voltae*time) that passed through the interface I2)
Software example: specification of a procedure

- **Pre- and Post-conditions**
  - They define assumptions about input parameters, and guarantees on output parameters, respectively
  - They apply when a software procedure is called

Conformance for A/G specifications

- When does specification $A'$ conform to specification $A$?
  - Iff $P_{A'} \Rightarrow (\text{implies}) P_A$
  - Iff $(A_{A'} \Rightarrow G_{A'}) \Rightarrow (A_A \Rightarrow G_A)$
  - Iff $(A_A \Rightarrow A_{A'})$ and $(G_A \Rightarrow G_A)$
Distinction of input and output

Examples

- In Lotos: A single interaction of a process may have input and output parameters
  - E.g. “open connection” primitive with user password input and session-Id output
- OO software design: Input and output parameters of a procedures (method)
  - Lotos interactions and a method call may be abstractly modelled as a single transition of an LTS
- In contrast, the distinction of input and output in IOA is mainly concerned with the question which component controls when an interactions takes place

Input/Output Automata

- Nature of input/output (*simultaneous input/output, but not rendezvous*)
  - Output: time and parameters of an interaction are determined by the system component producing the output
  - Input: The component receiving the interaction cannot influence the time nor parameter values
- Specification of component behavior
  - Output: The specification gives guarantees about timing and parameter values
  - Input: The specification may make assumptions about timing of inputs and the received parameter values
Example

The meaning of A depends on whether a, b, and c are input or output
- An output may be performed in a state only if such a transition starts in that state
- If an input arrives in some state and no transition is specified for this input in that state, then the assumption is not satisfied (this situation is called unspecified reception, probably a design error) -- there is no blocking as in the case of LTS

Note: Some authors only consider completely specified machines.
A non-trivial assumption implies a partially specified state machine

Comparing the behavior of IOAs

In general for A/G specifications:
A specification A’ conforms to a specification A iff \(( A_A => A_A’ ) \) and \(( G_A’ => G_A ) \)

For IOA, we consider any trace and consider the interactions in the order in which they occur. Assume that an initial trace T of interaction has already been checked, we then consider the next interaction. The following condition must be satisfied:
- Case of input:
  If it is allowed for A after the trace T (no unspec. Reception) then idem for A’
- Case of output:
  If it is allowed for A after the trace T then idem for A’

Note: For partially specified deterministic FSMs, this is called quasi-equivalence
Overview

- Overview of problems and approaches
- Preliminaries: Basic concepts and models
  - Labelled transition systems (LTS)
  - Input-Output Automata (IOA)
  - Communication Finite State Machines (CFSM)
- Avoiding unspecified receptions
- Deriving submodule specifications
- Deriving protocol specifications from service specifications

Communicating finite state machines (CFSM)

- They are like IOAs, except that the generation of output by one machine and the reception by the destination machine are separate events (interactions are queued in unbounded input queues)

For this case, intuition tells us:

- Q1 contains a or is empty
- Q2 contains c* or c*b
Composition for CFSM

- The state space of the composed system is in general unbounded. It includes the content of the queues. As a result:
  - Many interesting analysis questions are in general undecidable
    - Presence of deadlock
    - Boundedness of communication queues

Approaches to analysis

- Ad hoc proving of properties (e.g. Intuition for example above)
- Analysis up to some selected maximum queue length
- Limiting the model: Consider only Mealy machines (i.e. with alternate input and output)
  - Single input will lead to single output or internal looping
  - Never more than one non-processed interaction in the system
  - Externally, apply one input at a time
Overview

- Overview of problems and approaches
- Preliminaries: Basic concepts and models
  - Labelled transition systems (LTS)
  - Input-Output Automata (IOA)
  - Communication Finite State Machines (CFSM)
- Avoiding unspecified receptions
- Deriving submodule specifications
- Deriving protocol specifications from service specifications

Designing communicating systems: An example

The same example as above, only the direction of the b interaction is changed

A

\[ 1 \]
\[-a, -b \]
\[ 2 \]
\[ +c \]

B

\[ 1 \]
\[ +a, +b \]
\[ 2 \]
\[-c \]

Q1

\[ \rightarrow \]

Q2

Note: When A sends b, many c may still be in A’s input queue. They lead to unspecified receptions in state 1 or will be received in state 2.
Another example

- A and B do jointly either sequence a c (initiated by A) or sequence b e* d (initiated by B)

Reachability analysis (of example)

Reachability analysis means building a model of the composition (which may be infinite, because of unlimited queues)

** : unspecified receptions

Note: If one introduces transitions to consume the input and stay in the same state (as the implicit transitions in SDL), then one obtains a deadlock (both machines wait for an input and the queues are empty)
A tool for protocol design

- Zafilropoulos et al. built a tool for interactive protocol design (around 1980)
  - Starting with the initial states of the two machines, the designer adds (one by one) sending transitions to these machines. After each added sending transition, the tool automatically adds corresponding receiving transitions into the peer protocol entity for all states in which the message may be received; thus avoiding all unspecified receptions.

