On Global Types and Multi-Party Sessions

Giuseppe Castagna

CNRS
Université Paris Diderot

(joint work with Mariangiola Dezani and Luca Padovani)

FMOODS & FORTE invited talk
DisCoTec 2011 - Reykjavík
Relating global descriptions distributed systems with sets of descriptions of their components is the subject of an important and long-standing research.

Recently, the community of behavioral types for web services has joined this effort.

The aim of this talk is to give an overview of the research done by these newcomers, addressing its goals and specificities.
Relating global descriptions distributed systems with sets of descriptions of their components is the subject of an important and long-standing research.

Recently, the community of behavioral types for web services has joined this effort.

The aim of this talk is to give an overview of the research done by these newcomers, addressing its goals and specificities.

For survey and pointers refer to the long version available online. The version in the proceedings focuses on technical content.
Context

Alice, Bob, and Charlie want to collaborate on the net.
Context

They do it by exchanging some messages

Alice, Bob, and Charlie want to collaborate on the net
send "hello" to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
    send ok to Bob;
    send ok to Alice
} else {
    send ok to Alice;
    send ok to Bob
}

receive ok from Alice;

They do it by exchanging some messages

Alice, Bob, and Charlie want to collaborate on the net
send "hello" to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
  send ok to Bob;
  send ok to Alice
} else {
  send ok to Alice;
  send ok to Bob
}

receive ok from Alice;

Several potential problems
send "hello" to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
    send ok to Bob;
    send ok to Alice
} else {
    send ok to Alice;
    send ok to Bob
}

receive ok from Alice;

Several potential problems

- Communication errors
Context

send "hello" to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
  send ok to Bob;
  send ok to Alice
} else {
  send ok to Alice;
  send ok to Bob
}

receive ok from Alice;

A string is sent but a Boolean is expected

Several potential problems
- Communication errors
Context

send "hello" to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
  send ok to Bob;
  send ok to Alice } else {
  send ok to Alice;
  send ok to Bob }

receive ok from Alice;

Several potential problems

- Communication errors

A string is sent but a Boolean is expected
send true to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
  send ok to Bob;
  send ok to Alice }
else {
  send ok to Alice;
  send ok to Bob }

receive ok from Alice;

Several potential problems

- Communication errors
send true to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
  send ok to Bob;
  send ok to Alice
} else {
  send ok to Alice;
  send ok to Bob
}

receive ok from Alice;

Several potential problems

- Communication errors
- Protocol errors
send true to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
  send ok to Bob;
  send ok to Alice }
else {
  send ok to Alice;
  send ok to Bob }

receive ok from Alice;

A message is sent but there is no corresponding reception

Several potential problems

- Communication errors
- Protocol errors
Context

send true to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
send ok to Bob;
send ok to Alice}
else {
send ok to Alice;
send ok to Bob}

receive ok from Alice;

A message is sent but there is no corresponding reception

Several potential problems

- Communication errors
- Protocol errors
send true to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
  send ok to Bob;
  send ok to Alice }
else {
  send ok to Alice;
  send ok to Bob }

receive ok from Alice;
receive ok from Charlie

Several potential problems
- Communication errors
- Protocol errors
Context

send true to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
    send ok to Bob;
send ok to Alice
} else {
    send ok to Alice;
send ok to Bob
}

receive ok from Alice;
receive ok from Charlie

Several potential problems

- Communication errors
- Protocol errors

There may be deadlocks
send true to Charlie;
receive ok from Charlie;
send ok to Bob
receive $x$ from Alice
if $x$ then {
    send ok to Bob;
send ok to Alice
} else {
    send ok to Alice;
send ok to Bob
} receive ok from Alice;
receive ok from Charlie

Several potential problems

- Communication errors
- Protocol errors

There may be deadlocks
Context

send true to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice
if $x$ then {
  send ok to Bob;
  send ok to Alice
} else {
  send ok to Alice;
  send ok to Bob
}

receive ok from Charlie;
receive ok from Alice

Several potential problems

- Communication errors
- Protocol errors
Context

There may be starvation

Several potential problems
- Communication errors
- Protocol errors
repeat
  send false to Charlie;
  receive $x$ from Charlie;
  until $x$; send ok to Bob

repeat
  receive $x$ from Alice;
  send $x$ to Alice;
  until $x$;
  send ok to Bob

receive ok from Charlie;
receive ok from Alice

There may be starvation
Here Bob starves

Several potential problems
- Communication errors
- Protocol errors
repeat
    send false to Charlie;
    receive $x$ from Charlie;
until $x$; send ok to Bob

repeat
    receive $x$ from Alice;
    send $x$ to Alice;
until $x$;
    send ok to Bob

receive ok from Charlie;
receive ok from Alice

These problems may be due to:

Several potential problems

- Communication errors
- Protocol errors
repeat
   send false to Charlie;
   receive $x$ from Charlie;
until $x$; send ok to Bob

repeat
   receive $x$ from Alice;
   send $x$ to Alice;
until $x$;
   send ok to Bob

receive ok from Charlie;
receive ok from Alice

These problems may be due to:
- Programming errors

Several potential problems
- Communication errors
- Protocol errors
repeat
send false to Charlie;
receive $x$ from Charlie;
until $x$; send ok to Bob

repeat
receive $x$ from Alice;
send $x$ to Alice;
until $x$;
send ok to Bob

receive ok from Charlie;
receive ok from Alice

These problems may be due to:
- Programming errors
- Software evolution

Several potential problems
- Communication errors
- Protocol errors
repeat
  send false to Charlie;
  receive $x$ from Charlie;
until $x$; send ok to Bob

repeat
  receive $x$ from Alice;
  send $x$ to Alice;
until $x$;
  send ok to Bob

receive ok from Charlie;
receive ok from Alice

These problems may be due to:
- Programming errors
- Software evolution
- Rogue participants

Several potential problems:
- Communication errors
- Protocol errors
**Global vs. Local specifications**

**Global specification**

- Do not describe (just) the behavior of each single participant.
- Describe the abstract global behavior of the protocol.
- Match against/Extract the behaviors of the participants.

**Example of global description**

Alice sends a Boolean to Charlie; either Charlie sends ok to Bob; Charlie sends ok to Alice; or Charlie sends ok to Alice; Charlie sends ok to Bob;
Global vs. Local specifications

Global specification

- Do not describe (just) the behavior of each single participant
Global vs. Local specifications

Global specification

- Do not describe (just) the behavior of each single participant
- Describe the abstract global behavior of the protocol
Global vs. Local specifications

Global specification

- Do not describe (just) the behavior of each single participant
- Describe the abstract global behavior of the protocol
- Match against/Extract the behaviors of the participants.

Example of global description

Alice sends a Boolean to Charlie; either Charlie sends ok to Bob; Charlie sends ok to Alice; or Charlie sends ok to Alice; Charlie sends ok to Bob;
Global vs. Local specifications

**Global specification**
- Do not describe (just) the behavior of each single participant
- Describe the abstract global behavior of the protocol
- Match against/Extract the behaviors of the participants.

**Example of global description**

Alice sends a Boolean to Charlie;
- either Charlie sends ok to Bob; Charlie sends ok to Alice;
- or Charlie sends ok to Alice; Charlie sends ok to Bob;
Global vs. Local specifications

The global specification is compact and synthetic

Example of global description

Alice sends a Boolean to Charlie;
either Charlie sends ok to Bob; Charlie sends ok to Alice;
or Charlie sends ok to Alice; Charlie sends ok to Bob;
Global vs. Local specifications

send true to Charlie;  
receive ok from Charlie;  
send ok to Bob

receive $x$ from Alice;  
if $x$ then {
    send ok to Bob;  
    send ok to Alice 
} else {
    send ok to Alice;  
    send ok to Bob 
}

switch
   | receive ok from Alice  ->  receive ok from Charlie 
   | receive ok from Charlie  ->  receive ok from Alice

Example of global description

Alice sends a Boolean to Charlie;  
either Charlie sends ok to Bob; Charlie sends ok to Alice;  
or Charlie sends ok to Alice; Charlie sends ok to Bob;
Global vs. Local specifications

send true to Charlie;
receive ok from Charlie;
send ok to Bob

receive $x$ from Alice;
if $x$ then {
  send ok to Bob;
send ok to Alice }
else {
  send ok to Alice;
send ok to Bob }

switch
  | receive ok from Alice -> receive ok from Charlie
  | receive ok from Charlie -> receive ok from Alice

It abstracts values

Example of global description

Alice sends a Boolean to Charlie;
either Charlie sends ok to Bob; Charlie sends ok to Alice;
or Charlie sends ok to Alice; Charlie sends ok to Bob;
Global vs. Local specifications

send true to Charlie;
receive ok from Charlie;
send ok to Bob

switch
| receive ok from Alice -> receive ok from Charlie
| receive ok from Charlie -> receive ok from Alice

receive $x$ from Alice;
if $x$ then {
send ok to Bob;
send ok to Alice }
else {
send ok to Alice;
send ok to Bob }

Example of global description

It abstracts choices
Alice sends Boolean to Charlie;
either Charlie sends ok to Bob; Charlie sends ok to Alice;
or Charlie sends ok to Alice; Charlie sends ok to Bob;

It abstracts values
Interest of global descriptions

Alice sends a Boolean to Charlie;
either Charlie sends ok to Bob; Charlie sends ok to Alice;
or Charlie sends ok to Alice; Charlie sends ok to Bob;

Given a distributed implementation that "satisfies" this global specification:

1. Every send of a given type is matched by a reception of the same type;
2. It should be easier to check the absence of deadlocks and starvation on global specifications.

We must ensure that all and only the expected synchronizations happen.

