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

1. Relating global descriptions distributed systems with sets of descriptions of their components is the subject of an important and long-standing research.
2. Recently, the community of behavioral types for web services has joined this effort.
3. The aim of this talk is to give an overview of the research done by these newcomers, addressing its goals and specificities.

1. Relating global descriptions distributed systems with sets of descriptions of their components is the subject of an important and long-standing research.

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

3. 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

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;
```

They do it by exchanging some messages

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

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

```
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;
```

Several potential problems

- **Communication errors**

A string is sent but a Boolean is expected

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

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;
```

Several potential problems

- **Communication errors**

Context

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

```
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 }
```

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;
```

Several potential problems

- Communication errors
- **Protocol errors**

A message is sent but there is no corresponding reception

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;
```

Several potential problems

- Communication errors
- **Protocol errors**

A message is sent but there is no corresponding reception

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**

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
```

There may be deadlocks

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

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

Several potential problems

- Communication errors
- **Protocol errors**

There may be starvation

Context

```
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
```

Several potential problems

- Communication errors
- **Protocol errors**

There may be starvation

Here Bob starves

Context

```
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

Context

```
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

Context

```
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
```

Several potential problems

- Communication errors
- Protocol errors

These problems may be due to:

- **Programming errors**
- **Software evolution**
- **Rogue participants**

Global vs. Local specifications

Global specification

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.

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

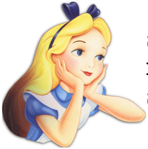
The global specification is compact and synthetic

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

An abstract description of this protocol



```
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

An abstract description of this protocol

```
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          lice -> receive ok from Charlie
 | receive          harlie -> receive ok from Alice
```

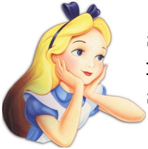
It abstracts values

Example of global descrip on

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

An abstract description of this protocol



```
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        Alice -> receive ok from Charlie
 | receive        harlie -> receive ok from Alice
```

It abstracts values

It abstracts choices

Examp   l descrip   on

Alice   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;

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;
```

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;

Int lobal descriptions

Alice sends a Boolean

```
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;

Int… lobal descriptions

Alice sends a Boolean

Charlie receives a Boolean

```
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.

Interest of g… scriptions

This send by Charlie must synch

```
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

This send by Charlie must synch

... with this reception by Bob

```
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

This send by Charlie must synch

... with this reception by Bob

... and not with this one!

```
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.

$$\begin{array}{l}(\texttt{seller} \xrightarrow{\textit{descr}} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer})\,;\\ (\texttt{buyer} \xrightarrow{\textit{offer}} \texttt{seller}\,;\texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer})*\,;\\ (\texttt{buyer} \xrightarrow{\textit{accept}} \texttt{seller} \vee \texttt{buyer} \xrightarrow{\textit{quit}} \texttt{seller})\end{array}$$

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.

$$
\begin{array}{l}
(\mathtt{seller} \xrightarrow{descr} \mathtt{buyer} \wedge \mathtt{seller} \xrightarrow{price} \mathtt{buyer});\\
(\mathtt{buyer} \xrightarrow{offer} \mathtt{seller}; \mathtt{seller} \xrightarrow{price} \mathtt{buyer})*;\\
(\mathtt{buyer} \xrightarrow{accept} \mathtt{seller} \vee \mathtt{buyer} \xrightarrow{quit} \mathtt{seller})
\end{array}
$$

- *Atomic actions:* "*seller sends buyer a price*" gets $\mathtt{seller} \xrightarrow{price} \mathtt{buyer}$

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.

$$(\texttt{seller} \xrightarrow{\textit{descr}} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{\textit{offer}} \texttt{seller}; \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{\textit{accept}} \texttt{seller} \vee \texttt{buyer} \xrightarrow{\textit{quit}} \texttt{seller})$$

- *Atomic actions:* "*seller sends buyer a price*" gets $\texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer}$
- *Connectives:* "and", "then", "or" become "$\wedge$", "$;$", "$\vee$"

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.

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}; \texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

- *Atomic actions:* "*seller sends buyer a price*" gets $\texttt{seller} \xrightarrow{price} \texttt{buyer}$
- *Connectives:* "and", "then", "or" become "$\wedge$", "$;$", "$\vee$"
- *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.

$$(\texttt{seller} \xrightarrow{\textit{descr}} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer})\,\texttt{;}$$
$$(\texttt{buyer} \xrightarrow{\textit{offer}} \texttt{seller}\,\texttt{;}\,\texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer})\ast\,\texttt{;}$$
$$(\texttt{buyer} \xrightarrow{\textit{accept}} \texttt{seller} \vee \texttt{buyer} \xrightarrow{\textit{quit}} \texttt{seller})$$

- *Atomic actions:* "*seller sends buyer a price*" gets $\texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer}$
- *Connectives:* "and", "then", "or" become "$\wedge$", "$\texttt{;}$", "$\vee$"
- *Control loops:* "*may repeatedly*" becomes "$\texttt{(...)}\ast$"

Syntax of Global Types

Global Types

$$
\begin{array}{llll}
\mathscr{G} & ::= & \mathsf{skip} & \text{(skip)} \\
& \mid & \mathtt{p} \xrightarrow{a} \mathtt{p} & \text{(interaction)} \\
& \mid & \mathscr{G} ; \mathscr{G} & \text{(sequence)} \\
& \mid & \mathscr{G} \wedge \mathscr{G} & \text{(both)} \\
& \mid & \mathscr{G} \vee \mathscr{G} & \text{(either)} \\
& \mid & \mathscr{G}^{*} & \text{(star)}
\end{array}
$$

Syntax of Global Types

Global Types

$$
\begin{array}{llll}
\mathscr{G} & ::= & \mathsf{skip} & \text{(skip)} \\
& \mid & \mathtt{p} \xrightarrow{a} \mathtt{p} & \text{(interaction)} \\
& \mid & \mathscr{G} ; \mathscr{G} & \text{(sequence)} \\
& \mid & \mathscr{G} \wedge \mathscr{G} & \text{(both)} \\
& \mid & \mathscr{G} \vee \mathscr{G} & \text{(either)} \\
& \mid & \mathscr{G}^{*} & \text{(star)}
\end{array}
$$

Two observations:

Syntax of Global Types

Global Types

$$
\begin{array}{llll}
\mathscr{G} & ::= & \mathsf{skip} & \text{(skip)} \\
& | & \mathtt{p} \xrightarrow{a} \mathtt{p} & \text{(interaction)} \\
& | & \mathscr{G} ; \mathscr{G} & \text{(sequence)} \\
& | & \mathscr{G} \wedge \mathscr{G} & \text{(both)} \\
& | & \mathscr{G} \vee \mathscr{G} & \text{(either)} \\
& | & \mathscr{G}^{*} & \text{(star)}
\end{array}
$$

Two observations:

1. Actually instead of just one sender

Syntax of Global Types

Global Types

$$
\begin{array}{lcll}
\mathscr{G} & ::= & \mathsf{skip} & \text{(skip)} \\
 & \mid & \{\mathtt{p}_1, \ldots, \mathtt{p}_n\} \xrightarrow{a} \mathtt{p} & \text{(interaction)} \\
 & \mid & \mathscr{G} ; \mathscr{G} & \text{(sequence)} \\
 & \mid & \mathscr{G} \wedge \mathscr{G} & \text{(both)} \\
 & \mid & \mathscr{G} \vee \mathscr{G} & \text{(either)} \\
 & \mid & \mathscr{G}^* & \text{(star)}
\end{array}
$$

Two observations:

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

Syntax of Global Types

Global Types

$$
\begin{array}{llll}
\mathscr{G} & ::= & \mathsf{skip} & \text{(skip)} \\
 & | & \boldsymbol{\pi} \xrightarrow{a} \mathtt{p} & \text{(interaction)} \\
 & | & \mathscr{G};\mathscr{G} & \text{(sequence)} \\
 & | & \mathscr{G} \wedge \mathscr{G} & \text{(both)} \\
 & | & \mathscr{G} \vee \mathscr{G} & \text{(either)} \\
 & | & \mathscr{G}^{*} & \text{(star)}
\end{array}
$$

Two observations:

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

Syntax of Global Types

Global Types

$$
\begin{array}{lcll}
\mathscr{G} & ::= & \mathsf{skip} & \text{(skip)} \\
 & | & \boldsymbol{\pi} \xrightarrow{a} \mathtt{p} & \text{(interaction)} \\
 & | & \mathscr{G} ; \mathscr{G} & \text{(sequence)} \\
 & | & \mathscr{G} \wedge \mathscr{G} & \text{(both)} \\
 & | & \mathscr{G} \vee \mathscr{G} & \text{(either)} \\
 & | & \mathscr{G}^{*} & \text{(star)}
\end{array}
$$

Two observations:

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

$$(\mathtt{seller} \xrightarrow{\mathit{price}} \mathtt{buyer1} \wedge \mathtt{bank} \xrightarrow{\mathit{mortgage}} \mathtt{buyer2});$$
$$(\{\mathtt{buyer1},\mathtt{buyer2}\} \xrightarrow{\mathit{accept}} \mathtt{seller} \wedge \{\mathtt{buyer1},\mathtt{buyer2}\} \xrightarrow{\mathit{accept}} \mathtt{bank})$$

Syntax of Global Types

Global Types	$\mathscr{G}$	$::=$	skip	(skip)
		$\mid$	$\boldsymbol{\pi} \xrightarrow{a} \mathtt{p}$	(interaction)
		$\mid$	$\mathscr{G};\mathscr{G}$	(sequence)
		$\mid$	$\mathscr{G} \wedge \mathscr{G}$	(both)
		$\mid$	$\mathscr{G} \vee \mathscr{G}$	(either)
		$\mid$	$\mathscr{G}^{*}$	(star)

Two observations:

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

$$(\mathtt{seller} \xrightarrow{\textit{price}} \mathtt{buyer1} \wedge \mathtt{bank} \xrightarrow{\textit{mortgage}} \mathtt{buyer2});$$
$$(\{\mathtt{buyer1},\mathtt{buyer2}\} \xrightarrow{\textit{accept}} \mathtt{seller} \wedge \{\mathtt{buyer1},\mathtt{buyer2}\} \xrightarrow{\textit{accept}} \mathtt{bank})$$

2. Kleene star yields *termination under fairness* for free.

From Global to Local

Back to our example:

$$(\texttt{seller} \xrightarrow{\textit{descr}} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{\textit{offer}} \texttt{seller}; \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{\textit{accept}} \texttt{seller} \vee \texttt{buyer} \xrightarrow{\textit{quit}} \texttt{seller})$$

Back to our example:

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}; \texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

A possible implementation:

seller

$\texttt{buyer}!\,descr.$
$\texttt{buyer}!\,price.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{buyer}?\,offer.\texttt{buyer}!\,price.X$
$\ +\texttt{buyer}?\,accept.\mathsf{end}$
$\ +\texttt{buyer}?\,quit.\mathsf{end}\ )$

buyer

$\texttt{seller}?\,descr.$
$\texttt{seller}?\,price.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{seller}!\,offer.\texttt{seller}?\,price.X$
$\ \oplus\texttt{seller}!\,accept.\mathsf{end}$
$\ \oplus\texttt{seller}!\,quit.\mathsf{end}\ )$

From Global to Local

Back to our example:

Why is this an implementation?

$$(\mathtt{seller} \xrightarrow{descr} \mathtt{buyer} \wedge \mathtt{seller} \xrightarrow{price} \mathtt{buyer});$$
$$(\mathtt{buyer} \xrightarrow{offer} \mathtt{seller};\mathtt{seller} \xrightarrow{price} \mathtt{buyer})*;$$
$$(\mathtt{buyer} \xrightarrow{accept} \mathtt{seller} \vee \mathtt{buyer} \xrightarrow{quit} \mathtt{seller})$$

A possible implementation:

seller

$\mathtt{buyer}\,\mathbf{!}\,descr.$
$\mathtt{buyer}\,\mathbf{!}\,price.$
$\mathsf{rec}\ X\ .\ ($
$\quad \mathtt{buyer}\,\mathbf{?}\,offer.\mathtt{buyer}\,\mathbf{!}\,price.X$
$\ +\mathtt{buyer}\,\mathbf{?}\,accept.\mathsf{end}$
$\ +\mathtt{buyer}\,\mathbf{?}\,quit.\mathsf{end}\ )$

buyer

$\mathtt{seller}\,\mathbf{?}\,descr.$
$\mathtt{seller}\,\mathbf{?}\,price.$
$\mathsf{rec}\ X\ .\ ($
$\quad \mathtt{seller}\,\mathbf{!}\,offer.\mathtt{seller}\,\mathbf{?}\,price.X$
$\ \oplus\mathtt{seller}\,\mathbf{!}\,accept.\mathsf{end}$
$\ \oplus\mathtt{seller}\,\mathbf{!}\,quit.\mathsf{end}\ )$

From Global to Local

Back to our example:

$$(\texttt{seller} \xrightarrow{\mathit{descr}} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{\mathit{price}} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{\mathit{offer}} \texttt{seller}; \texttt{seller} \xrightarrow{\mathit{price}} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{\mathit{accept}} \texttt{seller} \vee \texttt{buyer} \xrightarrow{\mathit{quit}} \texttt{seller})$$

Every action corresponds to a pair of communications.