A method for protocol design

The following method applies when one has the specification of the temporal behavior of one side (in terms of an FSM model) and wants to obtain a communication protocol which synchronizes the other side, as much as possible, with the first one (i.e. when no message is in transit, the two sides should be in corresponding states).

Gouda’s design method

- Given: An FSM A representing the initial behavior of Side A.
- Construct an FSM B from A by interchanging “sending” with “receiving”.
  - If each state of the FSM is either a sending state (the outgoing transitions are all sending) or a receiving state (only receiving transitions), then B is the solution to the problem. All messages sent in a state X will be received by the other side in the same state X.
  - If there is a “mixed” state (with outgoing sending and receiving transitions) then the corresponding state at the other side is also mixed, and when these two states are reached by the two respective protocol entities, it may happen that both take a sending transition (which in general will lead to different corresponding states). This corresponds to a cross-over of messages within the medium, and the message will not be received in the state where they would normally be expected.
- For each mixed state, one has to make a design choice among the following alternatives:

Suite (1)

... design choice:
- the initiative of Side A has priority
- the initiative of Side B has priority
- both sides pull back to the respective mixed state and try again (there is the possibility of live-lock, which can be avoided by choosing random time delays before retries)

- Example:
Suite (2)

- State 1 is a mixed state
- If we give priority to Side A for this state, we should add the missing receptions which are required for the case of message cross-over in such a manner that Side A remains in the state in which it is and the other side goes into the same corresponding state (which is state 2 in this case). We obtain the following solution:

```
+ b
+ c
- a
+ b
3
2
4
- d
- e
```

**Side A (final version)**

```
+ a
- c
+ d
+ e
```

**Side B (final version)**

Overview

- Overview of problems and approaches
- Preliminaries: Basic concepts and models
  - Labelled transition systems (LTS)
  - Input-Output Automata (IOA)
  - Communication Finite State Machines (CFSM)
- Avoiding unspecified receptions
- Deriving submodule specifications
- Deriving protocol specifications from service specifications
Equation solving: Integer division

- Multiplication: $R_1 \times R_2 = ?$
- Equation solving: $R_1 \times X = R_3$
  - What is the value of $X$?
- Solution: definition of the division operation
  - Written “$X = R_3 / R_1$”
  - What does it mean?
    - $X = \text{biggest } Y \text{ such that } R_1 \times X \leq R_3$
    - Note: in many cases, there is no exact solution, that is, there is no $X$ such that $R_1 \times X = R_3$
      - For instance: $7 / 3 = 2$, and $3 \times 2 = 6 \leq 7$

Context of this talk

- Multiplication → Machine composition
- Division → Submodule construction
  ("equation solving")
- Example:
Overview of this section

- Machine composition and equation solving
- Applications
- Solution formulas
- A generalization: Relational databases
- The cases of synchronous finite state machines and labelled transition systems
- The case of specifications based on assumptions and guarantees: e.g. IO-Automata and asynchronous finite state machines
- Conclusions

Machine composition: concepts

- Port (where interactions take place)
- Machine structure (has a number of ports)
- Machine behavior (determines the order of interactions)
- Interface (where ports of different machines are interconnected)
Basic operations on machines

- Composition of several machines (interconnection structure)
  - All unconnected ports and internal interfaces are visible ports of the composition
  - The constraints on execution order defined by the dynamic behavior of all interconnected machines will be satisfied by the composition
- Hiding (abstraction): hiding the interactions taking place at one or several ports
- [some recent work on generalizing and formalizing these concepts]

Equation solving for machines

Given machine $R_1$ and specification $R_3$ for the behavior of the composition of $R_1$ with $X$, find a behavior of machine $X$ such that $\text{hide } a_3 \text{ in } (R_1 \bowtie X) \leq R_3$