We need a theoretical framework for:
Defining global specifications,
Defining local specifications,
Relating them,
Proving their properties.
Alice sends a Boolean to Charlie;
either Charlie sends ok to Bob; Charlie sends ok to Alice;
or Charlie sends ok to Alice; Charlie sends ok to Bob;

Given a distributed implementation that “satisfies” this global specification:

1. Every send of a given type is matched by a reception of the same type;
Alice sends a Boolean to Charlie; 
either Charlie sends ok to Bob; Charlie sends ok to Alice; 
or Charlie sends ok to Alice; Charlie sends ok to Bob;

Given a distributed implementation that “satisfies” this global specification:

1. Every send of a given type is matched by a reception of the same type;
Interest of global descriptions

Alice sends a Boolean to Charlie;
either Charlie sends ok to Bob; Charlie sends ok to Alice;
or Charlie sends ok to Alice; Charlie sends ok to Bob;

Given a distributed implementation that "satisfies" this global specification:
1. Every send of a given type is matched by a reception of the same type;
Interest of global descriptions

Alice sends a Boolean to Charlie;
   either Charlie sends ok to Bob; Charlie sends ok to Alice;
   or Charlie sends ok to Alice; Charlie sends ok to Bob;

Given a distributed implementation that "satisfies" this global specification:
  1. Every send of a given type is matched by a reception of the same type;
  2. It should be easier to check the absence of deadlocks and starvation
     on global specifications.
Interest of global descriptions

Alice sends a Boolean to Charlie;
either Charlie sends ok to Bob; Charlie sends ok to Alice;
or Charlie sends ok to Alice; Charlie sends ok to Bob;

Given a distributed implementation that "satisfies" this global specification:

1. Every send of a given type is matched by a reception of the same type;
2. It should be easier to check the absence of deadlocks and starvation on global specifications.

We must ensure that all and only the expected synchronizations happen.
Given a distributed implementation that "satisfies" this global specification:

1. Every send of a given type is matched by a reception of the same type;
2. It should be easier to check the absence of deadlocks and starvation on global specifications.

We must ensure that *all and only* the expected synchronizations happen.
Given a distributed implementation that "satisfies" this global specification:

1. Every send of a given type is matched by a reception of the same type;
2. It should be easier to check the absence of deadlocks and starvation on global specifications.

We must ensure that all and only the expected synchronizations happen.
Interest of global descriptions

Alice sends a boolean to Charlie;
either Charlie sends ok to Bob; Charlie sends ok to Alice;
or Charlie sends ok to Alice; Charlie sends ok to Bob;

Given a distributed implementation that "satisfies" this global specification:

1. Every send of a given type is matched by a reception of the same type;
2. It should be easier to check the absence of deadlocks and starvation on global specifications.

We must ensure that all and only the expected synchronizations happen.
Interest of global descriptions

Alice sends a Boolean to Charlie;
either Charlie sends ok to Bob; Charlie sends ok to Alice;
or Charlie sends ok to Alice; Charlie sends ok to Bob;

Given a distributed implementation that “satisfies” this global specification:

1. Every send of a given type is matched by a reception of the same type;
2. It should be easier to check the absence of deadlocks and starvation on global specifications.

We must ensure that all and only the expected synchronizations happen.

We need a theoretical framework for:
- Defining global specifications,
- Defining local specifications,
- Relating them,
- Proving their properties.
A long-standing quest

Several communities formalize and study the relation between a global description and a set of components implementing it.
A long-standing quest

Several communities formalize and study the relation between a global description and a set of components implementing it.

**Typical issues:**
- **Verification:** does a given set of components *implement* a global specification?
- **Implementability:** does a set of components that implement the specification *exist* and can it be automatically produced?
- **Analysis:** which properties of the specification can be *checked and transposed* to every implementation that satisfies it?
A long-standing quest

Several communities formalize and study the relation between a global description and a set of components implementing it.

**Typical issues:**
- **Verification:** does a given set of components *implement* a global specification?
- **Implementability:** does a set of components that implement the specification *exist* and can it be automatically produced?
- **Analysis:** which properties of the specification can be checked and transposed to every implementation that satisfies it?

**Typical approaches:**
- **Automata:** software engineering for telecommunications; MSG and SDL-core (*ie*, CFSM); decidability and complexity;
- **Protocols:** cryptographic protocols; MSC, rewriting systems, process algebras; confidentiality, availability;
- **Services:** web services interactions; behavioral types and process algebras; soundness and progress.
A long-standing quest

- **Automata**: MSG and CFSM; decidability and complexity.
- **Protocols**: MSC, rewriting, concurrency; confidentiality, availability;
- **Services**: types and process algebras; soundness and progress.

These approaches differ by:
- the tackled problems,
- the levels of abstraction,
- the paradigms,
- the techniques.
A long-standing quest

- **Automata**: MSG and CFSM; decidability and complexity.
- **Protocols**: MSC, rewriting, concurrency; confidentiality, availability;
- **Services**: types and process algebras; soundness and progress.

These approaches differ by:
- the tackled problems,
- the levels of abstraction,
- the paradigms,
- the techniques.

However their frontiers are blurred
A long-standing quest

- **Automata**: MSG and CFSM; decidability and complexity.
- **Protocols**: MSC, rewriting, concurrency; confidentiality, availability;
- **Services**: types and process algebras; soundness and progress.

These approaches differ by:
- the tackled problems,
- the levels of abstraction,
- the paradigms,
- the techniques.

However their frontiers are blurred

**In the rest of this talk:**

1. Present a study typical of the **Services** approach;
2. Use it to briefly survey the related **Services**-oriented research;
3. Hint at and compare it with the **Automata** and **Protocols** approaches;
4. Draw few conclusions.
A study in the “services” approach.
From informal descriptions to global types

Seller sends buyer a price and a description of the product; then buyer may repeatedly send seller an offer then wait for a new price; then buyer sends seller acceptance or quits the conversation.
From informal descriptions to global types

Seller sends buyer a price and a description of the product; then buyer may repeatedly send seller an offer then wait for a new price; then buyer sends seller acceptance or quits the conversation.

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer}) \land (\text{seller} \xrightarrow{\text{price}} \text{buyer})\); \\
(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; (\text{seller} \xrightarrow{\text{price}} \text{buyer})*; \\
(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor (\text{buyer} \xrightarrow{\text{quit}} \text{seller}))\)
From informal descriptions to global types

Seller sends buyer a price and a description of the product; then buyer may repeatedly send seller an offer then wait for a new price; then buyer sends seller acceptance or quits the conversation.

\[(\text{seller } \xrightarrow{\text{descr}} \text{ buyer} \land \text{seller } \xrightarrow{\text{price}} \text{ buyer}); \]
\[(\text{buyer } \xrightarrow{\text{offer}} \text{ seller}; \text{seller } \xrightarrow{\text{price}} \text{ buyer})*; \]
\[(\text{buyer } \xrightarrow{\text{accept}} \text{ seller} \lor \text{buyer } \xrightarrow{\text{quit}} \text{ seller}) \]

- **Atomic actions:** “seller sends buyer a price” gets seller \(\xrightarrow{\text{price}}\) buyer
From informal descriptions to global types

Seller sends buyer a price and a description of the product; then buyer may repeatedly send seller an offer then wait for a new price; then buyer sends seller acceptance or quits the conversation.

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer})\);  
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer})^*\);  
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

- **Atomic actions:** “seller sends buyer a price” gets seller $\xrightarrow{\text{price}}$ buyer
- **Connectives:** “and”, “then”, “or” become “$\land$”, “$;$”, “$\lor$”
From informal descriptions to global types

Seller sends buyer a price and a description of the product; then buyer may repeatedly send seller an offer then wait for a new price; then buyer sends seller acceptance or quits the conversation.

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer});
\]
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer})^*;
\]
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})
\]

- **Atomic actions:** “seller sends buyer a price” gets seller \(\xrightarrow{\text{price}}\) buyer
- **Connectives:** “and”, “then”, “or” become “\(\land\)”, “;”, “\(\lor\)"
- **Control loops:** “may repeatedly” becomes “\((\ldots)^*\)"
From informal descriptions to global types

Seller sends buyer a price and a description of the product; then buyer may repeatedly send seller an offer then wait for a new price; then buyer sends seller acceptance or quits the conversation.

\[(\text{seller } descr \rightarrow \text{buyer } \land \text{seller } price \rightarrow \text{buyer});
\]

\[(\text{buyer } offer \rightarrow \text{seller}; \text{seller } price \rightarrow \text{buyer})*;
\]

\[(\text{buyer } accept \rightarrow \text{seller } \lor \text{buyer } quit \rightarrow \text{seller})
\]

- **Atomic actions**: “seller sends buyer a price” gets seller \(\rightarrow \text{buyer} \)
- **Connectives**: “and”, “then”, “or” become “\(\land\)”, “;”, “\(\lor\)”
- **Control loops**: “may repeatedly” becomes “(…)∗”
Syntax of Global Types

Global Types

\[ G ::= \text{skip} \quad (\text{skip}) \]
\[ \quad \mid \text{p} \xrightarrow{a} \text{p} \quad (\text{interaction}) \]
\[ \quad \mid G ; G \quad (\text{sequence}) \]
\[ \quad \mid G \land G \quad (\text{both}) \]
\[ \quad \mid G \lor G \quad (\text{either}) \]
\[ \quad \mid G^* \quad (\text{star}) \]
Syntax of Global Types

**Global Types**

\[ G ::= \text{skip} \mid p \xrightarrow{a} p \mid G; G \mid G \land G \mid G \lor G \mid G^* \]

Two observations:

1. Actually instead of just one sender we may specify multiple senders (ranged over by \( \pi \)) (seller price \( \xrightarrow{} \) buyer1 \( \land \) bank mortgage \( \xrightarrow{} \) buyer2);
2. Kleene star yields termination under fairness for free.
Syntax of Global Types

Global Types

\[ G ::= \text{skip} \mid \text{pp} \xrightarrow{a} \text{p} \mid G ; G \mid G \land G \mid G \lor G \mid G^* \]

Two observations:

1. Actually instead of just one sender
Syntax of Global Types

**Global Types**

\[ G ::= \text{skip} \quad \text{(skip)} \]
\[ \{p_1, \ldots, p_n\} \xrightarrow{a} p \quad \text{(interaction)} \]
\[ G ; G \quad \text{(sequence)} \]
\[ G \land G \quad \text{(both)} \]
\[ G \lor G \quad \text{(either)} \]
\[ G^* \quad \text{(star)} \]

**Two observations:**

1. Actually instead of just one sender we may specify multiple senders...
Syntax of Global Types

Global Types

\[
G ::= \text{skip} \quad (\text{skip}) \\
| \pi \rightarrow p \quad (\text{interaction}) \\
| G; G \quad (\text{sequence}) \\
| G \land G \quad (\text{both}) \\
| G \lor G \quad (\text{either}) \\
| G^* \quad (\text{star})
\]