Every communication comes from an action.

A possible implementation:

seller

$\texttt{buyer}!\mathit{descr}.$
$\texttt{buyer}!\mathit{price}.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{buyer}?\mathit{offer}.\texttt{buyer}!\mathit{price}.X$
$\ +\texttt{buyer}?\mathit{accept}.\mathsf{end}$
$\ +\texttt{buyer}?\mathit{quit}.\mathsf{end}\ )$

buyer

$\texttt{seller}?\mathit{descr}.$
$\texttt{seller}?\mathit{price}.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{seller}!\mathit{offer}.\texttt{seller}?\mathit{price}.X$
$\ \oplus\texttt{seller}!\mathit{accept}.\mathsf{end}$
$\ \oplus\texttt{seller}!\mathit{quit}.\mathsf{end}\ )$

Back to our example:

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller};\texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

Every action corresponds to a pair of communications.

Every communication comes from an action.

A possible implementation:

seller

$\texttt{buyer!}\mathit{descr}.$
$\texttt{buyer!}\mathit{price}.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{buyer?}\mathit{offer}.\texttt{buyer!}\mathit{price}.X$
$\quad +\texttt{buyer?}\mathit{accept}.\mathrm{end}$
$\quad +\texttt{buyer?}\mathit{quit}.\mathrm{end}\ )$

buyer

$\texttt{seller?}\mathit{descr}.$
$\texttt{seller?}\mathit{price}.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{seller!}\mathit{offer}.\texttt{seller?}\mathit{price}.X$
$\quad \oplus\texttt{seller!}\mathit{accept}.\mathrm{end}$
$\quad \oplus\texttt{seller!}\mathit{quit}.\mathrm{end}\ )$

Back to our example:

$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$
$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}; \texttt{seller} \xrightarrow{price} \texttt{buyer})*;$
$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$

Every action corresponds to a pair of communications.

Every communication comes from an action.

A possible implementation:

seller

$\texttt{buyer!}\mathit{descr}.$
$\texttt{buyer!}\mathit{price}.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{buyer?}\mathit{offer}.\texttt{buyer!}\mathit{price}.X$
$\quad + \texttt{buyer?}\mathit{accept}.\mathrm{end}$
$\quad + \texttt{buyer?}\mathit{quit}.\mathrm{end}\ )$

buyer

$\texttt{seller?}\mathit{descr}.$
$\texttt{seller?}\mathit{price}.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{seller!}\mathit{offer}.\texttt{seller?}\mathit{price}.X$
$\quad \oplus \texttt{seller!}\mathit{accept}.\mathrm{end}$
$\quad \oplus \texttt{seller!}\mathit{quit}.\mathrm{end}\ )$

From Global to Local

Back to our example:

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \land \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}; \texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \lor \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

Every action corresponds to a pair of communications.

Every communication comes from an action.

A possible i on:

output to buyer

seller

$\texttt{buyer}!descr.$
$\texttt{buyer}!price.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{buyer}?offer.\texttt{buyer}!price.X$
$\quad +\texttt{buyer}?accept.\mathrm{end}$
$\quad +\texttt{buyer}?quit.\mathrm{end}\ )$

input from seller

buyer

$\texttt{seller}?descr.$
$\texttt{seller}?price.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{seller}!offer.\texttt{seller}?price.X$
$\quad \oplus\texttt{seller}!accept.\mathrm{end}$
$\quad \oplus\texttt{seller}!quit.\mathrm{end}\ )$

From Global to Local

Back to our example:

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}\,;\texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

Global choices correspond to internal/external choice pairs

A possible implementation:

seller

$\texttt{buyer!}\mathit{descr}.$
$\texttt{buyer!}\mathit{price}.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{buyer?}\mathit{offer}.\texttt{buyer!}\mathit{price}.X$
$\ +\texttt{buyer?}\mathit{accept}.\mathsf{end}$
$\ +\texttt{buyer?}\mathit{quit}.\mathsf{end}\ )$

buyer

$\texttt{seller?}\mathit{descr}.$
$\texttt{seller?}\mathit{price}.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{seller!}\mathit{offer}.\texttt{seller?}\mathit{price}.X$
$\ \oplus\texttt{seller!}\mathit{accept}.\mathsf{end}$
$\ \oplus\texttt{seller!}\mathit{quit}.\mathsf{end}\ )$

From Global to Local

Back to our example:

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}; \texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

Global choices correspond to internal/external choice pairs

A possible implementation:

seller

$\texttt{buyer}!descr.$
$\texttt{buyer}!price.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{buyer}?offer.\texttt{buyer}!price.X$
$\ +\texttt{buyer}?accept.\mathrm{end}$
$\ +\texttt{buyer}?quit.\mathrm{end}\ )$

buyer

$\texttt{seller}?descr.$
$\texttt{seller}?price.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{seller}!offer.\texttt{seller}?price.X$
$\ \oplus\texttt{seller}!accept.\mathrm{end}$
$\ \oplus\texttt{seller}!quit.\mathrm{end}\ )$

From Global to Local

Back to our example:

$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$

$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}; \texttt{seller} \xrightarrow{price} \texttt{buyer})*;$

$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$

Global choices correspond to internal/external choice pairs

A possible implementation:

seller

$\texttt{buyer}\,!\,descr.$
$\texttt{buyer}\,!\,price.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{buyer}\,?\,offer.\texttt{buyer}\,!\,price.X$
$+\texttt{buyer}\,?\,accept.\mathsf{end}$
$+\texttt{buyer}\,?\,quit.\mathsf{end}\ )$

buyer

$\texttt{seller}\,?\,descr.$
$\texttt{seller}\,?\,price.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{seller}\,!\,offer.\texttt{seller}\,?\,price.X$
$\oplus\texttt{seller}\,!\,accept.\mathsf{end}$
$\oplus\texttt{seller}\,!\,quit.\mathsf{end}\ )$

From Global to Local

Back to our example:

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}; \texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

Global choices correspond to internal/external choice pairs

A possible implementation:

seller

$\texttt{buyer!}\,descr.$
bu
rec ... $.\texttt{buyer!}\,price.X$
$+\texttt{buyer?}\,accept.\mathrm{end}$
$+\texttt{buyer?}\,quit.\mathrm{end}$)

external choice

buyer

$\texttt{seller?}\,descr.$
se
re ... $er.\texttt{seller?}\,price.X$
$\oplus\texttt{seller!}\,accept.\mathrm{end}$
$\oplus\texttt{seller!}\,quit.\mathrm{end}$)

internal choice

From Global to Local

Back to our example:

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \land \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}\,;\texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \lor \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

Kleene stars correspond to recursion

A possible implementation:

seller

$\texttt{buyer!}\mathit{descr}.$
$\texttt{buyer!}\mathit{price}.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{buyer?}\mathit{offer}.\texttt{buyer!}\mathit{price}.X$
$\quad +\texttt{buyer?}\mathit{accept}.\mathsf{end}$
$\quad +\texttt{buyer?}\mathit{quit}.\mathsf{end}\ )$

buyer

$\texttt{seller?}\mathit{descr}.$
$\texttt{seller?}\mathit{price}.$
$\mathsf{rec}\ X\ .\ ($
$\quad \texttt{seller!}\mathit{offer}.\texttt{seller?}\mathit{price}.X$
$\quad \oplus\texttt{seller!}\mathit{accept}.\mathsf{end}$
$\quad \oplus\texttt{seller!}\mathit{quit}.\mathsf{end}\ )$

From Global to Local

Back to our example:

$$(\mathtt{seller} \xrightarrow{descr} \mathtt{buyer} \wedge \mathtt{seller} \xrightarrow{price} \mathtt{buyer});$$
$$(\mathtt{buyer} \xrightarrow{offer} \mathtt{seller}; \mathtt{seller} \xrightarrow{price} \mathtt{buyer})*;$$
$$(\mathtt{buyer} \xrightarrow{accept} \mathtt{seller} \vee \mathtt{buyer} \xrightarrow{quit} \mathtt{seller})$$

The order of sequential compositions is respected

A possible implementation:

seller

$\mathtt{buyer}!\mathit{descr}.$
$\mathtt{buyer}!\mathit{price}.$
$\mathrm{rec}\ X\ .\ ($
$\quad \mathtt{buyer}?\mathit{offer}.\mathtt{buyer}!\mathit{price}.X$
$\quad +\mathtt{buyer}?\mathit{accept}.\mathrm{end}$
$\quad +\mathtt{buyer}?\mathit{quit}.\mathrm{end}\ )$

buyer

$\mathtt{seller}?\mathit{descr}.$
$\mathtt{seller}?\mathit{price}.$
$\mathrm{rec}\ X\ .\ ($
$\quad \mathtt{seller}!\mathit{offer}.\mathtt{seller}?\mathit{price}.X$
$\quad \oplus\mathtt{seller}!\mathit{accept}.\mathrm{end}$
$\quad \oplus\mathtt{seller}!\mathit{quit}.\mathrm{end}\ )$

From Global to Local

Back to our example:

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}; \texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

Actions composed by "$\wedge$" appear in some order.

A possible implementation:

seller

$\texttt{buyer}!descr.$
$\texttt{buyer}!price.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{buyer}?offer.\texttt{buyer}!price.X$
$\ +\texttt{buyer}?accept.\mathrm{end}$
$\ +\texttt{buyer}?quit.\mathrm{end}\ )$

buyer

$\texttt{seller}?descr.$
$\texttt{seller}?price.$
$\mathrm{rec}\ X\ .\ ($
$\quad \texttt{seller}!offer.\texttt{seller}?price.X$
$\ \oplus\texttt{seller}!accept.\mathrm{end}$
$\ \oplus\texttt{seller}!quit.\mathrm{end}\ )$

From Global to Local

Back to our example:

$$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}; \texttt{seller} \xrightarrow{price} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$$

Actions composed by "$\wedge$" appear in some order.