- Meaning of $\leq$: set inclusion of possible execution sequences ("traces", i.e. sequences of interactions), also called trace inclusion
Applications of machine equation solving

- Communication protocols
  - Protocol design (Merlin-Bochmann, 1980)
  - Design of communication gateways
- Controller design for discrete event systems
- Component reuse, e.g. in software engineering

Communication protocol design

- Protocol entities $\text{PE}_1$ and $\text{PE}_2$ use the underlying service $S$ and provide the service $R_3$ to the users of the protocol
- $\text{PE}_1$ and $S$ are given
- $\text{PE}_2$ is to be found
- $R_1$ corresponds to $(\text{PE}_1 \otimes S)$
**Communication gateways**

- **Given**
  - desired end-to-end communication service $E2E$
  - Protocols in the two networks (different)

- **To be found**: gateway behavior (shown by red box)

---

**Controller design**

- Applications in process control, robotics, etc.
  - Also called “Discrete event systems” (a separate research community, e.g. [Ramage-Wonham, 1989] and many subsequent papers)
  - Distinction between non-controllable and controllable interactions (*like input/output*)
Component reuse

- A given submodule does not completely correspond to the specification of the system to be built
- An additional submodule to be built (and designed through equation solving) makes up the "difference"

Equation solving for labelled transition systems

- Rendezvous interactions
  - $a_3$: between $R_1$ and $X$
  - $a_2$: between $R_1$ and environment
  - $a_1$: between $X$ and environment

- Behavior definition
  - set of allowed execution sequences
  - e.g. for $X$: execution sequences over interactions at $a_3$ or $a_1$

- Chaos [term introduced by Hoare in CSP]
  - all execution sequences over an alphabet
  - e.g. over alphabet of $X$: written Ch[$\{a_1, a_3\}$]
The problem and its solution

- **Problem:** Find most general $X$ (largest set of execution sequences) such that
  \[
  \text{hide } a_3 \text{ in } (R_1 \circ X) \leq R_3
  \]

- **Solution:** $X = \text{Ch}\{a_1, a_3\} \setminus \text{(minus)}$
  any sequence that could lead to an observable execution sequence not in $R_3$, i.e.
  \[
  \text{hide } a_2 \text{ in } (R_1 \circ (\text{Ch}\{a_1, a_2\} \setminus R_3))
  \]

A comment

- Since all execution sequences of $X$ must go in interaction with $R_1$ and $R_3$, we may replace the chaos for $X$ with all sequences that are obtained by the composition of $R_1$ and $R_3$, that is [Merlin and Bochmann, 1980]

- **Solution:** $X = \text{hide } a_2 \text{ in } (R_1 \circ R_3) \setminus \text{(minus)}$
  \[
  \text{hide } a_2 \text{ in } (R_1 \circ (\text{Ch}\{a_1, a_2\} \setminus R_3))
  \]
An example

- $\alpha_2 = \{a, b\}$, $\alpha_1 = \emptyset$, $\alpha_3 = \{c, d\}$
- $R_3 = \{ab\}^*$, $R_1 = \{acb, adb, ada\}^*$
  - Notation: $W^*$ = set of all prefixes of the strings in the set $W$,
  e.g. $\{ab, ac\}^* = \{\varepsilon, a, ab, ac\}$
- Solution: $X = \text{hide } a_2 \text{ in } (R_1 \otimes R_3) \setminus (\text{minus})$
  - hide $a_2$ in $(R_1 \otimes R_3)$ = $\{\varepsilon, c\}$
  - $(\text{Ch} \{\{a_1, a_2\}\} \setminus R_3) = \{b, bx, aa, aax, abx\}$ where $x$ represents any non-empty sequence over the alphabet $\{a, b\}$
  - $R_1 \otimes (\text{Ch} \{\{a_1, a_2\}\} \setminus R_3) = \{ada\}$
  - hide $a_2$ in $(R_1 \otimes (\text{Ch} \{\{a_1, a_2\}\} \setminus R_3)) = \{d\}$
  - $X = \{\varepsilon, c\}$

Equation solving for synchronous automata

- Synchronous communication
  - Simultaneous interactions at all interfaces; at each clock pulse, there is a vector of interactions
- Behavior definition
  - set of allowed sequences of interaction vectors
  - e.g. for $X$: the interaction vectors include interactions at $a_3$ and $a_1$
Solution of equation solving