Two observations:

1. Actually instead of just one sender we may specify multiple senders (ranged over by \(\pi\))
Syntax of Global Types

**Global Types**

\[ G ::= \text{skip} \] (skip)
\[ \quad \pi \xrightarrow{a} p \] (interaction)
\[ \quad G ; G \] (sequence)
\[ \quad G \land G \] (both)
\[ \quad G \lor G \] (either)
\[ \quad G^* \] (star)

**Two observations:**

1. Actually instead of just one sender we may specify multiple senders (ranged over by \( \pi \))

   \[(\text{seller} \xrightarrow{\text{price}} \text{buyer1} \land \text{bank} \xrightarrow{\text{mortgage}} \text{buyer2}); (\{\text{buyer1, buyer2}\} \xrightarrow{\text{accept}} \text{seller} \land \{\text{buyer1, buyer2}\} \xrightarrow{\text{accept}} \text{bank})\]
Syntax of Global Types

Global Types

\[ G ::= \begin{align*}
& \text{skip} \\
& \pi \rightarrow p \\
& G; G \\
& G \land G \\
& G \lor G \\
& G^* 
\end{align*} \]

Two observations:

1. Actually instead of just one sender we may specify multiple senders (ranged over by \( \pi \))

\[
(\text{seller} \xrightarrow{\text{price}} \text{buyer}_1 \land \text{bank} \xrightarrow{\text{mortgage}} \text{buyer}_2);
(\{\text{buyer}_1, \text{buyer}_2\} \xrightarrow{\text{accept}} \text{seller} \land \{\text{buyer}_1, \text{buyer}_2\} \xrightarrow{\text{accept}} \text{bank})
\]

2. Kleene star yields \textit{termination under fairness} for free.
Back to our example:

\[(\text{seller } \xrightarrow{\text{descr}} \text{buyer} \land \text{seller } \xrightarrow{\text{price}} \text{buyer});\]
\[(\text{buyer } \xrightarrow{\text{offer}} \text{seller}; \text{seller } \xrightarrow{\text{price}} \text{buyer})^*;\]
\[(\text{buyer } \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer } \xrightarrow{\text{quit}} \text{seller})\]
From Global to Local

Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer});\]
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer})^*;\]
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

A possible implementation:

\[\text{seller}\]

\[
\text{buyer}!\text{descr}.
\text{buyer}!\text{price}.
\text{rec } X . (\]
\[
\text{buyer}?\text{offer}.\text{buyer}!\text{price}.X
+\text{buyer}?\text{accept}.\text{end}
+\text{buyer}?\text{quit}.\text{end} )
\]

\[\text{buyer}\]

\[
\text{seller}?\text{descr}.
\text{seller}?\text{price}.
\text{rec } X . (\]
\[
\text{seller}!\text{offer}.\text{seller}?\text{price}.X
\oplus\text{seller}!\text{accept}.\text{end}
\oplus\text{seller}!\text{quit}.\text{end} )
\]
From Global to Local

Back to our example:

$seller\overset{\text{ descr}}{\rightarrow} buyer \land s\ell\varepsilon r\overset{\text{ price}}{\rightarrow} buyer);$

$(buyer\overset{\text{ offer}}{\rightarrow} seller;seller\overset{\text{ price}}{\rightarrow} buyer)*;$

$(buyer\overset{\text{ accept}}{\rightarrow} seller \lor buyer\overset{\text{ quit}}{\rightarrow} seller)$

A possible implementation:

**seller**

buyer!\textit{ descr}.  
buyer!\textit{ price}.  
rec \textit{ X} . (  
  buyer?\textit{ offer}.buyer!\textit{ price}.\textit{ X}  
  +buyer?\textit{ accept}.end  
  +buyer?\textit{ quit}.end )

**buyer**

seller?\textit{ descr}.  
seller?\textit{ price}.  
rec \textit{ X} . (  
  seller!\textit{ offer}.seller?\textit{ price}.\textit{ X}  
  +seller!\textit{ accept}.end  
  +seller!\textit{ quit}.end )
From Global to Local

Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer});\]
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer});\]
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller});\]

A possible implementation:

\begin{align*}
\text{seller} & \quad \text{buyer} \\
\text{buyer}!\text{descr}. & \quad \text{seller}?\text{descr}.
\text{buyer}!\text{price}. & \quad \text{seller}?\text{price}.
\text{rec } X . ( & \quad \text{rec } X . ( \\
\quad \text{buyer}?\text{offer}.\text{buyer}!\text{price}.X & \quad \text{seller}!\text{offer}.\text{seller}?\text{price}.X \\
\quad +\text{buyer}?\text{accept}.\text{end} & \quad +\text{seller}!\text{accept}.\text{end} \\
\quad +\text{buyer}?\text{quit}.\text{end} & \quad +\text{seller}!\text{quit}.\text{end} )
\end{align*}

Every action corresponds to a pair of communications.
Every communication comes from an action.
Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer})\); 
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer})*; 
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

Every action corresponds to a pair of communications. 
Every communication comes from an action.

A possible implementation:

```
seller

buyer!descr.
buyer!price.
rec X . ( 
    buyer?offer.buyer!price.X
    +buyer?accept.end
    +buyer?quit.end )
```

```
buyer

seller?descr.
seller?price.
rec X . ( 
    seller!offer.seller?price.X
    ⊕seller!accept.end
    ⊕seller!quit.end )
```
From Global to Local

Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer})\];
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer})^*\];
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

Every action corresponds to a pair of communications.

Every communication comes from an action.

A possible implementation:

```
seller
---
| buyer! descr.
| buyer! price.
| rec X . ( 
|   buyer? offer.buyer! price.X
| + buyer? accept.end
| + buyer? quit.end )

buyer
---
| seller? descr.
| seller? price.
| rec X . ( 
|   seller! offer.seller? price.X
| \oplus seller! accept.end
| \oplus seller! quit.end )
```
From Global to Local

Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer})\];
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer})*\];
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

A possible implementation:

```
output to buyer

seller

buyer! descr.
buyer! price.
```

```
input from seller

buyer

seller? descr.
seller? price.
rec X . ( seller! offer. seller? price. X + seller? accept. end + seller? quit. end )
```

Every action corresponds to a pair of communications.
Every communication comes from an action.
From Global to Local

Back to our example:

\[(\text{seller} \xmapsto{\text{descr}} \text{buyer} \wedge \text{seller} \xmapsto{\text{price}} \text{buyer});\]
\[(\text{buyer} \xmapsto{\text{offer}} \text{seller}; \text{seller} \xmapsto{\text{price}} \text{buyer})*;\]
\[(\text{buyer} \xmapsto{\text{accept}} \text{seller} \lor \text{buyer} \xmapsto{\text{quit}} \text{seller});\]

A possible implementation:

![Diagram of implementation]

Global choices correspond to internal/external choice pairs.
From Global to Local

Back to our example:

\[
\begin{align*}
(seller \xrightarrow{descr} buyer \land seller \xrightarrow{price} buyer) ;
\quad (buyer \xrightarrow{offer} seller ; seller \xrightarrow{price} buyer) * ;
\quad (buyer \xrightarrow{accept} seller \lor buyer \xrightarrow{quit} seller)
\end{align*}
\]

A possible implementation:

```
G. Castagna (CNRS)  
On Global Types and Multi-Party Sessions  
DisCoTec 2011 - Reykjavík
```
From Global to Local

Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer});\]
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer});\]
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

Global choices correspond to internal/external choice pairs

A possible implementation:

```
seller
buyer!\text{descr}.
buyer!\text{price}.
rec X . ( 
  buyer?\text{offer}.buyer!\text{price}.X
  +buyer?\text{accept}.end
  +buyer?\text{quit}.end )

buyer
seller?\text{descr}.
seller?\text{price}.
rec X . ( 
  seller!\text{offer}.seller?\text{price}.X
  \oplus\text{seller!\text{accept}.end}
  \ominus\text{seller!\text{quit}.end} )
```
From Global to Local

Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer});\]
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer});\]
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

A possible implementation:

Global choices correspond to internal/external choice pairs

G. Castagna (CNRS)
From Global to Local

Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer})\];

\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer})^*\];

\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

Kleene stars correspond to recursion

A possible implementation:

\[
\begin{align*}
\text{seller} & : \text{buyer}!\text{descr}. \\
& \quad \text{buyer}!\text{price}. \\
& \quad \text{rec } X . ( \text{buyer}?\text{offer}.\text{buyer}!\text{price}.X \\
& \quad + \text{buyer}?\text{accept}.\text{end} \\
& \quad + \text{buyer}?\text{quit}.\text{end} ) \\
\end{align*}
\]

\[
\begin{align*}
\text{buyer} & : \text{seller}?\text{descr}. \\
& \quad \text{seller}?\text{price}. \\
& \quad \text{rec } X . ( \text{seller}!\text{offer}.\text{seller}?\text{price}.X \\
& \quad + \text{seller}?\text{accept}.\text{end} \\
& \quad + \text{seller}?\text{quit}.\text{end} ) \\
\end{align*}
\]
From Global to Local

Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer}) \land (\text{buyer} \xrightarrow{\text{offer}} \text{seller} ; \text{seller} \xrightarrow{\text{price}} \text{buyer})* ; (\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

The order of sequential compositions is respected

A possible implementation:

\[
\begin{align*}
\text{seller} & \quad \text{buyer} \\
\text{buyer!descr.} & \quad \text{seller?descr.} \\
\text{buyer!price.} & \quad \text{seller?price.} \\
\text{rec X. (} & \quad \text{rec X. (}
\quad \text{buyer?offer.buyer!price.X} & \quad \text{seller!offer.seller?price.X}
\quad + \text{buyer?accept.end} & \quad + \text{seller!accept.end}
\quad + \text{buyer?quit.end}) & \quad \oplus \text{seller!quit.end})
\end{align*}
\]
From Global to Local

Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer})\];
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer})^*\];
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

A possible implementation:

A possible implementation:

\[
\begin{array}{l}
\text{seller} \quad \text{buyer} \\
\text{buyer}!\text{descr}. \\
\text{buyer}!\text{price}. \\
\text{rec } X . ( \\
\quad \text{buyer}?\text{offer}.\text{buyer}!\text{price}.X \\
\quad + \text{buyer}?\text{accept}.\text{end} \\
\quad + \text{buyer}?\text{quit}.\text{end} ) \\
\end{array}
\]

\[
\begin{array}{l}
\text{buyer} \quad \text{seller} \\
\text{seller}?\text{descr}. \\
\text{seller}?\text{price}. \\
\text{rec } X . ( \\
\quad \text{seller}?\text{offer}.\text{seller}?\text{price}.X \\
\quad \oplus \text{seller}?\text{accept}.\text{end} \\
\quad \oplus \text{seller}?\text{quit}.\text{end} ) \\
\end{array}
\]
Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer});\]
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller}; \text{seller} \xrightarrow{\text{price}} \text{buyer})*;\]
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

A possible implementation:

\[
\text{seller}
\begin{cases}
\text{buyer! descr.} \\
\text{buyer! price.} \\
\text{rec } X . ( \\
\quad \text{buyer? offer. buyer! price. } X \\
\quad + \text{buyer? accept. end} \\
\quad + \text{buyer? quit. end} ) \\
\end{cases}
\]

\[
\text{buyer}
\begin{cases}
\text{seller? descr.} \\
\text{seller? price.} \\
\text{rec } X . ( \\
\quad \text{seller! offer. seller? price. } X \\
\quad \oplus \text{seller! accept. end} \\
\quad \oplus \text{seller! quit. end} ) \\
\end{cases}
\]

Actions composed by “\(\land\)” appear in some order.

But other orders are admitted.
Back to our example:

\[(\text{seller} \xrightarrow{\text{descr}} \text{buyer} \land \text{seller} \xrightarrow{\text{price}} \text{buyer})\);\]
\[(\text{buyer} \xrightarrow{\text{offer}} \text{seller};\text{seller} \xrightarrow{\text{price}} \text{buyer})^*\);
\[(\text{buyer} \xrightarrow{\text{accept}} \text{seller} \lor \text{buyer} \xrightarrow{\text{quit}} \text{seller})\]

动作由“\(\land\)”结合在一起，按照某些顺序出现。

但其他顺序也允许。

可能的实现：

```
seller

buyer!price.

buyer!descr.

rec X. (buyer?offer.buyer!price.X
+buyer?accept.end
+buyer?quit.end )
```

```
buyer

seller?price.

seller?descr.

rec X. (seller!offer.seller?price.X
⊕seller!accept.end
⊕seller!quit.end )
```
Local Types and Projection

Implementations are specified by:

\[
T ::= \begin{array}{ll}
\text{end} & \text{(termination)} \\
\ p! a. T & \text{(output)} \\
\ T \oplus T & \text{(internal choice)} \\
\ \text{rec } X. T & \text{(recursion)} \\
\end{array}
\]

| \ T \ | \ X \ | \ (variable) \\
| \ p! a. T \ | \ \pi? a. T \ | \ (input) \\
| \ T \oplus T \ | \ T + T \ | \ (external choice) \\

Given a global type we want to automatically produce a mapping from participants to local types that is sound and complete, that is:

1. There is a 1-1 correspondence between actions and communications;
2. Communications of actions in ";;;" respect the order (sequentiality);
3. Communications of actions in "∧∧∧" occur in any order (shuffling);
4. Communications of actions in "∨∨∨" are mutually exclusive (alternative).
Local Types and Projection

Implementations are specified by:

\[ T ::= \text{end} \quad \text{(termination)} \]
\[ p!a.T \quad \text{(output)} \]
\[ T \oplus T \quad \text{(internal choice)} \]
\[ \text{rec } X.T \quad \text{(recursion)} \]
\[ X \quad \text{(variable)} \]
\[ \pi?a.T \quad \text{(input)} \]
\[ T + T \quad \text{(external choice)} \]

Given a global type we want to automatically produce a mapping from participants to local types that is \textit{sound and complete}, that is:
Implementations are specified by:

\[ T ::= \text{end} \quad \text{(termination)} \]
\[ | \quad p! a. T \quad \text{(output)} \]
\[ | \quad T \oplus T \quad \text{(internal choice)} \]
\[ | \quad \text{rec } X. T \quad \text{(recursion)} \]
\[ | \quad X \quad \text{(variable)} \]
\[ | \quad \pi ? a. T \quad \text{(input)} \]
\[ | \quad T + T \quad \text{(external choice)} \]

Given a global type we want to automatically produce a mapping from participants to local types that is \textit{sound and complete}, that is:

1. There is a \textit{1-1} correspondence between actions and communications;
2. Communications of actions in “;” respect the order (\textit{sequentiality});
3. Communications of actions in “∧” occur in any order (\textit{shuffling});
4. Communications of actions in “∨” are mutually exclusive (\textit{alternative})
Define the traces of a global types in the obvious way:

\[
\begin{align*}
\text{tr}(&\text{skip}) = \{\varepsilon\} \\
\text{tr}(\pi \xrightarrow{a} p) = \{\pi \xrightarrow{a} p\} \\
\text{tr}(\mathcal{G}^*) = (\text{tr}(\mathcal{G}))^* \\
\text{tr}(\mathcal{G}_1; \mathcal{G}_2) = \text{tr}(\mathcal{G}_1)\text{tr}(\mathcal{G}_2) \\
\text{tr}(\mathcal{G}_1 \lor \mathcal{G}_2) = \text{tr}(\mathcal{G}_1) \cup \text{tr}(\mathcal{G}_2) \\
\text{tr}(\mathcal{G}_1 \land \mathcal{G}_2) = \text{tr}(\mathcal{G}_1) \sqcup \text{tr}(\mathcal{G}_2)
\end{align*}
\]
Define the traces of a global type in the obvious way:

\[ tr(\text{skip}) = \{ \varepsilon \} \]
\[ tr(\pi \xrightarrow{a} p) = \{ \pi \xrightarrow{a} p \} \]
\[ tr(G^*) = (tr(G))^* \]

\[ tr(G_1; G_2) = tr(G_1) tr(G_2) \]
\[ tr(G_1 \lor G_2) = tr(G_1) \cup tr(G_2) \]
\[ tr(G_1 \land G_2) = tr(G_1) \sqcup tr(G_2) \text{ (shuffle)} \]
Define the traces of a global types in the obvious way:

\[ \text{tr}(\text{skip}) = \{\varepsilon\} \]
\[ \text{tr}(\pi \xrightarrow{a} p) = \{\pi \xrightarrow{a} p\} \]
\[ \text{tr}(G^*) = (\text{tr}(G))^* \]
\[ \text{tr}(G_1; G_2) = \text{tr}(G_1) \text{tr}(G_2) \]
\[ \text{tr}(G_1 \lor G_2) = \text{tr}(G_1) \cup \text{tr}(G_2) \]
\[ \text{tr}(G_1 \land G_2) = \text{tr}(G_1) \sqcup \text{tr}(G_2) \]

Define the traces of sets of components as traces of an LTS:

\[ \exists \text{tr}(G_2) \text{ (shuffle)} \]
### Soundness and completeness

1. Define the traces of a global types in the obvious way:

   - $tr(\text{skip}) = \{\varepsilon\}$
   - $tr(\pi \xrightarrow{a} p) = \{\pi \xrightarrow{a} p\}$
   - $tr(G^*) = (tr(G))^*$
   - $tr(G_1; G_2) = tr(G_1)tr(G_2)$
   - $tr(G_1 \lor G_2) = tr(G_1) \cup tr(G_2)$
   - $tr(G_1 \land G_2) = tr(G_1) \sqcup tr(G_2)$

2. Define the traces of *sets* of components as traces of an LTS:

   $$\begin{array}{c}
   \begin{bmatrix}
   \cdots, p : \bigoplus_{i \in I} p_i ! a_i . T_i, \cdots \\
   \end{bmatrix}
   \end{array}
   \xrightarrow{(p \xrightarrow{a_k} p_k) :: B}
   \begin{bmatrix}
   \cdots, p : T_k, \cdots \\
   \end{bmatrix}$$

   $(k \in I)$
Define the traces of a global types in the obvious way:

\[tr(\text{skip}) = \{\varepsilon\}\]
\[tr(\pi \xrightarrow{a} p) = \{\pi \xrightarrow{a} p\}\]
\[tr(G^*) = (tr(G))^*\]
\[tr(G_1; G_2) = tr(G_1) tr(G_2)\]
\[tr(G_1 \lor G_2) = tr(G_1) \cup tr(G_2)\]
\[tr(G_1 \land G_2) = tr(G_1) \sqcup tr(G_2)\]

Define the traces of sets of components as traces of an LTS:

\[
\begin{align*}
\mathbb{B} & \{\ldots, p : \bigoplus_{i \in I} p_i! a_i. T_i, \ldots\} \\
\mathbb{B} \vdash (p_i \xrightarrow{a} p)_{i \in I} & \{\ldots, p : \sum_{j \in J} \pi_j? a_j. T_j, \ldots\} \\
\pi_k \xrightarrow{a} p & \mathbb{B} \{\ldots, p : T_k, \ldots\}
\end{align*}
\]

\[
\mathbb{B} \vdash (p \xrightarrow{a_k} p_k) :: \mathbb{B} \{\ldots, p : T_k, \ldots\} \quad (k \in I)
\]

\[
\pi_k = \{p_i\}_{i \in I} \mathbb{B} \{\ldots, p : T_k, \ldots\} \quad (\pi_k = \{p_i\}_{i \in I})
\]
Soundness and completeness

1. Define the traces of a global types in the obvious way:
   \[ tr(\text{skip}) = \{ \varepsilon \} \]
   \[ tr(\pi \xrightarrow{a} p) = \{ \pi \xrightarrow{a} p \} \]
   \[ tr(G^*) = (tr(G))^* \]
   \[ tr(G_1; G_2) = tr(G_1)tr(G_2) \]
   \[ tr(G_1 \lor G_2) = tr(G_1) \cup tr(G_2) \]
   \[ tr(G_1 \land G_2) = tr(G_1) \sqcup tr(G_2) \]