But other orders are admitted

A possible implementation:

seller

$\texttt{buyer}!descr.$
$\texttt{buyer}!price.$
$\text{rec } X . ($
$\quad \texttt{buyer}?offer.\texttt{buyer}!price.X$
$\quad +\texttt{buyer}?accept.\text{end}$
$\quad +\texttt{buyer}?quit.\text{end})$

buyer

$\texttt{seller}?descr.$
$\texttt{seller}?price.$
$\text{rec } X . ($
$\quad \texttt{seller}!offer.\texttt{seller}?price.X$
$\quad \oplus\texttt{seller}!accept.\text{end}$
$\quad \oplus\texttt{seller}!quit.\text{end})$

Back to our example:

$(\texttt{seller} \xrightarrow{descr} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{price} \texttt{buyer})\,;$

$(\texttt{buyer} \xrightarrow{offer} \texttt{seller}\,;\texttt{seller} \xrightarrow{price} \texttt{buyer})*\,;$

$(\texttt{buyer} \xrightarrow{accept} \texttt{seller} \vee \texttt{buyer} \xrightarrow{quit} \texttt{seller})$

Actions composed by "$\wedge$" appear in some order.

But other orders are admitted

A possible implementation:

seller

$\texttt{buyer}\,\boldsymbol{!}\,price.$
$\texttt{buyer}\,\boldsymbol{!}\,descr.$
$\text{rec}\ X\ .\ ($
$\quad \texttt{buyer}\,\boldsymbol{?}\,offer.\texttt{buyer}\,\boldsymbol{!}\,price.X$
$\ +\texttt{buyer}\,\boldsymbol{?}\,accept.\text{end}$
$\ +\texttt{buyer}\,\boldsymbol{?}\,quit.\text{end}\ )$

buyer

$\texttt{seller}\,\boldsymbol{?}\,price.$
$\texttt{seller}\,\boldsymbol{?}\,descr.$
$\text{rec}\ X\ .\ ($
$\quad \texttt{seller}\,\boldsymbol{!}\,offer.\texttt{seller}\,\boldsymbol{?}\,price.X$
$\ \oplus\texttt{seller}\,\boldsymbol{!}\,accept.\text{end}$
$\ \oplus\texttt{seller}\,\boldsymbol{!}\,quit.\text{end}\ )$

Local Types and Projection

Implementations are specified by:

$$
\begin{array}{rclllcll}
T & ::= & \mathsf{end} & \text{(termination)} & & \mid & X & \text{(variable)} \\
& \mid & \mathtt{p}\,\mathbf{!}a.T & \text{(output)} & & \mid & \boldsymbol{\pi}\mathbf{?}a.T & \text{(input)} \\
& \mid & T \oplus T & \text{(internal choice)} & & \mid & T + T & \text{(external choice)} \\
& \mid & \mathsf{rec}\ X.T & \text{(recursion)} & & & &
\end{array}
$$

Local Types and Projection

Implementations are specified by:

$$
\begin{array}{rclll}
T & ::= & \mathsf{end} & \text{(termination)} & \mid \; X \quad \text{(variable)} \\
 & \mid & \mathtt{p}\,\mathbf{!}\,a.T & \text{(output)} & \mid \; \boldsymbol{\pi}\mathbf{?}a.T \quad \text{(input)} \\
 & \mid & T \oplus T & \text{(internal choice)} & \mid \; T + T \quad \text{(external choice)} \\
 & \mid & \mathsf{rec}\ X.T & \text{(recursion)} &
\end{array}
$$

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

Local Types and Projection

Implementations are specified by:

$$
\begin{array}{rclll}
T & ::= & \mathsf{end} & \text{(termination)} & \mid \; X \quad \text{(variable)} \\
& \mid & \mathtt{p}!a.T & \text{(output)} & \mid \; \pi?a.T \quad \text{(input)} \\
& \mid & T \oplus T & \text{(internal choice)} & \mid \; T + T \quad \text{(external choice)} \\
& \mid & \mathsf{rec}\ X.T & \text{(recursion)} &
\end{array}
$$

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 "$\wedge$" occur in any order (*shuffling*);
4. Communications of actions in "$\vee$" are mutually exclusive (*alternative*)

Soundness and completeness [first technical slide]

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

$$tr(\mathsf{skip}) = \{\varepsilon\} \qquad tr(\mathscr{G}_1;\mathscr{G}_2) = tr(\mathscr{G}_1)tr(\mathscr{G}_2)$$
$$tr(\pi \xrightarrow{a} \mathtt{p}) = \{\pi \xrightarrow{a} \mathtt{p}\} \qquad tr(\mathscr{G}_1 \vee \mathscr{G}_2) = tr(\mathscr{G}_1) \cup tr(\mathscr{G}_2)$$
$$tr(\mathscr{G}^*) = (tr(\mathscr{G}))^\star \qquad tr(\mathscr{G}_1 \wedge \mathscr{G}_2) = tr(\mathscr{G}_1) \sqcup\!\sqcup\, tr(\mathscr{G}_2)$$

Soundness and completeness [first technical slide]

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

$$tr(\mathsf{skip}) = \{\varepsilon\} \qquad tr(\mathscr{G}_1;\mathscr{G}_2) = tr(\mathscr{G}_1)tr(\mathscr{G}_2)$$
$$tr(\pi \xrightarrow{a} \mathtt{p}) = \{\pi \xrightarrow{a} \mathtt{p}\} \qquad tr(\mathscr{G}_1 \vee \mathscr{G}_2) = tr(\mathscr{G}_1) \cup tr(\mathscr{G}_2)$$
$$tr(\mathscr{G}^*) = (tr(\mathscr{G}))^\star \qquad tr(\mathscr{G}_1 \wedge \mathscr{G}_2) = tr(\mathscr{G}_1) \sqcup\!\sqcup\, tr(\mathscr{G}_2) \quad (\textit{shuffle})$$

Soundness and completeness [first technical slide]

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

$$tr(\mathsf{skip}) = \{\varepsilon\} \qquad tr(\mathcal{G}_1;\mathcal{G}_2) = tr(\mathcal{G}_1)tr(\mathcal{G}_2)$$
$$tr(\pi \xrightarrow{a} \mathtt{p}) = \{\pi \xrightarrow{a} \mathtt{p}\} \qquad tr(\mathcal{G}_1 \vee \mathcal{G}_2) = tr(\mathcal{G}_1) \cup tr(\mathcal{G}_2)$$
$$tr(\mathcal{G}^*) = (tr(\mathcal{G}))^\star \qquad tr(\mathcal{G}_1 \wedge \mathcal{G}_2) = tr(\mathcal{G}_1) \sqcup\!\sqcup\, tr(\mathcal{G}_2)$$

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

Soundness and completeness [first technical slide]

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

$$tr(\mathtt{skip}) = \{\varepsilon\} \qquad tr(\mathscr{G}_1;\mathscr{G}_2) = tr(\mathscr{G}_1)tr(\mathscr{G}_2)$$
$$tr(\pi \xrightarrow{a} \mathtt{p}) = \{\pi \xrightarrow{a} \mathtt{p}\} \qquad tr(\mathscr{G}_1 \vee \mathscr{G}_2) = tr(\mathscr{G}_1) \cup tr(\mathscr{G}_2)$$
$$tr(\mathscr{G}^*) = (tr(\mathscr{G}))^\star \qquad tr(\mathscr{G}_1 \wedge \mathscr{G}_2) = tr(\mathscr{G}_1) \sqcup tr(\mathscr{G}_2)$$

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

$$\left[\begin{array}{c} \mathbb{B} \\ \{\ldots, \mathtt{p} : \bigoplus_{i\in I} \mathtt{p}_i!a_i.T_i, \ldots\} \end{array}\right] \longrightarrow \left[\begin{array}{c} (\mathtt{p} \xrightarrow{a_k} \mathtt{p}_k)::\mathbb{B} \\ \{\ldots, \mathtt{p} : T_k, \ldots\} \end{array}\right] \quad (k\in I)$$

Soundness and completeness [first technical slide]

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

$$tr(\mathsf{skip}) = \{\varepsilon\} \qquad tr(\mathscr{G}_1;\mathscr{G}_2) = tr(\mathscr{G}_1)tr(\mathscr{G}_2)$$
$$tr(\pi \xrightarrow{a} \mathtt{p}) = \{\pi \xrightarrow{a} \mathtt{p}\} \qquad tr(\mathscr{G}_1 \vee \mathscr{G}_2) = tr(\mathscr{G}_1) \cup tr(\mathscr{G}_2)$$
$$tr(\mathscr{G}^*) = (tr(\mathscr{G}))^\star \qquad tr(\mathscr{G}_1 \wedge \mathscr{G}_2) = tr(\mathscr{G}_1) \sqcup tr(\mathscr{G}_2)$$

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

$$\left[\begin{array}{c}\mathbb{B}\\ \{\dots,\mathtt{p}:\bigoplus_{i\in I}\mathtt{p}_i\,!\,a_i.T_i,\dots\}\end{array}\right] \longrightarrow \left[\begin{array}{c}(\mathtt{p}\xrightarrow{a_k}\mathtt{p}_k)::\mathbb{B}\\ \{\dots,\mathtt{p}:T_k,\dots\}\end{array}\right] \quad (k\in I)$$

$$\left[\begin{array}{c}\mathbb{B}::(\mathtt{p}_i\xrightarrow{a}\mathtt{p})_{i\in I}\\ \{\dots,\mathtt{p}:\sum_{j\in J}\pi_j\,?\,a_j.T_j,\dots\}\end{array}\right] \xrightarrow{\pi_k\xrightarrow{a}\mathtt{p}} \left[\begin{array}{c}\mathbb{B}\\ \{\dots,\mathtt{p}:T_k,\dots\}\end{array}\right] \quad (\pi_k=\{\mathtt{p}_i\}_{i\in I})$$

Soundness and completeness [first technical slide]

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

$$tr(\text{skip}) = \{\varepsilon\} \qquad tr(\mathscr{G}_1;\mathscr{G}_2) = tr(\mathscr{G}_1)tr(\mathscr{G}_2)$$
$$tr(\pi \xrightarrow{a} \mathtt{p}) = \{\pi \xrightarrow{a} \mathtt{p}\} \qquad tr(\mathscr{G}_1 \vee \mathscr{G}_2) = tr(\mathscr{G}_1) \cup tr(\mathscr{G}_2)$$
$$tr(\mathscr{G}^*) = (tr(\mathscr{G}))^\star \qquad tr(\mathscr{G}_1 \wedge \mathscr{G}_2) = tr(\mathscr{G}_1) \sqcup tr(\mathscr{G}_2)$$

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

$$\left[\begin{array}{c}\mathbb{B}\\ \{\dots,\mathtt{p}:\bigoplus_{i\in I}\mathtt{p}_i!a_i.T_i,\dots\}\end{array}\right] \longrightarrow \left[\begin{array}{c}(\mathtt{p}\xrightarrow{a_k}\mathtt{p}_k)::\mathbb{B}\\ \{\dots,\mathtt{p}:T_k,\dots\}\end{array}\right] \quad (k\in I)$$

$$\left[\begin{array}{c}\mathbb{B}::(\mathtt{p}_i\xrightarrow{a}\mathtt{p})_{i\in I}\\ \{\dots,\mathtt{p}:\sum_{j\in J}\pi_j?a_j.T_j,\dots\}\end{array}\right] \xrightarrow{\pi_k\xrightarrow{a}\mathtt{p}} \left[\begin{array}{c}\mathbb{B}\\ \{\dots,\mathtt{p}:T_k,\dots\}\end{array}\right] \quad (\pi_k=\{\mathtt{p}_i\}_{i\in I})$$

3. **Soundness:** $tr(\{\mathtt{p}_i:T_i\}_{i\in I}) \subseteq tr(\mathscr{G})$
every trace of $\{\mathtt{p}_i:T_i\}_{i\in I}$ is a trace of $\mathscr{G}$

Soundness and completeness [first technical slide]