- **Identical form of formulas**
- Meaning of operators have changed
  - $\infty$: *synchronous* composition
  - *hide* operator
    - ignores a component of the vector

[Yevtushenko et al., 1999]

---

Relational database (intro)

- A DB is a set of *relations*
- A relation is a table
  - Each column is an *attribute*
  - Each row is an “object”
  - An element at position $(a_i, o_k)$ in the table represents the value that object $o_k$ takes for attribute $a_i$
  - With each attribute $a_i$ is associated a set of possible values $D_i$
Relational database concepts

Formal definitions:
- Attributes: $A = \{a_1, a_2, \ldots, a_n\}$
- Attribute values: $D = \bigcup D_i$
- Relation over $A_r$ ($A_r \subseteq A$), written $R[A_r]$:
  (possibly infinite) set of mappings $T: A_r \rightarrow D$
  with $T(a_i) \in D_i$
  Note: each mapping corresponds to a row

Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>53</td>
<td>50000</td>
</tr>
<tr>
<td>Paul</td>
<td>50</td>
<td>60000</td>
</tr>
<tr>
<td>Alice</td>
<td>21</td>
<td>40000</td>
</tr>
<tr>
<td>Suzanne</td>
<td>35</td>
<td>50000</td>
</tr>
<tr>
<td>Bob</td>
<td>20</td>
<td>30000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>BigOne</td>
</tr>
<tr>
<td>Alice</td>
<td>BigOne</td>
</tr>
<tr>
<td>Fred</td>
<td>SmallOne</td>
</tr>
<tr>
<td>Suzanne</td>
<td>SmallOne</td>
</tr>
</tbody>
</table>
Relational operators

- **Projection**
  Given $R[A_x]$ and $A_x \subseteq A_r$, the projection of $R[A_x]$ onto $A_x$, written $\text{proj}_{A_x} (R)$, is a relation over $A_x$ with
  \[ T \in \text{proj}_{A_x} (R) \iff \exists T' \in R \text{ s.t. } \forall a_i \in A_x: T(a_i) = T'(a_i) \]

- **Join**
  Given $R_1[A_1]$ and $R_2[A_2]$, the join of $R_1$ and $R_2$, written $R_1 \bowtie R_2$, is a relation over $A_1 \cup A_2$ with
  \[ T \in (R_1 \bowtie R_2) \iff \text{proj}_{A_1}(T) \in R_1 \text{ and } \text{proj}_{A_2}(T) \in R_2 \]

- **Chaos**
  Given $A_x \subseteq A$, the chaos over $A_x$, written $\text{Ch}[A_x]$, is the relation which includes all mappings $T$: $A_x \rightarrow D$ with $T(a) \in D$.

Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>53</td>
<td>50000</td>
</tr>
<tr>
<td>Paul</td>
<td>50</td>
<td>60000</td>
</tr>
<tr>
<td>Alice</td>
<td>21</td>
<td>40000</td>
</tr>
<tr>
<td>Suzanne</td>
<td>35</td>
<td>50000</td>
</tr>
<tr>
<td>Bob</td>
<td>20</td>
<td>30000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>BigOne</td>
</tr>
<tr>
<td>Alice</td>
<td>BigOne</td>
</tr>
<tr>
<td>Suzanne</td>
<td>SmallOne</td>
</tr>
</tbody>
</table>

- $\text{Proj}_{\{\text{Project}\}} (R_2) = \text{BigOne, SmallOne}$
- $R_1 \bowtie R_2 = \text{Fred, BigOne, SmallOne}$
Equation solving for relational databases

- We consider
  - Three attributes $a_1, a_2, a_3$
  - Two relations $R_1 \{a_3, a_2\}, R_3 \{a_1, a_2\}$
- Problem: What is the biggest relation $X \{a_1, a_3\}$ satisfying $\text{proj}_{a_1, a_2}(R_1 \bowtie X) \subseteq R_3$
- Solution: $X = \text{Ch}\{a_1, a_3\} \setminus \text{proj}_{a_1, a_3}(R_1 \bowtie (\text{Ch}\{a_1, a_2\} \setminus R_3))$
- Proof: not difficult
  - Generalization to more complex attribute structures is also easy