2. Define the traces of sets of components as traces of an LTS:
   \[
   \begin{align*}
   & \exists \{ ..., p : \bigoplus_{i \in I} p_i! a_i. T_i, ... \} \quad \xrightarrow{p \xrightarrow{a_k} p_k} \quad \{ ..., p : T_k, ... \} \\
   & \exists \{ ..., p : \sum_{j \in J} \pi_j? a_j. T_j, ... \} \quad \xrightarrow{\pi_k \xrightarrow{a} p} \quad \{ ..., p : T_k, ... \}
   \end{align*}
   \]

3. Soundness: \[ tr(\{ p_i: T_i \}_{i \in I}) \subseteq tr(G) \]
   every trace of \( \{ p_i: T_i \}_{i \in I} \) is a trace of \( G \)
Soundness and completeness  

1. Define the traces of a global types in the obvious way:

\[
tr(\text{skip}) = \{ \varepsilon \} \\
tr(\pi \xrightarrow{a} p) = \{ \pi \xrightarrow{a} p \} \\
tr(G^*) = (tr(G))^*
\]

\[
tr(G_1; G_2) = tr(G_1)tr(G_2) \\
tr(G_1 \lor G_2) = tr(G_1) \cup tr(G_2) \\
tr(G_1 \land G_2) = tr(G_1) \sqcap tr(G_2)
\]

2. Define the traces of sets of components as traces of an LTS:

\[
\begin{align*}
B & \{ \ldots, p : \bigoplus_{i \in I} p_i!a_i \cdot T_i, \ldots \} & \xrightarrow{(p \xrightarrow{a_k} p_k):B} & \{ \ldots, p : T_k, \ldots \} \\
B & \{ \ldots, p : \sum_{j \in J} \pi_j?a_j \cdot T_j, \ldots \} & \xrightarrow{\pi_k \xrightarrow{a} p} & \{ \ldots, p : T_k, \ldots \}
\end{align*}
\]

3. Soundness: \( tr(\{ p_i : T_i \}_{i \in I}) \subseteq tr(G) \)

\( \)every trace of \( \{ p_i : T_i \}_{i \in I} \) is a trace of \( G \)

4. Completeness: \( tr(G) \subseteq tr(\{ p_i : T_i \}_{i \in I})^\circ \):

\( \)every trace of \( G \) is the permutation of a trace of \( \{ p_i : T_i \}_{i \in I} \).

\[
L^\circ \overset{\text{def}}{=} \{ \alpha_1 \cdots \alpha_n \mid \exists \text{ a permutation } \sigma \text{ s.t. } \alpha_{\sigma(1)} \cdots \alpha_{\sigma(n)} \in L \}
\]
Flawed global types

Some global types cannot be implemented by a sound and complete set of components

1. **No sequentiality:** Actions cannot synch without covert channels:

\[(p \xrightarrow{a} q; r \xrightarrow{b} s)\]
Flawed global types

Some global types cannot be implemented by a sound and complete set of components

1. **No sequentiality:** Actions cannot synch without covert channels:

   \[(p \xrightarrow{a} q; r \xrightarrow{b} s)\]

2. **No decision maker:** Branching must be decided by someone

   \[p \xrightarrow{a} q \lor q \xrightarrow{b} p\]
Flawed global types

Some global types cannot be implemented by a sound and complete set of components

1. **No sequentiality**: Actions cannot synch without covert channels:

   \[(p \xrightarrow{a} q; r \xrightarrow{b} s)\]

2. **No decision maker**: Branching must be decided by someone

   \[p \xrightarrow{a} q \lor q \xrightarrow{b} p\]

3. **No knowledge**: Other participants are not aware of the choice made.

   \[(p \xrightarrow{a} q; q \xrightarrow{a} r; r \xrightarrow{a} p) \lor (p \xrightarrow{b} q; q \xrightarrow{a} r; r \xrightarrow{b} p)\]
Flawed global types

Some global types cannot be implemented by a sound and complete set of components

1. *No sequentiality*: Actions cannot synch without covert channels:

   \[(p \xrightarrow{a} q; r \xrightarrow{b} s)\]

2. *No decision maker*: Branching must be decided by someone

   \[p \xrightarrow{a} q \lor q \xrightarrow{b} p\]

3. *No knowledge*: Other participants are not aware of the choice made.

   \[(p \xrightarrow{a} q; q \xrightarrow{a} r; r \xrightarrow{a} p)\]

   \[\lor\]

   \[(p \xrightarrow{b} q; q \xrightarrow{a} r; r \xrightarrow{b} p)\]

See proceedings for a formal characterization of the various kinds of flaw
Examples

This still leaves a lot of flexibility (cf. state of the art):

- **same message different receivers in a choice**

  $$\begin{align*}
  &\text{seller} \xrightarrow{\text{price}} \text{buyer}_1; \text{buyer}_1 \xrightarrow{\text{price}} \text{buyer}_2 \\
  \lor &\text{seller} \xrightarrow{\text{price}} \text{buyer}_2; \text{buyer}_2 \xrightarrow{\text{price}} \text{buyer}_1
  \end{align*}$$
Examples

This still leaves a lot of flexibility (*cf.* state of the art):

- **same message different receivers in a choice**

  (seller \(\xrightarrow{\text{price}}\) buyer\(_1\); buyer\(_1\) \(\xrightarrow{\text{price}}\) buyer\(_2\)

\(\lor\) seller \(\xrightarrow{\text{price}}\) buyer\(_2\); buyer\(_2\) \(\xrightarrow{\text{price}}\) buyer\(_1\))

- **different receivers to break a loop**

  seller \(\xrightarrow{\text{agency}}\) broker;

  (broker \(\xrightarrow{\text{offer}}\) buyer; buyer \(\xrightarrow{\text{counteroffer}}\) broker)*;

  (broker \(\xrightarrow{\text{result}}\) seller \(\land\) broker \(\xrightarrow{\text{result}}\) buyer)
Global types not inherently flawed are associated to sound and complete sets of components compositionally by a deduction system.
Global types not inherently flawed are associated to sound and complete sets of components compositionally by a deduction system.

(SP-Skip) \[ \Delta \vdash \text{skip} \triangleright \Delta \]

(SP-Action) \[ \{p : S, q : T, \ldots\} \vdash p \xrightarrow{a} q \triangleright \{p : q! a.S, q : p? a.T, \ldots\} \]

(SP-Sequence) \[ \Delta \vdash G_2 \triangleright \Delta' \quad \Delta' \vdash G_1 \triangleright \Delta'' \]
\[ \Delta \vdash G_1; G_2 \triangleright \Delta'' \]

(SP-Iteration) \[ \{p : T_1 \oplus T_2, \ldots\} \vdash G \triangleright \{p : T_1, \ldots\} \]
\[ \{p : T_2, \ldots\} \vdash G^* \triangleright \{p : T_1 \oplus T_2, \ldots\} \]

(SP-Alternative) \[ \Delta \vdash G_1 \triangleright \{p : T_1, \ldots\} \quad \Delta \vdash G_2 \triangleright \{p : T_2, \ldots\} \]
\[ \Delta \vdash G_1 \lor G_2 \triangleright \{p : T_1 \oplus T_2, \ldots\} \]

(SP-Subsumption) \[ \Delta \vdash G' \triangleright \Delta' \quad G' \subseteq G \quad \Delta'' \subseteq \Delta' \]
\[ \Delta \vdash G \triangleright \Delta'' \]

\( \Delta \vdash G \triangleright \Delta'' \)

\( X \subseteq Y \overset{\text{def}}{=} \text{tr}(L) \subseteq \text{tr}(M) \subseteq \text{tr}(L) \circ \)
Global types not inherently flawed are associated to **sound and complete** sets of components compositionally by a deduction system:

- **(SP-Skip)**
  \[ \Delta \vdash \text{skip} \triangleright \Delta \]

- **(SP-Action)**
  \[ \{ p : S, q : T, \ldots \} \vdash p \xrightarrow{a} q \triangleright \{ p : q! a.S, q : p?a.T, \ldots \} \]

- **(SP-Sequence)**
  \[ \Delta \vdash G_2 \triangleright \Delta' \quad \Delta' \vdash G_1 \triangleright \Delta'' \]
  \[ \Delta \vdash G_1; G_2 \triangleright \Delta'' \]

- **(SP-Iteration)**
  \[ \{ p : T_1 \oplus T_2, \ldots \} \vdash G \triangleright \{ p : T_1, \ldots \} \]
  \[ \{ p : T_2, \ldots \} \vdash G^* \triangleright \{ p : T_1 \oplus T_2, \ldots \} \]

- **(SP-Alternative)**
  \[ \Delta \vdash G_1 \triangleright \{ p : T_1, \ldots \} \quad \Delta \vdash G_2 \triangleright \{ p : T_2, \ldots \} \]
  \[ \Delta \vdash G_1 \lor G_2 \triangleright \{ p : T_1 \oplus T_2, \ldots \} \]

- **(SP-Subsumption)**
  \[ \Delta \vdash G' \triangleright \Delta' \quad G' \leq G \quad \Delta'' \leq \Delta' \]
  \[ \Delta \vdash G \triangleright \Delta'' \]

\[ (X \leq Y \overset{\text{def}}{=} \text{tr}(L) \subseteq \text{tr}(M) \subseteq \text{tr}(L)^{\circ}) \]
Global types not inherently flawed are associated to sound and complete sets of components compositionally by a deduction system

\begin{align*}
\text{(SP-Skip)} & \quad \Delta \vdash \text{skip} \quad \Delta \\
\text{(SP-Action)} & \quad \{p : S, q : T, \ldots\} \vdash p \xrightarrow{a} q \quad \{p : q! a. S, q : p? a. T, \ldots\}
\end{align*}

\begin{align*}
\text{(SP-Sequence)} & \quad \Delta \vdash G_2 \quad \Delta' \\
\text{(SP-Iteration)} & \quad \Delta' \vdash G_1 \quad \Delta'' \\
\text{(SP-Alternative)} & \quad \Delta \vdash G_1 \quad \Delta \vdash G_2 \\
\text{(SP-Subsumption)} & \quad \Delta \vdash G' \quad \Delta' \\
\end{align*}

\begin{align*}
\text{(X \leq Y) def } & \quad \text{tr}(L) \subseteq \text{tr}(M) \subseteq \text{tr}(L)^{\circ}
\end{align*}
Global types not inherently flawed are associated to sound and complete sets of components compositionally by a deduction system.