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

$$tr(\text{skip}) = \{\varepsilon\} \qquad tr(\mathcal{G}_1;\mathcal{G}_2) = tr(\mathcal{G}_1)tr(\mathcal{G}_2)$$
$$tr(\pi \xrightarrow{a} \mathrm{p}) = \{\pi \xrightarrow{a} \mathrm{p}\} \qquad tr(\mathcal{G}_1 \vee \mathcal{G}_2) = tr(\mathcal{G}_1) \cup tr(\mathcal{G}_2)$$
$$tr(\mathcal{G}^*) = (tr(\mathcal{G}))^\star \qquad tr(\mathcal{G}_1 \wedge \mathcal{G}_2) = tr(\mathcal{G}_1) \sqcup\!\sqcup\, tr(\mathcal{G}_2)$$

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

$$\left[\begin{array}{c}\mathbb{B}\\ \{\ldots,\mathrm{p} : \bigoplus_{i\in I}\mathrm{p}_i!a_i.T_i,\ldots\}\end{array}\right] \longrightarrow \left[\begin{array}{c}(\mathrm{p}\xrightarrow{a_k}\mathrm{p}_k)::\mathbb{B}\\ \{\ldots,\mathrm{p} : T_k,\ldots\}\end{array}\right] \quad (k\in I)$$

$$\left[\begin{array}{c}\mathbb{B}::(\mathrm{p}_i\xrightarrow{a}\mathrm{p})_{i\in I}\\ \{\ldots,\mathrm{p}:\sum_{j\in J}\pi_j?a_j.T_j,\ldots\}\end{array}\right] \xrightarrow{\pi_k\xrightarrow{a}\mathrm{p}} \left[\begin{array}{c}\mathbb{B}\\ \{\ldots,\mathrm{p} : T_k,\ldots\}\end{array}\right] \quad (\pi_k=\{\mathrm{p}_i\}_{i\in I})$$

3. **Soundness:** $tr(\{\mathrm{p}_i{:}T_i\}_{i\in I}) \subseteq tr(\mathcal{G})$
every trace of $\{\mathrm{p}_i{:}T_i\}_{i\in I}$ is a trace of $\mathcal{G}$

4. **Completeness:** $tr(\mathcal{G}) \subseteq tr(\{\mathrm{p}_i{:}T_i\}_{i\in I})^\circ$:
every trace of $\mathcal{G}$ is the *permutation* of a trace of $\{\mathrm{p}_i{:}T_i\}_{i\in I}$.

$L^\circ \stackrel{def}{=} \{\alpha_1\cdots\alpha_n \mid \exists$ a permutation σ 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:

$$(\mathtt{p} \xrightarrow{a} \mathtt{q}; \mathtt{r} \xrightarrow{b} \mathtt{s})$$

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:

$$(\mathtt{p} \xrightarrow{a} \mathtt{q}; \mathtt{r} \xrightarrow{b} \mathtt{s})$$

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

$$\mathtt{p} \xrightarrow{a} \mathtt{q} \vee \mathtt{q} \xrightarrow{b} \mathtt{p}$$

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:

$$(\mathtt{p} \xrightarrow{a} \mathtt{q}; \mathtt{r} \xrightarrow{b} \mathtt{s})$$

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

$$\mathtt{p} \xrightarrow{a} \mathtt{q} \vee \mathtt{q} \xrightarrow{b} \mathtt{p}$$

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

$$\begin{array}{c}(\mathtt{p} \xrightarrow{a} \mathtt{q}; \mathtt{q} \xrightarrow{a} \mathtt{r}; \mathtt{r} \xrightarrow{a} \mathtt{p})\\ \vee \\ (\mathtt{p} \xrightarrow{b} \mathtt{q}; \mathtt{q} \xrightarrow{a} \mathtt{r}; \mathtt{r} \xrightarrow{b} \mathtt{p})\end{array}$$

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:

$$(\mathtt{p} \xrightarrow{a} \mathtt{q}; \mathtt{r} \xrightarrow{b} \mathtt{s})$$

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

$$\mathtt{p} \xrightarrow{a} \mathtt{q} \vee \mathtt{q} \xrightarrow{b} \mathtt{p}$$

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

$$\begin{array}{c}(\mathtt{p} \xrightarrow{a} \mathtt{q}; \mathtt{q} \xrightarrow{a} \mathtt{r}; \mathtt{r} \xrightarrow{a} \mathtt{p}) \\ \vee \\ (\mathtt{p} \xrightarrow{b} \mathtt{q}; \mathtt{q} \xrightarrow{a} \mathtt{r}; \mathtt{r} \xrightarrow{b} \mathtt{p})\end{array}$$

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

$$\begin{array}{ll}(& \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer1}; \texttt{buyer1} \xrightarrow{\textit{price}} \texttt{buyer2} \\ \vee & \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer2}; \texttt{buyer2} \xrightarrow{\textit{price}} \texttt{buyer1})\end{array}$$

Examples

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

- same message different receivers in a choice

$$\begin{array}{l}(\quad \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer1}; \texttt{buyer1} \xrightarrow{\textit{price}} \texttt{buyer2} \\ \vee \ \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer2}; \texttt{buyer2} \xrightarrow{\textit{price}} \texttt{buyer1})\end{array}$$

- different receivers to break a loop

$$\begin{array}{l}\texttt{seller} \xrightarrow{\textit{agency}} \texttt{broker}; \\ (\texttt{broker} \xrightarrow{\textit{offer}} \texttt{buyer}; \texttt{buyer} \xrightarrow{\textit{counteroffer}} \texttt{broker})*; \\ (\texttt{broker} \xrightarrow{\textit{result}} \texttt{seller} \wedge \texttt{broker} \xrightarrow{\textit{result}} \texttt{buyer})\end{array}$$

Projection [last technical slide]

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)

$$\Delta \vdash \mathsf{skip} \triangleright \Delta$$

(SP-ACTION)

$$\{\mathrm{p} : S, \mathrm{q} : T, \dots\} \vdash \mathrm{p} \xrightarrow{a} \mathrm{q} \triangleright \{\mathrm{p} : \mathrm{q}!a.S, \mathrm{q} : \mathrm{p}?a.T, \dots\}$$

(SP-SEQUENCE)

$$\frac{\Delta \vdash \mathscr{G}_2 \triangleright \Delta' \qquad \Delta' \vdash \mathscr{G}_1 \triangleright \Delta''}{\Delta \vdash \mathscr{G}_1; \mathscr{G}_2 \triangleright \Delta''}$$

(SP-ITERATION)

$$\frac{\{\mathrm{p} : T_1 \oplus T_2, \dots\} \vdash \mathscr{G} \triangleright \{\mathrm{p} : T_1, \dots\}}{\{\mathrm{p} : T_2, \dots\} \vdash \mathscr{G}^* \triangleright \{\mathrm{p} : T_1 \oplus T_2, \dots\}}$$

(SP-ALTERNATIVE)

$$\frac{\Delta \vdash \mathscr{G}_1 \triangleright \{\mathrm{p} : T_1, \dots\} \qquad \Delta \vdash \mathscr{G}_2 \triangleright \{\mathrm{p} : T_2, \dots\}}{\Delta \vdash \mathscr{G}_1 \vee \mathscr{G}_2 \triangleright \{\mathrm{p} : T_1 \oplus T_2, \dots\}}$$

(SP-SUBSUMPTION)

$$\frac{\Delta \vdash \mathscr{G}' \triangleright \Delta' \qquad \mathscr{G}' \leqslant \mathscr{G} \qquad \Delta'' \leqslant \Delta'}{\Delta \vdash \mathscr{G} \triangleright \Delta''}$$

$$(X \leqslant Y \stackrel{def}{=} tr(L) \subseteq tr(M) \subseteq tr(L)^\circ)$$

Global types not inherently flawed are associated to sound and complete sets of components compositionally by a deduction system

(SP-Skip)
$$\Delta \vdash \mathsf{skip} \triangleright \Delta$$

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

(SP-Sequence)
$$\frac{\Delta \vdash \mathscr{G}_2 \triangleright \Delta' \qquad \Delta' \vdash \mathscr{G}_1 \triangleright \Delta''}{\Delta \vdash \mathscr{G}_1 ; \mathscr{G}_2 \triangleright \Delta''}$$

(SP-Iteration)
$$\frac{\{\mathtt{p} : T_1 \oplus T_2, \ldots\} \vdash \mathscr{G} \triangleright \{\mathtt{p} : T_1, \ldots\}}{\{\mathtt{p} : T_2, \ldots\} \vdash \mathscr{G}^* \triangleright \{\mathtt{p} : T_1 \oplus T_2, \ldots\}}$$

(SP-Alternative)
$$\frac{\Delta \vdash \mathscr{G}_1 \triangleright \{\mathtt{p} : T_1, \ldots\} \qquad \Delta \vdash \mathscr{G}_2 \triangleright \{\mathtt{p} : T_2, \ldots\}}{\Delta \vdash \mathscr{G}_1 \vee \mathscr{G}_2 \triangleright \{\mathtt{p} : T_1 \oplus T_2, \ldots\}}$$

(SP-Subsumption)
$$\frac{\Delta \vdash \mathscr{G}' \triangleright \Delta' \qquad \mathscr{G}' \leqslant \mathscr{G} \qquad \Delta'' \leqslant \Delta'}{\Delta \vdash \mathscr{G} \triangleright \Delta''}$$

$$(X \leqslant Y \stackrel{def}{=} tr(L) \subseteq tr(M) \subseteq tr(L)^{\circ})$$

Global types not inherently flawed are associated to sound and complete sets of components compositionally by a deduction system

(SP-Skip)

$$\Delta \vdash \mathsf{skip} \ \triangleright \ \Delta$$

(SP-Action)

$$\{\mathsf{p}: S, \mathsf{q}: T, \ldots\} \vdash \mathsf{p} \xrightarrow{a} \mathsf{q} \ \triangleright \ \{\mathsf{p}: \mathsf{q}!a.S, \mathsf{q}: \mathsf{p}?a.T, \ldots\}$$

(SP-Sequence)

$$\frac{\Delta \vdash \mathscr{G}_2 \ \triangleright \ \Delta' \qquad \Delta' \vdash \mathscr{G}_1 \ \triangleright \ \Delta''}{\Delta \vdash \mathscr{G}_1 ; \mathscr{G}_2 \ \triangleright \ \Delta''}$$

(SP-Iteration)

$$\frac{\{\mathsf{p}: T_1 \oplus T_2, \ldots\} \vdash \mathscr{G} \ \triangleright \ \{\mathsf{p}: T_1, \ldots\}}{\{\mathsf{p}: T_2, \ldots\} \vdash \mathscr{G}^* \ \triangleright \ \{\mathsf{p}: T_1 \oplus T_2, \ldots\}}$$

(SP-Alternative)

$$\frac{\Delta \vdash \mathscr{G}_1 \ \triangleright \ \{\mathsf{p}: T_1, \ldots\} \qquad \Delta \vdash \mathscr{G}_2 \ \triangleright \ \{\mathsf{p}: T_2, \ldots\}}{\Delta \vdash \mathscr{G}_1 \vee \mathscr{G}_2 \ \triangleright \ \{\mathsf{p}: T_1 \oplus T_2, \ldots\}}$$

(SP-Subsumption)

$$\frac{\Delta \vdash \mathscr{G}' \ \triangleright \ \Delta' \qquad \mathscr{G}' \leqslant \mathscr{G} \qquad \Delta'' \leqslant \Delta'}{\Delta \vdash \mathscr{G} \ \triangleright \ \Delta''}$$

$$(X \leqslant Y \stackrel{def}{=} tr(L) \subseteq tr(M) \subseteq tr(L)^{\circ})$$

Global types not inherently flawed are associated to sound and complete sets of components compositionally by a deduction system