An example

- $D_1 = \{\varepsilon\}$
- $D_2 = \{aa, ab, ba, bb\}$
- $D_3 = \{c, d\}$
- $X = \text{Ch}\{a_1, a_3\} \setminus \text{proj}_{a_1, a_3}(R_1 \bowtie (\text{Ch}\{a_1, a_2\} \setminus R_3))$

\[
\begin{array}{c}
\text{Ch}\{a_1, a_2\} \setminus R_3) \\
\text{R}_1 \bowtie (\text{Ch}\{a_1, a_2\} \setminus R_3)
\end{array}
\]

\[
\begin{array}{c}
\text{Ch}\{a_1, a_3\}
\end{array}
\]

Gregor v. Bochmann, University of Ottawa
A special case:
Trace specifications

- Attributes ↔ Interfaces
- \( D_i = I_i^* \) that is, all finite sequences of elements of \( I_i \), the possible interactions at the interface \( a_i \) (the “alphabet” at interface \( a_i \))
- Machine behavior ↔ Relation

Each row (DB object) represents a possible execution history (“trace”); the value for each attribute describes the interaction sequence occurring at the corresponding interface during that trace.

Synchrony constraint: The interaction sequences at the different interfaces for a given trace are of equal length.

Two sub-cases:
- synchronous operation (as above)
- interleaving semantics (below)

- Attributes ↔ Interfaces
- \( D_i = (I_i \cup \{\text{null}\})^* \) (as synchronous case, except that there is a real interaction at only one interface at a time; “interleaving semantics”)
- Machine behavior ↔ Relation

As in synchronous case, except that the “interleaving constraint” is satisfied for all mappings of a relation, that is, for any \( j \), the \( j \)-th element of \( T(a_i) \) is non-null for at most one attribute \( a_i \).
Algorithms for equation solving

Solution: \( X = \text{Ch}\{a_1, a_3\} \setminus \text{proj}\{a_1, a_3\}(R_1 \cap (\text{Ch}\{a_1, a_2\} \setminus R_3)) \)

- Algorithms for operations \( \cap, \setminus, \text{proj} \)
  - In general not possible (infinite sets of mappings)
  - For finite state models:
    - Polynomial complexity for \( \cap, \text{proj} \)
    - \( \text{proj} \) introduces non-determinism
    - \( \setminus \) requires conversion to deterministic models, which has exponential complexity
  - [Independently developed algorithms for LTS and synchronous machines are described in the references]

Systems with input and output

- Nature of input/output (non-rendezvous)
  - Output: time and parameters of an interaction are determined by the system component producing the output
  - Input: The component receiving the interaction cannot influence the time nor parameter values
- Specification of component behavior
  - Output: The specification gives guarantees about timing and parameter values
  - Input: The specification may make assumptions about timing of inputs and the received parameter values
Specification paradigms
with hypothesis and guarantees

- **Pre- and Post-conditions (software)**
  - They define hypotheses on input parameters, and guarantees on output parameters, respectively
  - apply when a software procedure is called

- **Finite state machines (state-deterministic)**
  - **Unspecified input**: hypothesis about the behavior of the environment: *this input will not occur when the machine is in this state*
  - **Nondeterministic output**: *don’t care*

Component specification and interconnection

- **Each attribute of a relation is either input or output**

- **Constraint on component interconnection**
  - **No output conflicts**: For each interface, there is only one connected component for which the corresponding attribute is output

- **For trace specifications**: **Unit delay constraint**
  - Output(s) at time t depend only on previous interactions of the same component (not on the input received at time t) [e.g. Broy, Lamport]
Conformance to specifications

- Given a specification R and a trace T
  1. Either $T \in R$ (we say T conforms to R) or ...
  2. T has wrong input: all prefixes of T up some time t conform to R, but there is wrong input at time (t+1)
  3. T has wrong output: similarly
  4. T has wrong input and output at the same time instant

- A component conforms to a specification R iff no trace T in which the component participates has wrong output in respect to R (cases 3 and 4 are excluded)
- Note: if a trace has wrong input, nothing can be assumed about wrong output at a later time instance

Equation solving for trace specifications with input/output

- Find most general specification X such that any trace T of the composition of $R_1$ and X has the following properties:
  - $\text{proj}_{\{a_1, a_2\}}(T)$ conforms to $R_3$
  - If $\text{proj}_{\{a_1, a_2\}}(T)$ has no wrong input in respect to $R_3$ then $\text{proj}_{\{a_2, a_3\}}(T)$ has no wrong input in resp. to $R_1$
Solution formula