(SP-Skip)
\[ \Delta \vdash \text{skip} \triangleright \Delta \]

(SP-Action)
\[ \{ p : S, q : T, \ldots \} \vdash p \xrightarrow{a} q \triangleright \{ p : q! a . S, q : p? a . T, \ldots \} \]

(SP-Sequence)
\[ \Delta \vdash G_2 \triangleright \Delta' \quad \Delta' \vdash G_1 \triangleright \Delta'' \]
\[ \Delta \vdash G_1 ; G_2 \triangleright \Delta'' \]

(SP-Iteration)
\[ \{ p : T_1 \oplus T_2, \ldots \} \vdash G \triangleright \{ p : T_1, \ldots \} \]
\[ \{ p : T_2, \ldots \} \vdash G^* \triangleright \{ p : T_1 \oplus T_2, \ldots \} \]

(SP-Alternative)
\[ \Delta \vdash G_1 \triangleright \{ p : T_1, \ldots \} \quad \Delta \vdash G_2 \triangleright \{ p : T_2, \ldots \} \]
\[ \Delta \vdash G_1 \lor G_2 \triangleright \{ p : T_1 \oplus T_2, \ldots \} \]

(SP-Subsumption)
\[ \Delta \vdash G' \triangleright \Delta' \quad G' \leq G \quad \Delta'' \leq \Delta' \]
\[ \Delta \vdash G \triangleright \Delta'' \]

\[ (X \leq Y \overset{def}{=} \text{tr}(L) \subseteq \text{tr}(M) \subseteq \text{tr}(L)°) \]
Global types not inherently flawed are associated to **sound and complete** sets of components compositionally by a deduction system.

\textbf{(SP-Skip)} \quad \Delta \vdash \text{skip} \triangleright \Delta

\textbf{(SP-Action)} \quad \{p : S, q : T, \ldots \} \vdash p \xrightarrow{a} q \triangleright \{p : q!a.S, q : p?a.T, \ldots \}

\textbf{(SP-Sequence)} \quad \Delta \vdash G_2 \triangleright \Delta' \quad \Delta' \vdash G_1 \triangleright \Delta''
\quad \Delta \vdash G_1 ; G_2 \triangleright \Delta''

\textbf{(SP-Iteration)} \quad \{p : T_1 \oplus T_2, \ldots \} \vdash G \triangleright \{p : T_1, \ldots \}
\quad \{p : T_2, \ldots \} \vdash G^* \triangleright \{p : T_1 \oplus T_2, \ldots \}

\textbf{(SP-Alternative)} \quad \Delta \vdash G_1 \triangleright \{p : T_1, \ldots \} \quad \Delta \vdash G_2 \triangleright \{p : T_2, \ldots \}
\quad \Delta \vdash G_1 \lor G_2 \triangleright \{p : T_1 \oplus T_2, \ldots \}

\textbf{(SP-Subsumption)} \quad \Delta \vdash G' \triangleright \Delta' \quad G' \leq G \quad \Delta'' \leq \Delta'
\quad \Delta \vdash G \triangleright \Delta''

**Makes projection algorithm very hard (see proceedings)**
A three-layered structure

Global Type
\[ G = alice \xrightarrow{nat} bob; \]
\[ bob \xrightarrow{nat} charlie. \]

Local Types
\[ T_{bob} = alice? nat. \]
\[ charlie! nat. \]
end

Processes
\[ P_{bob} = \text{receive } x \text{ from alice;} \]
\[ \text{send } x+42 \text{ to charlie;} \]
end

Projection

Type checking

Processes

Global Type

Local Types

Projection

Type checking
A three-layered structure

Global Type
\[ G = \text{alice} \xrightarrow{\text{nat}} \text{bob}; \]
\[ \text{bob} \xrightarrow{\text{nat}} \text{charlie} \]

Local Types
\[ T_{\text{bob}} = \text{alice}!\text{nat}. \]
\[ \text{charlie}?!\text{nat}. \]
end

Processes
\[ P_{\text{bob}} = \text{receive } x \text{ from alice;} \]
\[ \text{send } x+42 \text{ to charlie;} \]
end

Projection

Type checking

Projection

Global Type

Local Types

Processes
A three-layered structure

Global Type

$G = \text{alice}^{nat} \rightarrow \text{bob}$;
$\text{bob}^{nat} \rightarrow \text{charlie}$

Local Types

$T_{\text{bob}} = \text{alice}^{nat}$.  
$\text{charlie}^{nat}$.  
end

Processes

$P_{\text{bob}} = \text{receive } x \text{ from alice; send } x+42 \text{ to charlie; end}$
A three-layered structure

Global Type
\[ G = \text{alice} \rightarrow^{nat} \text{bob}; \]
\[ \text{bob} \rightarrow^{nat} \text{charlie}; \]

Local Types
\[ T_{\text{bob}} = \text{alice}?!nat. \]
\[ \text{charlie}!?!nat. \]
end

Processes
\[ P_{\text{bob}} = \text{receive } x \text{ from alice}; \]
\[ \text{send } x+42 \text{ to charlie}; \]
end

Sought properties (second-third layers):

1. Subject reduction: No communication errors;
2. Progress: No stuck processes (safety);
3. Fairness: No starving processes (liveness).
A three-layered structure

Global Type
\[ G = alice^{nat} \rightarrow bob^{nat} \]
\[ bob^{nat} \rightarrow charlie^{nat} \]

Local Types
\[ T_{bob} = alice?^{nat}. charlie!^{nat}. \]
end

Processes
\[ P_{bob} = \text{receive } x \text{ from alice; send } x+42 \text{ to charlie; end} \]

Sought properties (second-third layers):

1. **Subject reduction**: No communication errors;
2. **Progress**: No stuck processes (safety);
3. **Fairness**: No starving processes (liveness).

**Checked on Local Types**
Other approaches
Automata approach: global specifications

*Seller sends buyer a price and a description of the product; then buyer may repeatedly send seller an offer then wait for a new price; then buyer sends seller acceptance or quits the conversation.*
Automata approach: global specifications

Seller sends buyer a price and a description of the product; then buyer may repeatedly send seller an offer then wait for a new price; then buyer sends seller acceptance or quits the conversation.

Message Sequence Graphs:
Automata approach: global specifications

\[
\begin{align*}
(seller \xrightarrow{descr} buyer \land seller \xrightarrow{price} buyer); \\
(buyer \xrightarrow{offer} seller; seller \xrightarrow{price} buyer)^*; \\
(buyer \xrightarrow{accept} seller \lor buyer \xrightarrow{quit} seller)
\end{align*}
\]

Message Sequence Graphs:

\[
\begin{align*}
\text{buyer} & \quad \text{seller} \\
\text{ descr} & \quad \text{ price} \\
\text{ price} & \quad \text{ descr} \\
\text{ seller} & \quad \text{ buyer}
\end{align*}
\]

\[
\begin{align*}
\text{buyer} & \quad \text{seller} \\
\text{ offer} & \quad \text{ price} \\
\text{ price} & \quad \text{ offer} \\
\text{ seller} & \quad \text{ buyer}
\end{align*}
\]

\[
\begin{align*}
\text{buyer} & \quad \text{seller} \\
\text{ accept} & \quad \text{ quit} \\
\text{ quit} & \quad \text{ accept} \\
\text{ seller} & \quad \text{ buyer}
\end{align*}
\]
Automata approach: local specifications

CFSM

Communicating Finite State Machines

G. Castagna (CNRS) On Global Types and Multi-Party Sessions DisCoTec 2011 - Reykjavík 20 / 31
Automata approach: local specifications

---

**CFSM**

**Communicating Finite State Machines**

---

**G. Castagna (CNRS)**

**On Global Types and Multi-Party Sessions**

**DisCoTec 2011 - Reykjavík 20 / 31**
Automata approach: local specifications

**Seller**

- **start** $\rightarrow q_0$
- $q_0 \rightarrow q_1$ \(\text{buyer!} \text{descr}\)
- $q_1 \rightarrow q_2$ \(\text{buyer?} \text{offer}\)
- $q_2 \rightarrow q_3, q_4$ \(\text{buyer?} \text{quit}, \text{buyer?} \text{accept}\)

**Buyer**

- **start** $\rightarrow q_0$
- $q_0 \rightarrow q_1$ \(\text{seller?} \text{descr}\)
- $q_1 \rightarrow q_2$ \(\text{seller!} \text{offer}\)
- $q_2 \rightarrow q_3, q_4$ \(\text{seller!} \text{quit}, \text{seller!} \text{accept}\)

**Buffers**

- ** descr**
- ** buyer!!! descr**
- ** seller?? descr**
- ** buyer?? price**
- ** seller??? price**

**CFSM**

- **Communicating Finite State Machines**
Automata approach: local specifications

**Seller**

- Start: $q_0$
- Buy: $q_1$
- Finish: $q_2$

**Buffers**
- Buyer: $q_3, q_4$

**Buyer**

- Start: $q_0$
- Seller: $q_1$
- Finish: $q_2$

**CFSM**

- Communicating Finite State Machines
Automata approach: local specifications

CFSM

Communicating Finite State Machines

G. Castagna (CNRS)
On Global Types and Multi-Party Sessions
DisCoTec 2011 - Reykjavík 20 / 31
Automata approach: local specifications

**Seller**

- Start: $q_0$
- $q_0 \xrightarrow{\text{buyer!\textit{descr}}} q_1$
- $q_1 \xrightarrow{\text{buyer?\textit{offer}}} q_2$
- $q_2 \xrightarrow{\text{buyer?\textit{accept}}} q_3\ x q_4$

**Buyer**

- Start: $q_0$
- $q_0 \xrightarrow{\text{seller?\textit{descr}}} q_1$
- $q_1 \xrightarrow{\text{seller!\textit{offer}}} q_2$
- $q_2 \xrightarrow{\text{seller!\textit{accept}}} q_3\ x q_4$