(SP-Skip)

$$\Delta \vdash \mathsf{skip} \ \triangleright\ \Delta$$

(SP-Action)

$$\{\mathsf{p} : S, \mathsf{q} : T, \dots\} \vdash \mathsf{p} \xrightarrow{a} \mathsf{q} \ \triangleright\ \{\mathsf{p} : \mathsf{q}!a.S, \mathsf{q} : \mathsf{p}?a.T, \dots\}$$

(SP-Sequence)

$$\frac{\Delta \vdash \mathscr{G}_2 \ \triangleright\ \Delta' \qquad \Delta' \vdash \mathscr{G}_1 \ \triangleright\ \Delta''}{\Delta \vdash \mathscr{G}_1 ; \mathscr{G}_2 \ \triangleright\ \Delta''}$$

(SP-Iteration)

$$\frac{\{\mathsf{p} : T_1 \oplus T_2, \dots\} \vdash \mathscr{G} \ \triangleright\ \{\mathsf{p} : T_1, \dots\}}{\{\mathsf{p} : T_2, \dots\} \vdash \mathscr{G}^* \ \triangleright\ \{\mathsf{p} : T_1 \oplus T_2, \dots\}}$$

(SP-Alternative)

$$\frac{\Delta \vdash \mathscr{G}_1 \ \triangleright\ \{\mathsf{p} : T_1, \dots\} \qquad \Delta \vdash \mathscr{G}_2 \ \triangleright\ \{\mathsf{p} : T_2, \dots\}}{\Delta \vdash \mathscr{G}_1 \vee \mathscr{G}_2 \ \triangleright\ \{\mathsf{p} : T_1 \oplus T_2, \dots\}}$$

(SP-Subsumption)

$$\frac{\Delta \vdash \mathscr{G}' \ \triangleright\ \Delta' \qquad \mathscr{G}' \leqslant \mathscr{G} \qquad \Delta'' \leqslant \Delta'}{\Delta \vdash \mathscr{G} \ \triangleright\ \Delta''}$$

$$(X \leqslant Y \stackrel{def}{=} tr(L) \subseteq tr(M) \subseteq tr(L)^{\circ})$$

Projection [last technical slide]

Global types not inherently flawed are associated to sound and complete sets of components compositionally by a deduction system

(SP-Skip)

$$\Delta \vdash \mathsf{skip} \triangleright \Delta$$

(SP-Action)

$$\{\mathrm{p} : S, \mathrm{q} : T, \ldots\} \vdash \mathrm{p} \xrightarrow{a} \mathrm{q} \triangleright \{\mathrm{p} : \mathrm{q}!a.S, \mathrm{q} : \mathrm{p}?a.T, \ldots\}$$

(SP-Sequence)

$$\frac{\Delta \vdash \mathscr{G}_2 \triangleright \Delta' \qquad \Delta' \vdash \mathscr{G}_1 \triangleright \Delta''}{\Delta \vdash \mathscr{G}_1 ; \mathscr{G}_2 \triangleright \Delta''}$$

(SP-Iteration)

$$\frac{\{\mathrm{p} : T_1 \oplus T_2, \ldots\} \vdash \mathscr{G} \triangleright \{\mathrm{p} : T_1, \ldots\}}{\{\mathrm{p} : T_2, \ldots\} \vdash \mathscr{G}^* \triangleright \{\mathrm{p} : T_1 \oplus T_2, \ldots\}}$$

(SP-Alternative)

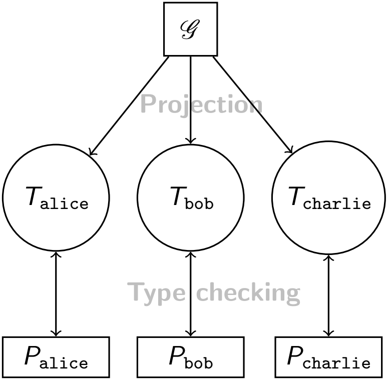
$$\frac{\Delta \vdash \mathscr{G}_1 \triangleright \{\mathrm{p} : T_1, \ldots\} \qquad \Delta \vdash \mathscr{G}_2 \triangleright \{\mathrm{p} : T_2, \ldots\}}{\Delta \vdash \mathscr{G}_1 \vee \mathscr{G}_2 \triangleright \{\mathrm{p} : T_1 \oplus T_2, \ldots\}}$$

(SP-Subsumption)

$$\frac{\Delta \vdash \mathscr{G}' \triangleright \Delta' \qquad \mathscr{G}' \leqslant \mathscr{G} \qquad \Delta'' \leqslant \Delta'}{\Delta \vdash \mathscr{G} \triangleright \Delta''}$$

Makes projection algorithm very hard (see proceedings)

A three-layered structure



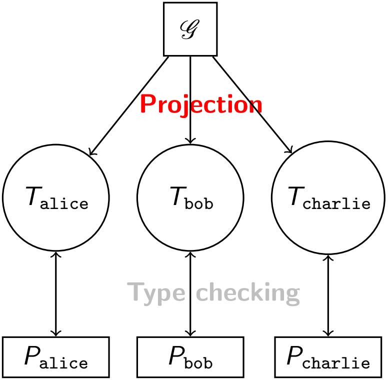
Global Type $\mathscr{G} = \texttt{alice} \xrightarrow{\mathit{nat}} \texttt{bob};\ \texttt{bob} \xrightarrow{\mathit{nat}} \texttt{charlie}$

Local Types $T_{\texttt{bob}} = \texttt{alice?}\mathit{nat}.\ \texttt{charlie!}\mathit{nat}.\ \mathsf{end}$

Processes $P_{\texttt{bob}} =$

```
receive x from alice;
send x+42 to charlie;
end
```

A three-layered structure



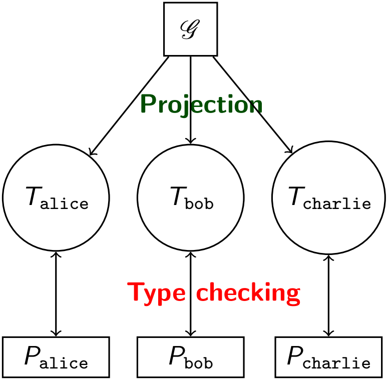
Global Type $\mathscr{G} = \texttt{alice} \xrightarrow{nat} \texttt{bob};\ \texttt{bob} \xrightarrow{nat} \texttt{charlie}$

Local Types $T_{\texttt{bob}} = \texttt{alice?}nat.\ \texttt{charlie!}nat.\ \mathsf{end}$

Processes $P_{\texttt{bob}} =$

```
receive x from alice;
send x+42 to charlie;
end
```

A three-layered structure

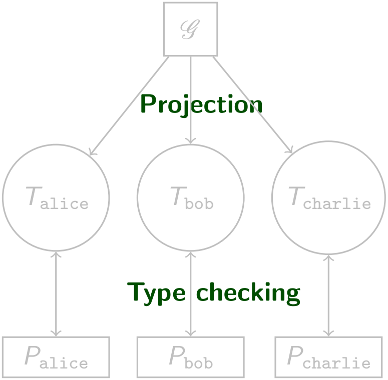


Global Type $\mathscr{G} = \texttt{alice} \xrightarrow{\mathit{nat}} \texttt{bob};$
$\texttt{bob} \xrightarrow{\mathit{nat}} \texttt{charlie}$

Local Types $T_{\texttt{bob}} = \texttt{alice}\textbf{?}\mathit{nat}.$
$\texttt{charlie}\textbf{!}\mathit{nat}.$
end

Processes $P_{\texttt{bob}} =$ `receive x from alice;`
`send x+42 to charlie;`
`end`

A three-layered structure



Global Type $\mathscr{G} =$ alice $\xrightarrow{nat}$ bob; bob $\xrightarrow{nat}$ charlie

soundness
completeness

Local Types $T_{\texttt{bob}} =$ alice?*nat*. charlie!*nat*. end

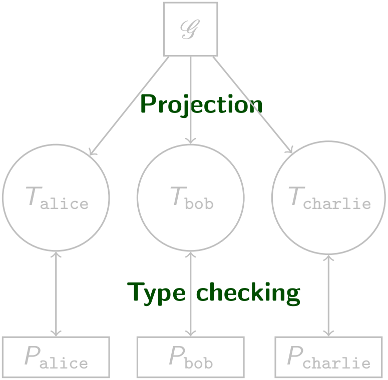
subject reduction
progress, fairness

Processes $P_{\texttt{bob}} =$ `receive x from alice; send x+42 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 $\mathscr{G} = \texttt{alice} \xrightarrow{nat} \texttt{bob};$
$\texttt{bob} \xrightarrow{nat} \texttt{charlie}$

soundness
completeness

Local Types $T_{\texttt{bob}} = \texttt{alice?}\textit{nat}.$
$\texttt{charlie!}\textit{nat}.$
end

subject reduction
progress, fairness

Processes $P_{\texttt{bob}} =$ `receive x from alice;`
`send x+42 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

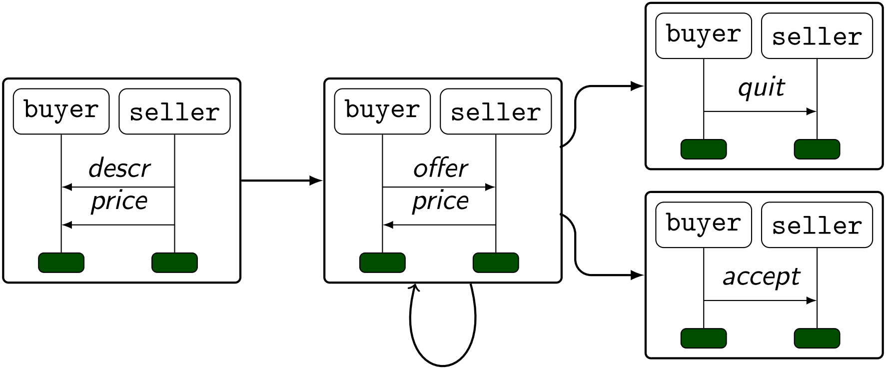
Automata approach: global specifications

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.

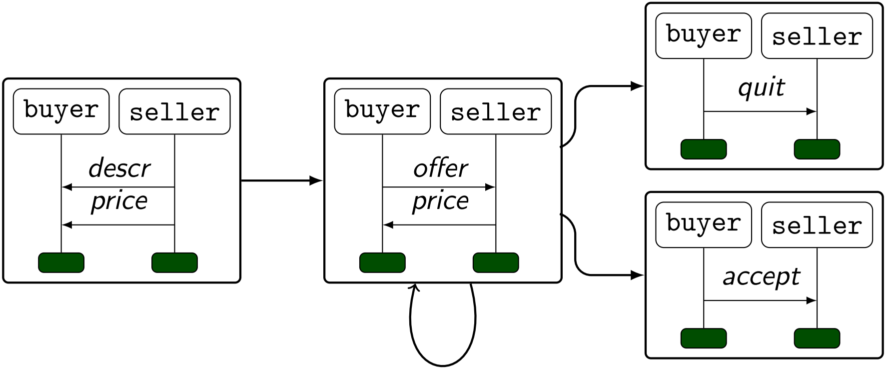
Message Sequence Graphs:



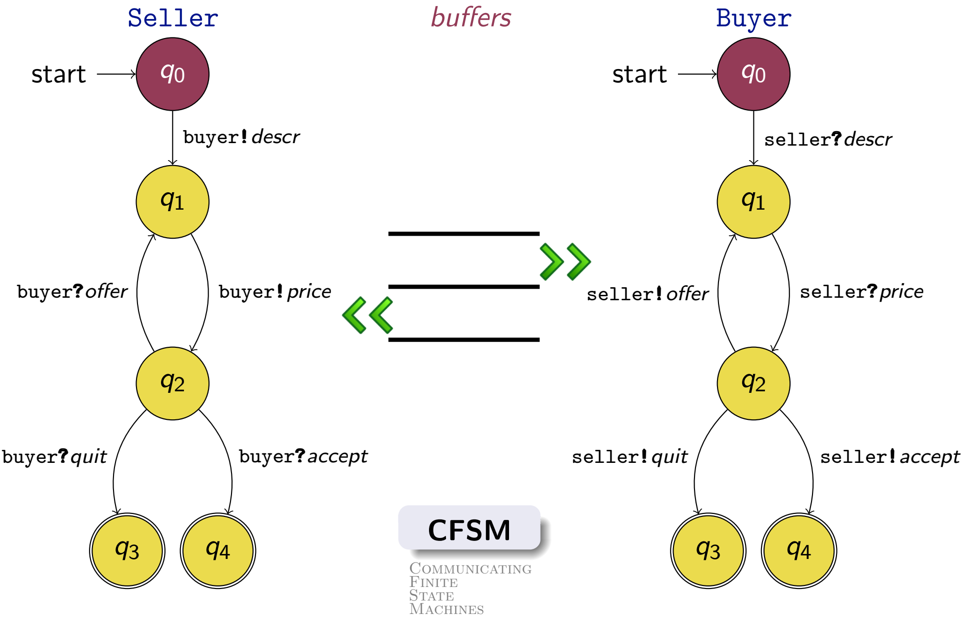
Automata approach: global specifications

$$(\texttt{seller} \xrightarrow{\textit{descr}} \texttt{buyer} \wedge \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer});$$
$$(\texttt{buyer} \xrightarrow{\textit{offer}} \texttt{seller}; \texttt{seller} \xrightarrow{\textit{price}} \texttt{buyer})*;$$
$$(\texttt{buyer} \xrightarrow{\textit{accept}} \texttt{seller} \vee \texttt{buyer} \xrightarrow{\textit{quit}} \texttt{seller})$$

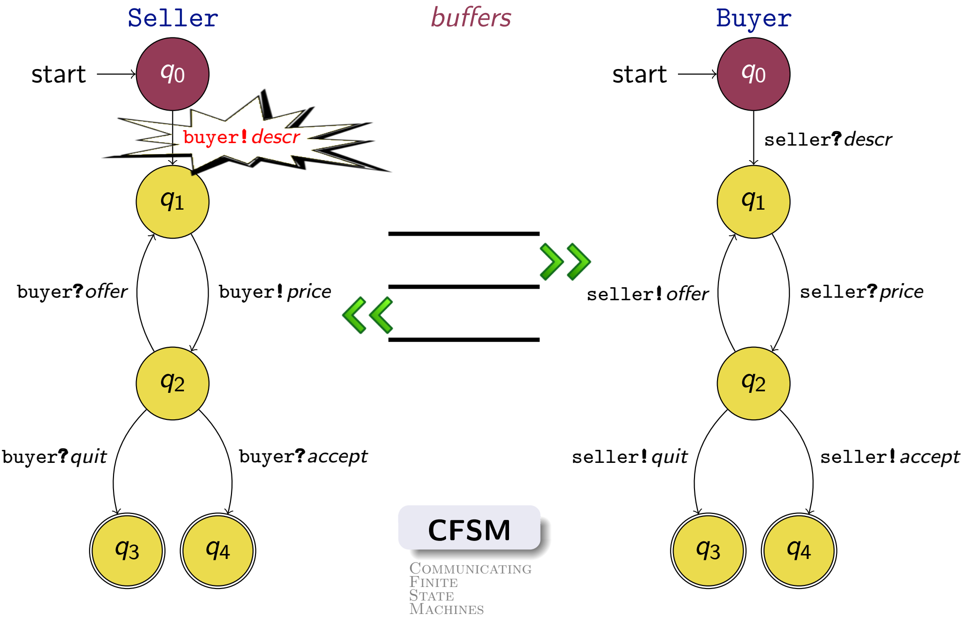
Message Sequence Graphs:



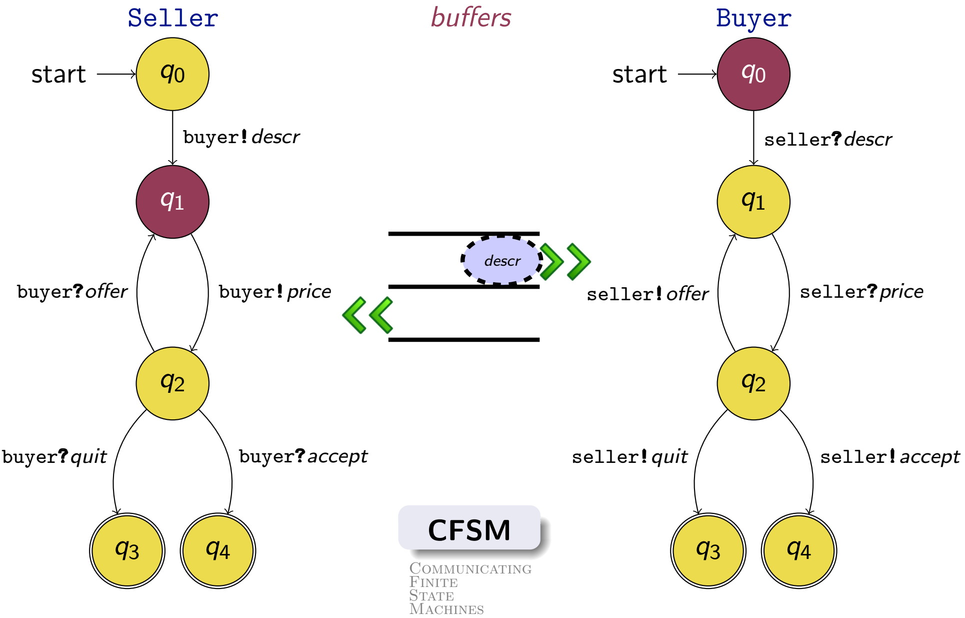
Automata approach: local specifications



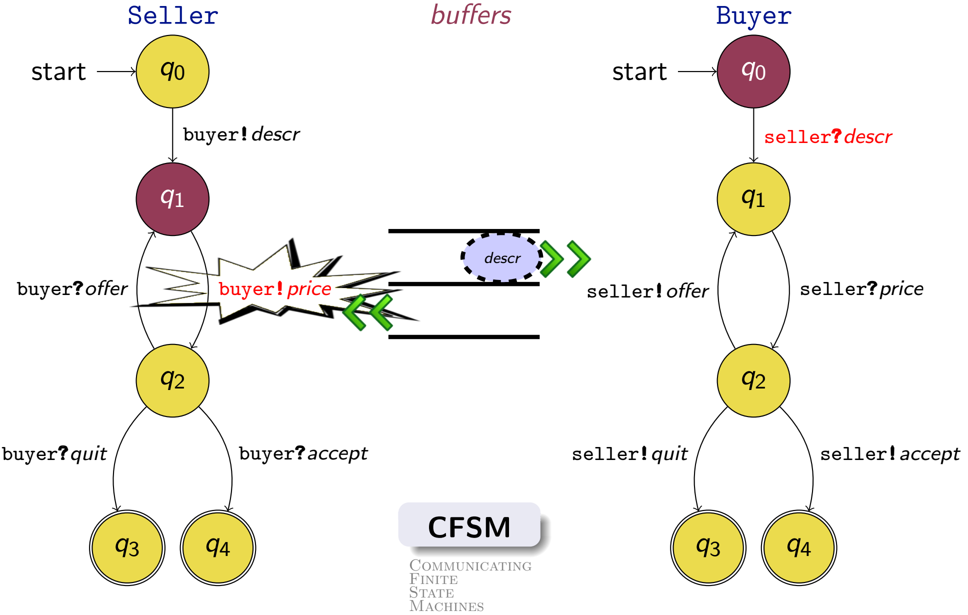
Automata approach: local specifications



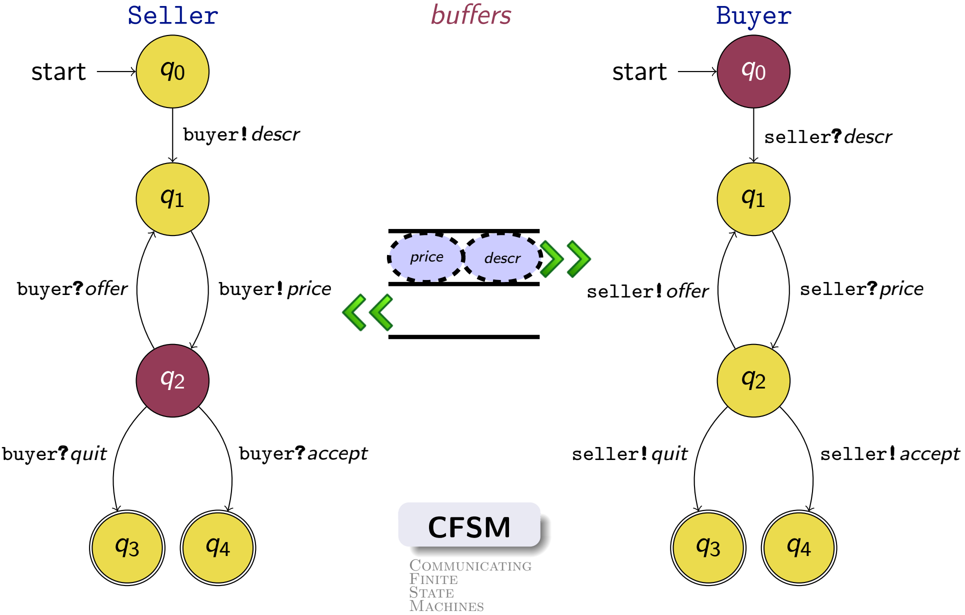
Automata approach: local specifications



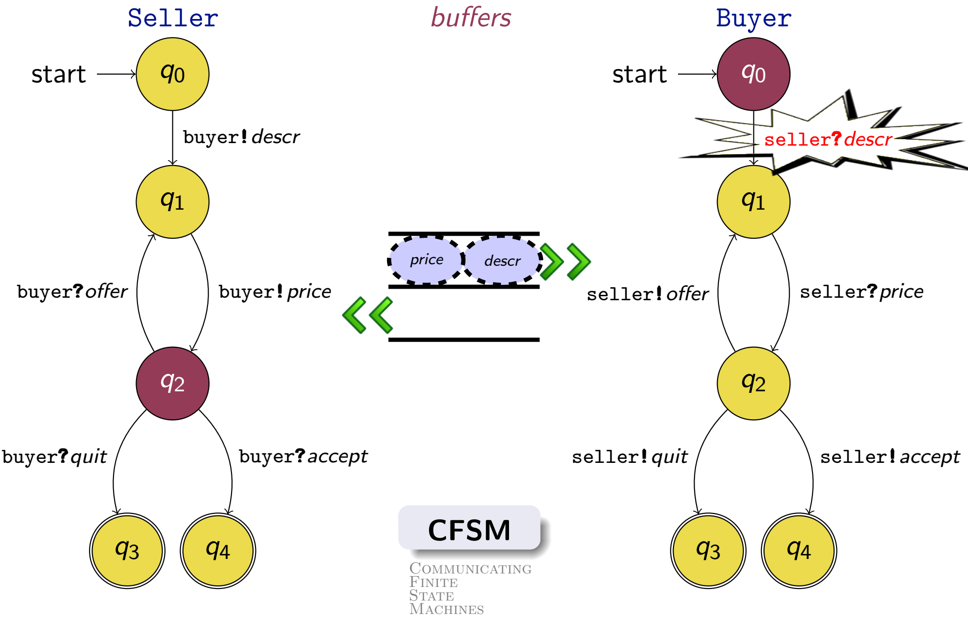
Automata approach: local specifications



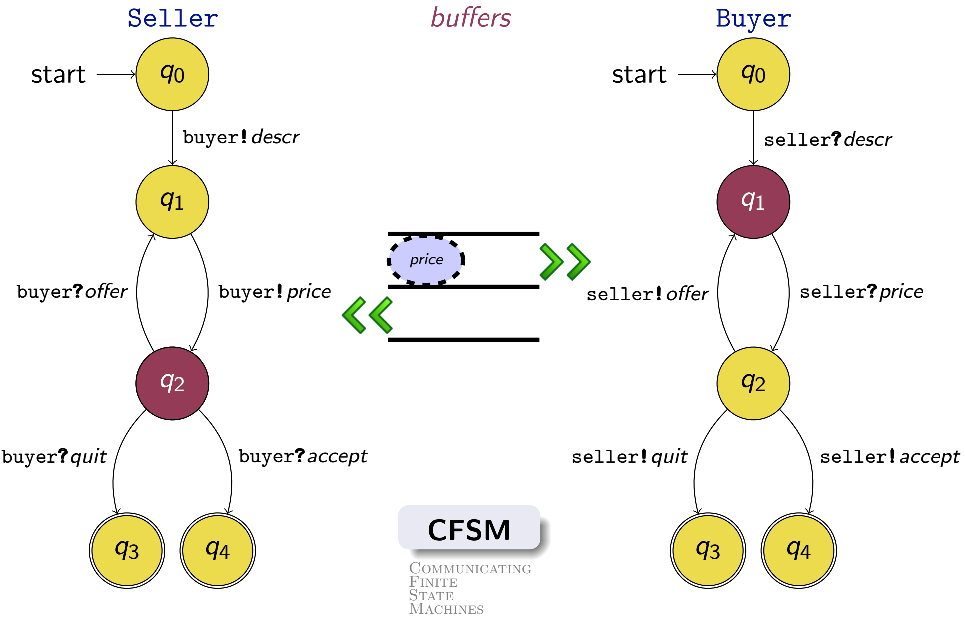
Automata approach: local specifications



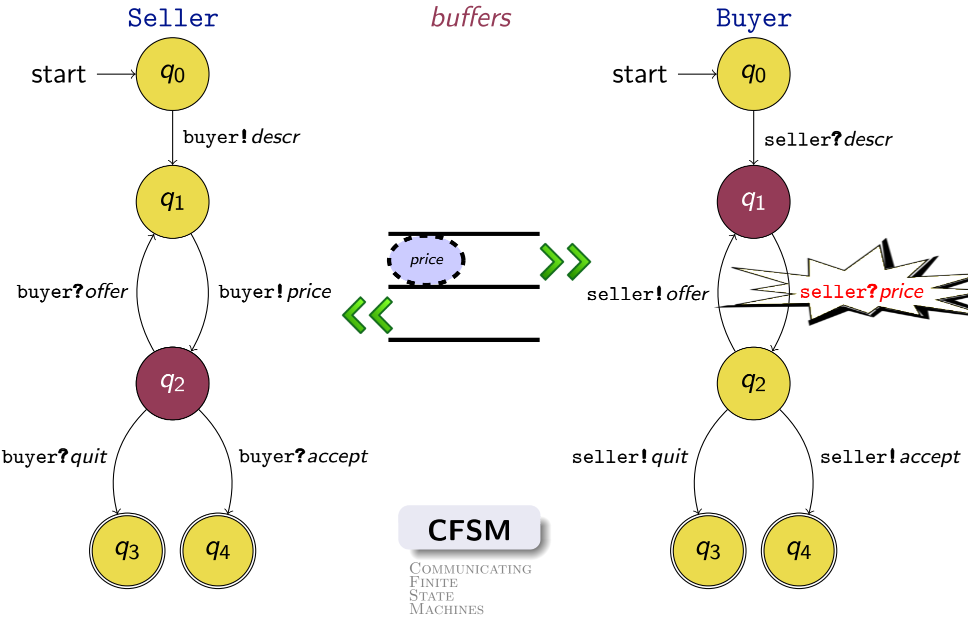
Automata approach: local specifications



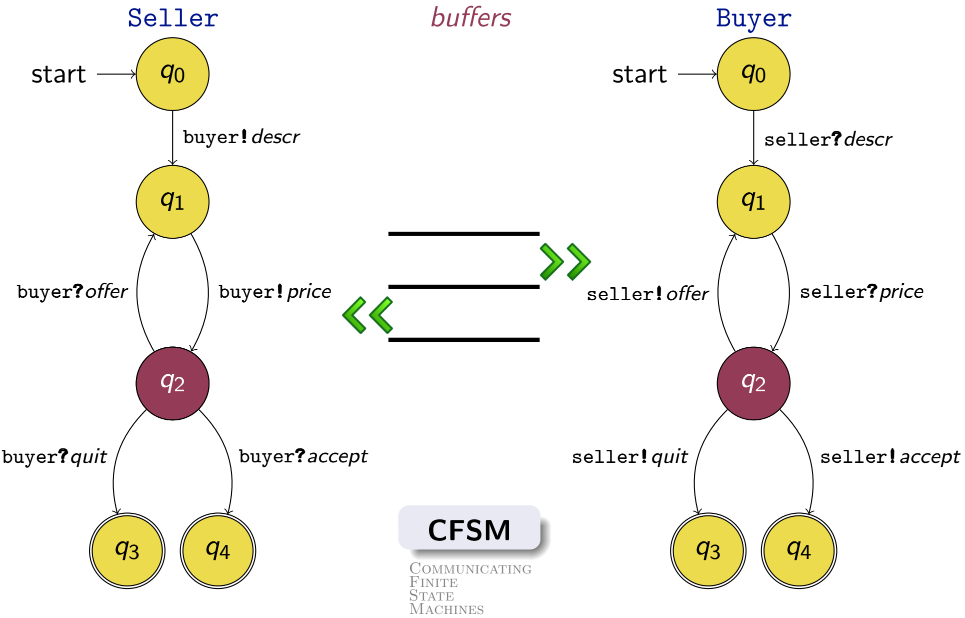
Automata approach: local specifications



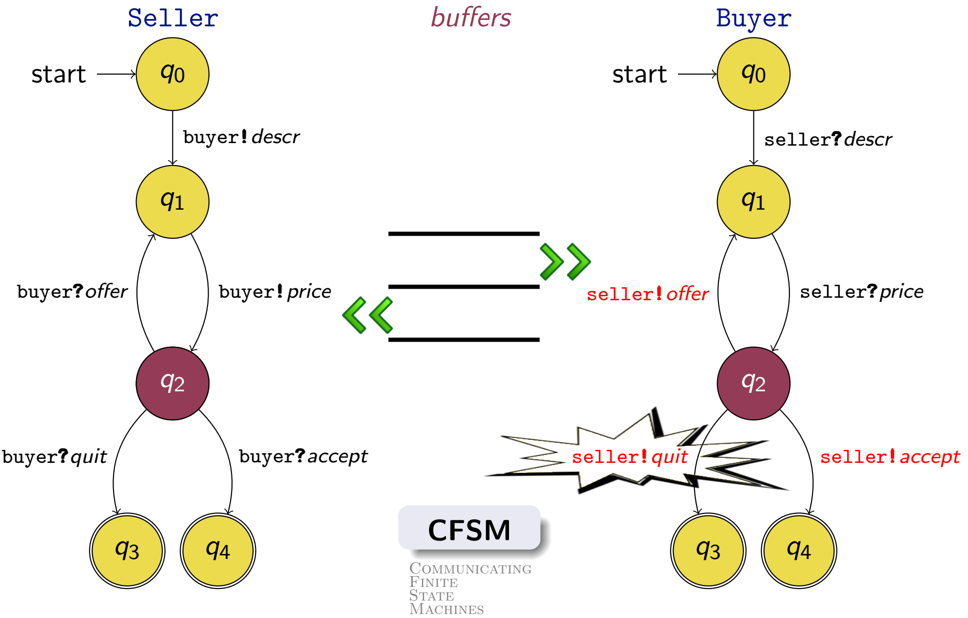
Automata approach: local specifications



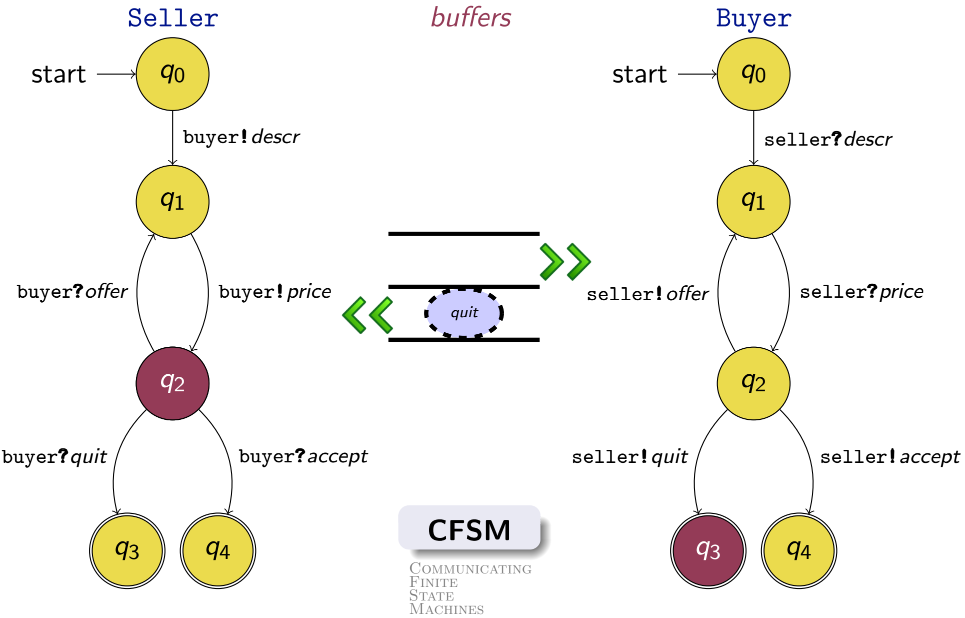
Automata approach: local specifications



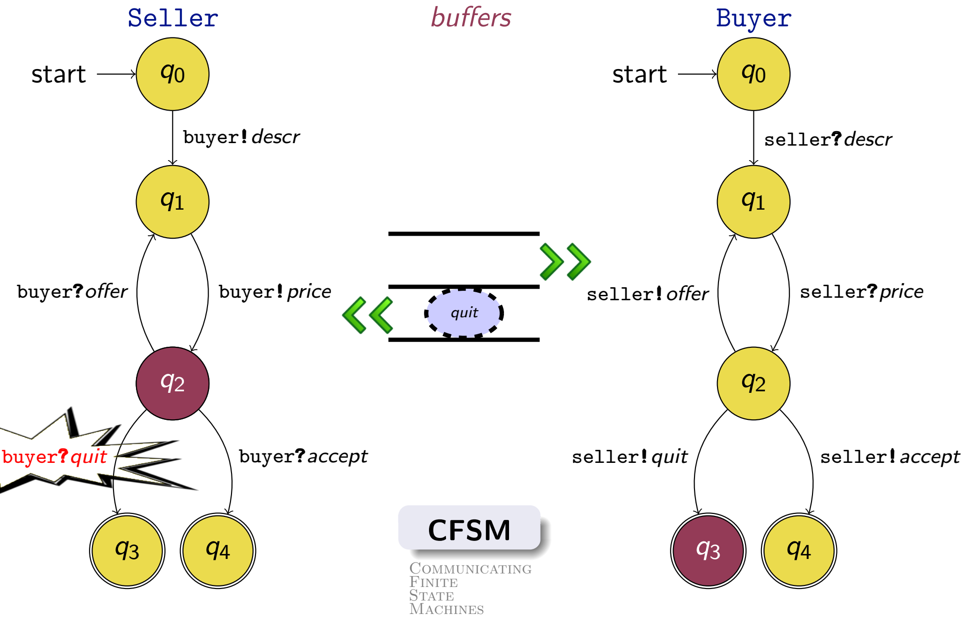
Automata approach: local specifications



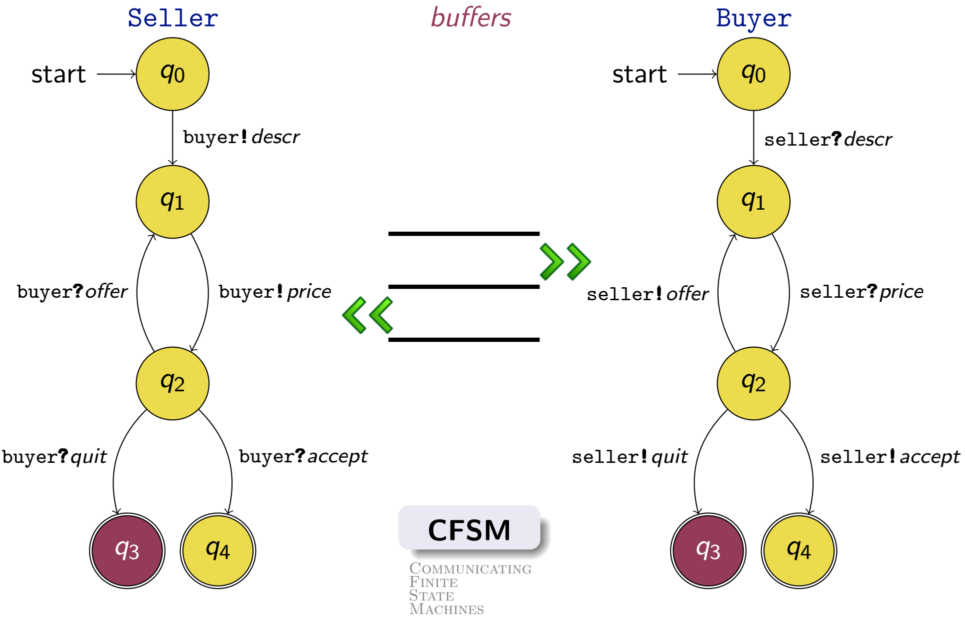
Automata approach: local specifications



Automata approach: local specifications



Automata approach: local specifications



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 (*eg*, 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* (*eg*, unsound implementations, implementations with a controlled use of covert channels, implementation admitting deadlocks) to obtain decidability and/or polynomial complexity.

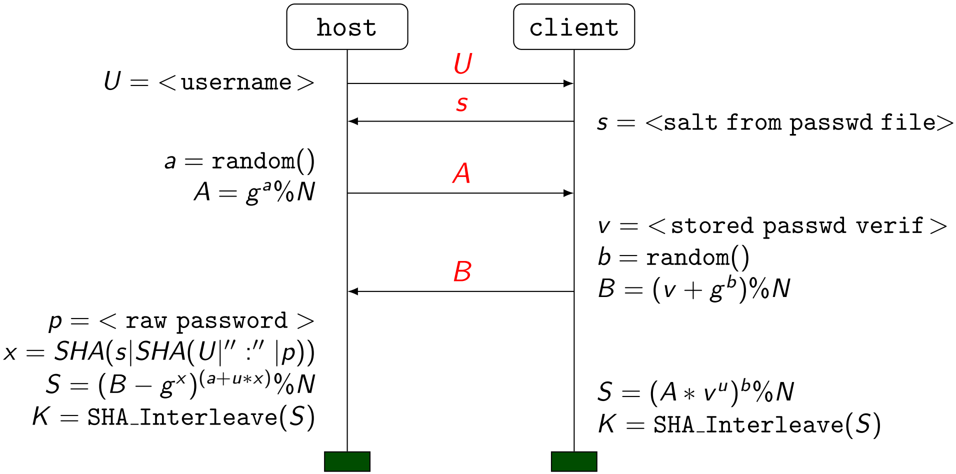
The protocol approach: global specifications

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

RFC 2945 (SRP Authentication and Key Exchange System)

The protocol approach: global specifications

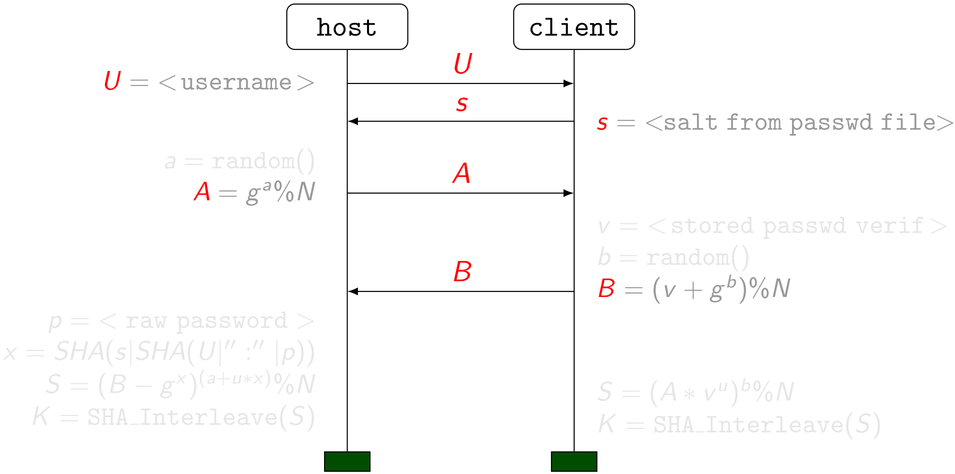
MSC (as for automata, but much more detailed):



RFC 2945 (SRP Authentication and Key Exchange System)

The protocol approach: global specifications

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



Says how messages are generated

RFC 2945 (SRP Authentication and Key Exchange System)

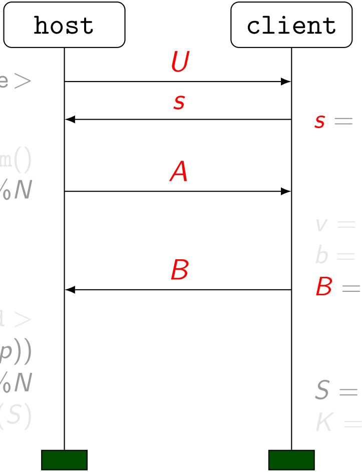
The protocol approach: global specifications

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

$U = <\texttt{username}>$

$a = \texttt{random}()$
$A = g^{a}\% N$

$p = <\texttt{raw password}>$
$x = SHA(s|SHA(U|'':''|p))$
$S = (B - g^{x})^{(a+u*x)}\% N$
$K = \texttt{SHA\_Interleave}(S)$



$s = <\texttt{salt from passwd file}>$

$v = <\texttt{stored passwd verif}>$
$b = \texttt{random}()$
$B = (v + g^{b})\% N$

$S = (A * v^{u})^{b}\% N$
$K = \texttt{SHA\_Interleave}(S)$

Says how messages are generated

Says how messages are used

RFC 2945 (SRP Authentication and Key Exchange System)

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

Higher-order sessions [Honda, Yoshida, Carbone 2008]

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.

$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*
$\texttt{Alice} \xrightarrow{b\langle c{:}!\mathsf{Int}\rangle} \texttt{Bob};$	*delegate the sending of an int on c*
$\texttt{Bob} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*
$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*

Higher order sessions [Honda, Yoshida, Carbone 2008]

Channels names are specified

Sp… s and pass them around

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

$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*
$\texttt{Alice} \xrightarrow{b\langle c{:}!\mathsf{Int}\rangle} \texttt{Bob};$	*delegate the sending of an int on c*
$\texttt{Bob} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*
$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*

Higher order sessions [Honda, Yoshida, Carbone 2008]

Channels names are specified

Channels are passed around

Sp… them around

Two chan…s: … ice and Bob, and c by Alice and Charlie.

$$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie}; \qquad \textit{send an integer on channel } c$$

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