- **Notation:**
  - $R^{WO(t)}$ = set of traces that have wrong output in respect to R at time instant $t$
  - $R^{WI(t)}$ : similarly for wrong input
  - $U_t$ : union over all values of $t$

- **Solution:**
  \[
  X = Ch\{a_1, a_3\} \setminus \text{proj}_{\{a_1, a_3\}} \cup_t (R_1 \cup R_3^{WO(t)} \cup R_1^{WI(t)} \cup R_3^{WI(t)} \cup R_3^{WO(t)})
  \]

Solution algorithms for I/O

- **Interleaving semantics**
  - **Simplification:** Never wrong input and output at the same time instant

- **IO-Automata**
  - Jawad Drissi (PhD thesis)

- **Communicating FSMs**
  - Yevtushenko and Petrenko

- **Synchronous FSMs**
  - Can be easily derived from above formula
Extensions of the specification formalisms

- More powerful specification languages
  - Petri nets, CSP, LOTOS, etc.
- Finer conformance relations
  - Safeness (no wrong interaction)
    - Trace inclusion (as discussed here)
  - Liveness - progress (some good interaction will occur)
    - Liveness [Thistle]
    - Absense of blockings [Tao, PhD thesis]
    - Optional and required progress [Drissi, PhD thesis]
- Real-time aspects
  - Timed automata [Grenoble; work on DES; Drissi, PhD thesis]

Overview

- Overview of problems and approaches
- Preliminaries: Basic concepts and models
  - Labelled transition systems (LTS)
  - Input-Output Automata (IOA)
  - Communication Finite State Machines (CFSM)
- Avoiding unspecified receptions
- Deriving submodule specifications
- Deriving protocol specifications from service specifications
Deriving a protocol specification from a given service specification: An example

**service A** (as above)

**architecture**

View of service A as an activity diagram
(with two swim lanes):

**Question:**
What should be the behavior of E1 and E2?

**Solution for the example**

**Service**

**Protocol**
Basic ideas

- **Sequential execution:** e.g. $a^{(1)} ; b^{(2)}$ becomes
  - in E1: $a^{(1)} ; \text{send}(2, x)$
  - in E2: $\text{receive}(1,x); b^{(2)}$

- **Independen parallelism:** no problem

- **Alternatives:** no problem if first actions of all alternatives are at the same site
  - Otherwise: Protocol for distributed choice required
    - e.g. logical token ring among all sites involved

  **Example:**

Additional info:

- About present state
- About calling stack

Who makes the decision between $c$ and $b$?
(when $b$ and $c$ are input, or when $c$ is output)

Further possibilities (a)

- More powerful languages for specifying services and protocols
  - LOTOS (includes recursive process calls; difficulty with $> \text{operator}$)
  - Petri nets (restricted classes)
    - With Registers
    - Colored Petri nets (includes token parameters; each transition has constraints on input token parameters and functions evaluating parameters of output tokens)
Further possibilities (b)

- Optimizations and implementation issues
  - Optimizing the number of messages for synchronization and parameter passing
  - Optimizing the localisation of registers
  - Re-design (after revisions of the service specification)
  - Implementation of the send and receive primitives, including data parameters:
    - Message encoding: XML, CORBA or ??
    - Underlying communication service: TCP, HTTP, CORBA, Java RMI or ??
    - Identification of parties (sites) involved

Overview

- Overview of problems and approaches
- Preliminaries: Basic concepts and models
  - Labelled transition systems (LTS)
  - Input-Output Automata (IOA)
  - Communication Finite State Machines (CFSM)
- Avoiding unspecified receptions
- Deriving submodule specifications
- Deriving protocol specifications from service specifications
- References and additional readings
Additional readings

- **Preliminaries: Basic concepts and models**
  - See course notes mentioned above

- **Avoiding unspecified receptions**

- **Deriving submodule specifications**

  - [Boch 01b] G. v. Bochmann, Submodule construction and supervisory control: a generalization, in Proc. of Int. Conf. on Implementation and Applications of Automata (invited paper), to be published as Springer Lecture Notes.
  - G. v. Bochmann, paper to be published at FORTE 2002

- **Deriving protocol specifications from service specifications**
Annex

- Some figures from

Extended Petri nets

Figure 1: Register Values and Token Location before and after Firing Transition in PNR
Service and corresponding protocol

Message optimization

Figure 4: Different Message Exchanges for the Same Resource Allocation
Example of re-synthesis

A realistic example: SE process