**Buffers**

- Price: $\text{price}$
- Description: $\text{descr}$

---

**CFSM**

Communicating Finite State Machines
Automata approach: local specifications

**Seller**

- **start** $\rightarrow q_0$
- $q_0 \rightarrow q_1$ : buyer!descr
- $q_1 \rightarrow q_2$ : buyer?offer
- $q_2 \rightarrow q_3$ : buyer?quit
- $q_2 \rightarrow q_4$ : buyer?accept

**Buyer**

- **start** $\rightarrow q_0$
- $q_0 \rightarrow q_1$ : seller?descr
- $q_1 \rightarrow q_2$ : seller!offer
- $q_2 \rightarrow q_3$ : seller!quit
- $q_2 \rightarrow q_4$ : seller!accept
Automata approach: local specifications

**Seller**

1. Start: $q_0$
2. Transition: $q_0 \xrightarrow{\text{buyer!descr}} q_1$
3. Transition: $q_1 \xrightarrow{\text{buyer?offer}} q_2$
4. Transition: $q_2 \xrightarrow{\text{buyer?accept}} q_3$
5. Transition: $q_2 \xrightarrow{\text{buyer?quit}} q_4$

**Buyer**

6. Start: $q_0$
7. Transition: $q_0 \xrightarrow{\text{seller?descr}} q_1$
8. Transition: $q_1 \xrightarrow{\text{price}} q_2$
9. Transition: $q_2 \xrightarrow{\text{seller!offer}} q_3$
10. Transition: $q_2 \xrightarrow{\text{seller!accept}} q_4$

**Buffers**

- Price communication
- Offer communication

**CFSM**

Communicating Finite State Machines

G. Castagna (CNRS)
Automata approach: local specifications

**Seller**

- Start: $q_0$
- $q_0$ transitions to $q_1$ on `buyer!descr`
- $q_1$ transitions to $q_2$ on `buyer?offer`
- $q_2$ transitions to $q_3$ on `buyer?quit`
- $q_2$ transitions to $q_4$ on `buyer?accept`

**Buyer**

- Start: $q_0$
- $q_0$ transitions to $q_1$ on `seller?descr`
- $q_1$ transitions to $q_2$ on `seller!offer`
- $q_2$ transitions to $q_3$ on `seller!quit`
- $q_2$ transitions to $q_4$ on `seller!accept`

**Buffers**

- `buyer!price` transitions from $q_1$ to $q_2$
- `seller?price` transitions from $q_2$ to $q_1$
- `buyer!offer` transitions from $q_2$ to $q_1$
- `seller?offer` transitions from $q_1$ to $q_2`

**CFSM**

Communicating Finite State Machines

G. Castagna (CNRS)

On Global Types and Multi-Party Sessions

DisCoTec 2011 - Reykjavík
Automata approach: local specifications

\begin{itemize}
\item Seller
\begin{itemize}
\item start \rightarrow q_0
\item buyer!descr \rightarrow q_1
\item buyer?offer \rightarrow q_2
\item buyer?quit \rightarrow q_3, q_4
\end{itemize}

\item buffers
\item buyer!price \rightarrow q_1
\item buyer?accept \rightarrow q_2

\item Buyer
\begin{itemize}
\item start \rightarrow q_0
\item seller?descr \rightarrow q_1
\item seller!offer \rightarrow q_2
\item seller?price \rightarrow q_3, q_4
\end{itemize}
\end{itemize}
Automata approach: local specifications

Seller

\begin{align*}
\text{start} & \xrightarrow{\text{buyer!descr}} q_0 \\
q_0 & \xrightarrow{\text{buyer?offer}} q_1 \\
q_1 & \xrightarrow{\text{buyer?price}} q_2 \\
q_2 & \xrightarrow{\text{buyer?accept}} q_3, q_4
\end{align*}

Buyer

\begin{align*}
\text{start} & \xrightarrow{\text{seller?descr}} q_0 \\
q_0 & \xrightarrow{\text{seller!offer}} q_1 \\
q_1 & \xrightarrow{\text{seller?price}} q_2 \\
q_2 & \xrightarrow{\text{seller!accept}} q_3, q_4 \\
q_3 & \xrightarrow{\text{seller!quit}} q_4
\end{align*}

CFSM

\begin{align*}
\text{seller?offer} & \xrightarrow{\text{buyer!price}} q_2 \\
\text{seller!offer} & \xrightarrow{\text{buyer?offer}} q_1 \\
\text{seller!quit} & \xrightarrow{\text{buyer?quit}} q_3, q_4
\end{align*}
Automata approach: local specifications

**Seller**

- **start** → $q_0$
- $q_0$ → $q_1$
- $q_1$ → $q_2$
- $q_2$ → $q_3$
- $q_3$ → $q_4$

**Buyer**

- **start** → $q_0$
- $q_0$ → $q_1$
- $q_1$ → $q_2$
- $q_2$ → $q_3$
- $q_3$ → $q_4$

---

**Buffers**

- buyer!$descr$
- buyer?$offer$
- buyer?$price$
- buyer?$accept$
- buyer?$quit$
- seller?$descr$
- seller!$offer$
- seller!$price$
- seller!$accept$
- seller!$quit$
Automata approach: local specifications

**Seller**

- Start: $q_0$
- Transition: $q_0 \xrightarrow{\text{buyer!descr}} q_1$
- Transition: $q_1 \xrightarrow{\text{buyer?offer}} q_2$
- Transition: $q_2 \xrightarrow{\text{buyer?quit}} q_3$
- Transition: $q_2 \xrightarrow{\text{buyer?accept}} q_4$

**Buyer**

- Start: $q_0$
- Transition: $q_0 \xrightarrow{\text{seller?descr}} q_1$
- Transition: $q_1 \xrightarrow{\text{seller!offer}} q_2$
- Transition: $q_2 \xrightarrow{\text{seller!accept}} q_3$
- Transition: $q_2 \xrightarrow{\text{seller!price}} q_4$

**Buffers**

- Transition: $q_0 \xrightarrow{\text{buyer!price}} q_1$
- Transition: $q_1 \xrightarrow{\text{seller?offer}} q_2$

**CFSM**

- Communicating Finite State Machines
Automata approach: problems and results

Research focused on *decidability, expressivity, and complexity*.

1. **CFSM are Turing complete.**
   - *Typical problems*: termination, reachability, deadlock freedom, boundedness (in general undecidable).
   - *Study of restrictions* to make them decidable (e.g., lossy channels, half-duplex, bounded buffers,...).

2. **MSG are finitely generated.**
   - *Typical problems*: model checking, implementability.
   - *Study of variants*: to have good closure properties, to make projection (into CFSM) effectively and efficiently implementable,

3. **Implementability** (generally meaning the same traces).
   - *Study of different notions of implementability* (e.g., unsound implementations, implementations with a controlled use of covert channels, implementation admitting deadlocks) to obtain decidability and/or polynomial complexity.
The protocol approach: global specifications

**MSC** (as for automata, but much more detailed):

\[
\begin{align*}
U &= \lt \text{username} \gt \\
A &= \text{random}() \\
p &= \lt \text{raw password} \gt \\
x &= \text{SHA}(s | \text{SHA}(U | '' | p)) \\
S &= (B - g x)(a + u * x) \\
K &= \text{SHA}\text{Interleave}(S) \\
\end{align*}
\]

Says how messages are generated

Says how messages are used
The protocol approach: global specifications

MSC (as for automata, but much more detailed):

\[
U = \langle \text{username} \rangle
\]
\[
a = \text{random}() \quad A = g^a \pmod{N}
\]
\[
p = \langle \text{raw password} \rangle \quad x = \text{SHA}(s | \text{SHA}(U) :'' | p))
\]
\[
S = (B - g^x)^{(a + u^x)} \pmod{N} \quad K = \text{SHA\_Interleave}(S)
\]

- **host**
  - \(U\)
  - \(s\)
  - \(A\)
  - \(B\)

- **client**
  - \(s\) = \langle \text{salt from passwd file} \rangle
  - \(v\) = \langle \text{stored passwd verif} \rangle
  - \(b = \text{random}() \quad B = (v + g^b) \pmod{N}
  - \(S = (A \ast v^u)^b \pmod{N} \quad K = \text{SHA\_Interleave}(S)\)
The protocol approach: global specifications

MSC (as for automata, but much more detailed):

U = <username>

\[ A = g^a \mod N \]

\[ a = \text{random()} \]

\[ p = < \text{raw password} > \]

\[ x = \text{SHA}(s|\text{SHA}(U|``:``}|p)) \]

\[ S = (B - g^x)^{a+u*x} \mod N \]

\[ K = \text{SHA}\_Interleave(S) \]

Says how messages are generated

RFC 2945 (SRP Authentication and Key Exchange System)
The protocol approach: global specifications

**MSC** (as for automata, but much more detailed):

![Diagram of the protocol]

- **U** = `<username>`
- **A** = $g^a \% N$
- **B**
- **s** = `<salt from passwd file>`
- **v** = `<stored passwd verif>`
- **b** = `random()`
- **B** = $(v + g^b) \% N$
- **S** = $(A^* v^u)^b \% N$
- **K** = `SHA_Interleave(S)`

*Says how messages are generated*

*Says how messages are used*

---

RFC 2945 (SRP Authentication and Key Exchange System)

G. Castagna (CNRS)  
On Global Types and Multi-Party Sessions  
DisCoTec 2011 - Reykjavík
Differences with automata and service approaches

**Simpler and lower-level paradigms:**

- *Interaction patterns are simpler*  
  (protocols are finite: MSCs instead of MSGs)

- *Content of interactions is richer and more detailed*  
  (in automata a finite set of message is often used).

- *The details of internal execution are exposed* both in global and local specifications (a small overlook may yield dramatic flaws)
Differences with automata and service approaches

**Simpler and lower-level paradigms:**
- *Interaction patterns are simpler*
  (protocols are finite: MSCs instead of MSGs)
- *Content of interactions is richer and more detailed*
  (in automata a finite set of message is often used).
- *The details of internal execution are exposed* both in global and local specifications (a small overlook may yield dramatic flaws)

**A larger variety** of specification languages (induced by the points above):
- **Global:** Carlsen, Casper, CAPSL, CASRUL, ...
- **Local:** modal logic, CSP, CCS, rewriting systems, spi-calculus.
Differences with automata and service approaches

Simpler and lower-level paradigms:

- *Interaction patterns are simpler*  
  (protocols are finite: MSCs instead of MSGs)
- *Content of interactions is richer and more detailed*  
  (in automata a finite set of message is often used).
- *The details of internal execution are exposed* both in global and local specifications (a small overlook may yield dramatic flaws)

A larger variety of specification languages (induced by the points above):

- **Global**: Carlsen, Casper, CAPSL, CASRUL, ...
- **Local**: modal logic, CSP, CCS, rewriting systems, spi-calculus.