$$\texttt{Bob} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie}; \qquad \textit{send an integer on channel } c$$

$$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie}; \qquad \textit{send an integer on channel } c$$

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.

$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*
$\texttt{Alice} \xrightarrow{b\langle c{:}!\mathsf{Int}\rangle} \texttt{Bob};$	*delegate the sending of an int on c*
$\texttt{Bob} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*
$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*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 ()
```

Charlie

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

Specify channels and pass them around

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

$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*
$\texttt{Alice} \xrightarrow{b\langle c{:}!\mathsf{Int}\rangle} \texttt{Bob};$	*delegate the sending of an int on c*
$\texttt{Bob} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*send an integer on channel c*
$\texttt{Alice} \xrightarrow{c\langle \mathsf{Int}\rangle} \texttt{Charlie};$	*integer on channel c*

Charlie is not aware that this communication is with Bob

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 ()
```

Charlie
```
receive $x on c in
receive $y on c in
receive $z on c in
$x+$y+$z
```

Dynamic multirole sessions [Deniélou & Yoshida 2011]

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

$$\forall x : \texttt{buyer}. \quad (\texttt{seller} \xrightarrow{descr} x \wedge \texttt{seller} \xrightarrow{price} x);$$
$$(x \xrightarrow{accept} \texttt{seller} \vee x \xrightarrow{quit} \texttt{seller})$$

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

Dynamic multirole sessions [Deniélou & Yoshida 2011]

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

$$\forall x : \texttt{buyer}. \quad (\texttt{seller} \xrightarrow{\textit{descr}} x \wedge \texttt{seller} \xrightarrow{\textit{price}} x); \\ (x \xrightarrow{\textit{accept}} \texttt{seller} \vee x \xrightarrow{\textit{quit}} \texttt{seller})$$

$\texttt{buyer}$ is a *role*: can be played by different participants (ranged over by x)

$\boxed{\texttt{seller}}$

$$\forall x : \texttt{buyer}.\ x\,\mathbf{!}\,\textit{descr}. \\ x\,\mathbf{!}\,\textit{price}. \\ (x\mathbf{?}\textit{accept} + x\mathbf{?}\textit{quit})$$

$\boxed{\texttt{buyer}}$

$$\texttt{seller}\mathbf{?}\textit{descr}. \\ \texttt{seller}\mathbf{?}\textit{price}. \\ (\texttt{seller}\,\mathbf{!}\,\textit{accept} \oplus \texttt{seller}\,\mathbf{!}\,\textit{quit})$$

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

$$\forall x : \texttt{buyer}. \quad (\texttt{seller} \xrightarrow{descr} x \wedge \texttt{seller} \xrightarrow{price} x);$$
$$(x \xrightarrow{accept} \texttt{seller} \vee x \xrightarrow{quit} \texttt{seller})$$

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

$\boxed{\texttt{seller}}$

$$\forall x : \texttt{buyer}.\ x\,!\,descr.\ x\,!\,price.\ (x\,?\,accept + x\,?\,quit)$$

$\boxed{\texttt{buyer}}$

$$\texttt{seller}\,?\,descr.\ \texttt{seller}\,?\,price.\ (\texttt{seller}\,!\,accept \oplus \texttt{seller}\,!\,quit)$$

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:

$$\forall x : \texttt{buyer}. \quad (\texttt{seller} \xrightarrow{descr} x \wedge \texttt{seller} \xrightarrow{price} x); \\ (x \xrightarrow{accept} \texttt{seller} \vee x \xrightarrow{quit} \texttt{seller})$$

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

$\boxed{\texttt{seller}}$

$$\forall x : \texttt{buyer}.\ x\,!\,descr. \\ x\,!\,price. \\ (x\,?\,accept + x\,?\,quit)$$

$\boxed{\texttt{buyer}}$

$$\texttt{seller}\,?\,descr. \\ \texttt{seller}\,?\,price. \\ (\texttt{seller}\,!\,accept \oplus \texttt{seller}\,!\,quit)$$

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.