Dynamicity (accounted for both by projection and by analysis)

- Protocols are specified for *roles*, implemented by several participants.
- Systems may include intruders and non specified participants that may alter the topology of interactions
- Different executions of the protocol may not be independent  
  *(cf. store and replay attacks)*
Related work in the “services” approach.
Related work in the “services” approach

The “services” approach explores different variants of global specifications, ... as the “automata” approach does.

The focus is on *how to model some use-cases* rather than how to satisfy some properties.

**Two examples:**

1. How to model a dynamically changing topology: *channels*.
2. How to model a dynamically changing set of participants: *roles*.

See the long version of the article for an extensive review of related work.
Specify channels and pass them around
Specify channels and pass them around

Two channels: $b$ shared by Alice and Bob, and $c$ by Alice and Charlie.

Alice $\xrightarrow{c\langle\text{Int}\rangle}$ Charlie; send an integer on channel $c$

Alice $\xrightarrow{b\langle c::!\text{Int}\rangle}$ Bob; delegate the sending of an int on $c$

Bob $\xrightarrow{c\langle\text{Int}\rangle}$ Charlie; send an integer on channel $c$

Alice $\xrightarrow{c\langle\text{Int}\rangle}$ Charlie; send an integer on channel $c$

send 1 on $c$ in
send 2 on $k$ in
send 3 on $c$ in

receive $x$ on $c$ in
receive $y$ on $c$ in
receive $z$ on $c$ in

$x+y+z$
Channels names are specified

Two channels: $b$ shared by Alice and Bob, and $c$ by Alice and Charlie.

- Alice $\xrightarrow{c\langle\text{Int}\rangle} \text{Charlie}$; send an integer on channel $c$
- Alice $\xrightarrow{b\langle c::!\text{Int}\rangle} \text{Bob}$; delegate the sending of an int on $c$
- Bob $\xrightarrow{c\langle\text{Int}\rangle} \text{Charlie}$; send an integer on channel $c$
- Alice $\xrightarrow{c\langle\text{Int}\rangle} \text{Charlie}$; send an integer on channel $c$
- Bob $\xrightarrow{c\langle\text{Int}\rangle} \text{Charlie}$; send an integer on channel $c$
Higher order sessions

[Honda, Yoshida, Carbone 2008]

Specify channels and pass them around

Two channels: 

- $b$ shared by Alice and Bob, and 
  
- $c$ by Alice and Charlie.

Alice $\xrightarrow{c\langle\text{Int}\rangle}$ Charlie; send an integer on channel $c$

Alice $\xrightarrow{b\langle c:\text{!Int}\rangle}$ Bob; delegate the sending of an int on $c$

Bob $\xrightarrow{c\langle\text{Int}\rangle}$ Charlie; send an integer on channel $c$

Alice $\xrightarrow{c\langle\text{Int}\rangle}$ Charlie; send an integer on channel $c$

Alice \& $\%$

send 1 on $c$ in

send 3 on $c$ in ()

Bob \& $\%$

receive $k$ on $b$ in

send 2 on $k$ in ()

Charlie \& $\%$

receive $x$ on $c$ in

receive $y$ on $c$ in

receive $z$ on $c$ in

$x + y + z$
Higher-order sessions

[Honda, Yoshida, Carbone 2008]

Specify channels and pass them around

Two channels: $b$ shared by Alice and Bob, and $c$ by Alice and Charlie.

Alice $\xrightarrow{\langle \text{Int} \rangle} c$ Charlie; send an integer on channel $c$

Alice $\xrightarrow{\langle \text{c!!Int} \rangle} b$ Charlie; delegate the sending of an int on $c$

Bob $\xrightarrow{\langle \text{Int} \rangle} c$ Charlie; send an integer on channel $c$

Alice $\xrightarrow{\langle \text{Int} \rangle} c$ Charlie; send an integer on channel $c$

Alice

send 1 on $c$ in
send $c$ on $b$ in
send 3 on $c$ in

() Bob

receive $k$ on $b$ in
send 2 on $k$ in
receive $x$ on $c$ in
receive $y$ on $c$ in
receive $z$ on $c$ in

$x+y+z$

Charlie

receive $x$ on $c$ in
receive $y$ on $c$ in
receive $z$ on $c$ in

$x+y+z$
Higher-order sessions

Specify channels and pass them around

Two channels: \( b \) shared by Alice and Bob, and \( c \) by Alice and Charlie.

- Alice \( \xrightarrow{c\langle\text{Int}\rangle} \) Charlie; send an integer on channel \( c \)
- Alice \( \xrightarrow{b\langle c::!\text{Int}\rangle} \) Bob; delegate the sending of an int on \( c \)
- Bob \( \xrightarrow{c\langle\text{Int}\rangle} \) Charlie; send an integer on channel \( c \)
- Alice \( \xrightarrow{c\langle\text{Int}\rangle} \) Charlie; send an integer on channel \( c \)

Charlie is not aware that this communication is with Bob

- Alice sends 1 on \( c \) in ()
- Bob \( \xrightarrow{\text{receive } k \text{ on } b \text{ in (}}} \)
- Charlie \( \xrightarrow{\text{receive } x \text{ on } c \text{ in (}}} \)
- Bob sends 2 on \( k \) in ()
- Bob receive \( k \) on \( c \) in ()
- Charlie receive \( y \) on \( c \) in ()
- Charlie receive \( z \) on \( c \) in ()
- Charlie \( \xrightarrow{\text{receive } x+y+z} \)
A seller that deal with just one or two buyers is unrealistic:

\[
\forall x : \text{buyer}. \quad (\text{seller} \xrightarrow{\text{descr}} x \land \text{seller} \xrightarrow{\text{price}} x);
\]

\[
(x \xrightarrow{\text{accept}} \text{seller} \lor x \xrightarrow{\text{quit}} \text{seller})
\]

buyer is a \textit{role}: can be played by different participants (ranged over by \(x\))
Dynamic multirole sessions

A seller that deal with just one or two buyers is unrealistic:

\[ \forall x : \text{buyer}. (\text{seller} \xrightarrow{\text{descr}} x \land \text{seller} \xrightarrow{\text{price}} x); \]
\[ (x \xrightarrow{\text{accept}} \text{seller} \lor x \xrightarrow{\text{quit}} \text{seller}) \]

buyer is a role: can be played by different participants (ranged over by \( x \))

\[ \forall x : \text{buyer}. x! \text{descr}. \]
\[ x! \text{price}. \]
\[ (x? \text{accept} + x? \text{quit}) \]

\[ \text{seller}? \text{descr}. \]
\[ \text{seller}? \text{price}. \]
\[ (\text{seller}! \text{accept} \oplus \text{seller}! \text{quit}) \]
Dynamic multirole sessions

A seller that deal with just one or two buyers is unrealistic:

\[ \forall x : \text{buyer}. \quad (\text{seller} \xrightarrow{\text{descr}} x \land \text{seller} \xrightarrow{\text{price}} x); \]
\[ (x \xrightarrow{\text{accept}} \text{seller} \lor x \xrightarrow{\text{quit}} \text{seller}) \]

buyer is a role: can be played by different participants (ranged over by \( x \))

\[ \forall x : \text{buyer}. \quad x! \text{descr.} \]
\[ x! \text{price.} \]
\[ (x? \text{accept} + x? \text{quit}) \]

\[ \text{seller? descr.} \]
\[ \text{seller? price.} \]
\[ (\text{seller! accept} \oplus \text{seller! quit}) \]

Main property: Communication safety and progress of projections are ensured also in the presence of dynamically joining and leaving participants
Dynamic multirole sessions [Deniélou & Yoshida 2011]

A seller that deal with just one or two buyers is unrealistic:

\[ \forall x : \text{buyer}. \ (\text{seller} x \rightarrow \text{seller} x) \land (\text{seller} x \rightarrow \text{seller} x) \]

\[ (x \rightarrow \text{seller} x \land x \rightarrow \text{seller} x) \]

buyer is a role: can be played by different participants (ranged over by \( x \))

```
 seller

 \forall x : \text{buyer}. x!\text{descr}.
 x!\text{price}.
 (x?\text{accept} + x?\text{quit})
```

```
 buyer

 seller?\text{descr}.
 seller?\text{price}.
 (seller!\text{accept} \oplus seller!\text{quit})
```

Main property: Communication safety and progress of projections are ensured also in the presence of dynamically joining and leaving participants

In this and the previous work roles and dynamicity are respectively internalized (in the “protocols” approach they usually are at the meta-level)
Conclusion
Conclusion

Automata and Services:

- The *automata approach* has a wealth of results in decidability and complexity that the *services* approach can use in studying its own framework and as guidelines for the definition of new ones.
- The automata community can find in the service framework new applications for their results and a new playground.

Protocol and Services:

- Protocols and Services approaches have a lot of common and they can mutually influence much more.
- Typing techniques are used to prove security properties while security protocols research spurs new research in type theory.
- Mutual influence is already happening:
  - WPPL [McCarthy & Krishnamurthi 2008] is a work in the verification of protocols directly inspired to multiparty section types
  - Dynamic multirole session types [Deniélou & Yoshida 2011] endow sessions with *roles* that protocols have been studying for many years.
A conclusion that Jacques II de Chabannes, seigneur de Lapalisse would have been proud of:

There are huge potential benefits for these communities to put their research efforts together.
Acknowledgments

The following persons helped us in preparing the survey in the full paper:

Martín Abadi, Roberto Amadio, Ahmed Bouajjani, Mario Bravetti, Roberto Bruni, Olivier Carton, Ivan Lanese, Anca Muscholl, Mihaela Sighireanu, Nobuko Yoshida, Gianluigi Zavattaro, Wiesław Zielonka.