A Theory of Contracts for Web Services

Giuseppe Castagna, Nils Gesbert, Luca Padovani

Université Paris 7, University of Glasgow, Università di Urbino

ParSec meeting 11/12/2007
Web services in a nutshell

- distributed processes

- communicating through standard Web protocols (TCP, HTTP, SOAP)

- exchanging data in platform-neutral format (XML)

- dynamically linked

- with machine-understandable descriptions
Web services in a nutshell

- distributed processes

- communicating through standard Web protocols (TCP, HTTP, SOAP)

- exchanging data in platform-neutral format (XML)

- dynamically linked

- with machine-understandable descriptions
Technologies for Web services

Interface descriptions

- WSDL 1.1 (W3C note, 2001)
- WSDL 2.0 (W3C recommendation, 2007)

Behavioural descriptions

- WSCL 1.0 (W3C note, 2002)
- WSCI 1.0 (W3C note, 2002)
- WS-BPEL 2.0 (OASIS standard, 2007)

“Enabling users to describe business process activities as Web services and define how they can be connected to accomplish specific tasks”

Registries

- UDDI 3.0.2 (OASIS standard, 2004)

“Defining a standard method for enterprises to dynamically discover and invoke Web services”
Technologies for Web services

Interface descriptions
- WSDL 1.1 (W3C note, 2001)
- WSDL 2.0 (W3C recommendation, 2007)

Behavioural descriptions
- WSCL 1.0 (W3C note, 2002)
- WSCI 1.0 (W3C note, 2002)
- WS-BPEL 2.0 (OASIS standard, 2007)

“Enabling users to describe business process activities as Web services and define how they can be connected to accomplish specific tasks”

Registries
- UDDI 3.0.2 (OASIS standard, 2004)

“Defining a standard method for enterprises to dynamically discover and invoke Web services”
Technologies for Web services

Interface descriptions
- WSDL 1.1 (W3C note, 2001)
- WSDL 2.0 (W3C recommendation, 2007)

Behavioural descriptions
- WSCL 1.0 (W3C note, 2002)
- WSCI 1.0 (W3C note, 2002)
- WS–BPEL 2.0 (OASIS standard, 2007)

“Enabling users to describe business process activities as Web services and define how they can be connected to accomplish specific tasks”

Registries
- UDDI 3.0.2 (OASIS standard, 2004)

“Defining a standard method for enterprises to dynamically discover and invoke Web services”
Discovering Web services

Search key

- name
- industrial classification
- location
  ...
- behavioural type!

Problem
We need a *semantic* notion of behavioural equivalence which

- preserves client satisfaction
- is abstract (based on the described, observable behaviour)

Plan
Synthesise *contracts* from Web service descriptions, give contracts a formal semantics, use contracts for searching (and possibly more... )
Summary

In this talk...

1. understand what contracts look like
2. define client satisfaction (compliance)
3. define contract equivalence (subcontract)
4. relate compliance and subcontract (filters)
5. apply to languages used to implement client/services
6. apply to service discovery
What is a contract?

1. Describes sequences of INPUT/OUTPUT actions

   Query.Catalog

2. Login.(ValidLogin.... ⊕ InvalidLogin....)

3. Query.Catalog.(Logout.... + Purchase....)
What is a contract?

1. Describes sequences of INPUT/OUTPUT actions

   Query.Catalog

2. Login.(ValidLogin. ... ⊕ InvalidLogin. ...)

3. Query.Catalog.(Logout. ... + Purchase. ...)

---

Castagna, Gesbert, and Padovani

A Theory of Contracts for Web Services

6/23
What is a contract?

1. Describes sequences of INPUT/OUTPUT actions

   - Query.Catalog

2. Login.(ValidLogin. ... ⊕ InvalidLogin. ...)

3. Query.Catalog.(Logout. ... + Purchase. ...)

---

Castagna, Gesbert, and Padovani  A Theory of Contracts for Web Services 6/23
What is a contract?

1. Describes sequences of INPUT/OUTPUT actions

   Query.Catalog

2. Login.(ValidLogin.... ⊕ InvalidLogin....)

3. Query.Catalog.(Logout.... + Purchase....)
What is a contract?

1. Describes sequences of INPUT/OUTPUT actions

   \textbf{Query.Catalog}

2. Describes possible internal choices

   \textbf{Login.(ValidLogin.\ldots \oplus InvalidLogin.\ldots)}

3. \textbf{Query.Catalog.(Logout.\ldots + Purchase.\ldots)}
What is a contract?

1. Describes sequences of INPUT/OUTPUT actions
   \underline{Query.Catalog}

2. Describes possible internal choices
   \underline{Login.(ValidLogin.\textellipsis \oplus \textit{InvalidLogin.\textellipsis)}}

3. \underline{Query.Catalog.\textit{(Logout.\textellipsis + Purchase.\textellipsis)}}
**What is a contract?**

1. **Describes sequences of INPUT/OUTPUT actions**
   
   Query.\text{Catalog}

2. **Describes possible internal choices**
   
   Login.(ValidLogin... \oplus InvalidLogin...)

3. **Available external choices**
   
   Query.\text{Catalog}.(Logout.... + Purchase....)
What is a contract?

1. Describes sequences of INPUT/OUTPUT actions
   
   Query.\text{Catalog}

2. Describes possible internal choices
   
   Login.(\overline{\text{ValidLogin}}. ... \oplus \text{InvalidLogin}. ...)

3. Describes available external choices
   
   Query.\text{Catalog}.(\text{Logout}. .... + \text{Purchase}. ....)
What is a contract?

1 Describes sequences of INPUT/OUTPUT actions
   \textit{Query.Catalog}

2 Describes possible internal choices
   \textit{Login.(ValidLogin.\ldots \oplus InvalidLogin.\ldots)}

3 Describes available external choices
   \textit{Query.Catalog.(Logout.\ldots + Purchase.\ldots)}
What is a contract?

1. Describes sequences of INPUT/OUTPUT actions
   
   **Query.**Catalog

2. Describes possible internal choices
   
   **Login.**(ValidLogin... Φ InvalidLogin...)

3. Describes available external choices
   
   Query.**Catalog.**(Logout.... + Purchase....)
What is a contract?

1. Describes sequences of INPUT/OUTPUT actions
   
   Query.\overline{Catalog}

2. Describes possible internal choices
   
   Login.(\overline{ValidLogin} \ldots \oplus \overline{InvalidLogin} \ldots)

3. Describes available external choices
   
   Query.\overline{Catalog}.(Logout.\ldots + Purchase.\ldots)

Note that the contract is recursive.
What is a contract?

\[
\text{rec } x. \text{Login} . (\text{InvalidLogin} . x \oplus \text{ValidLogin}. \text{rec } y. \text{Query}.\text{Catalog}. (y \oplus \text{Logout} \oplus \text{rec } z. \text{Purchase}. (\text{Accepted} \oplus \text{InvalidPayment}. (z \oplus \text{Logout}) \oplus \text{OutOfStock}. (y \oplus \text{Logout}))))
\]

Note that the contract is recursive
What is a contract?

We do not consider recursion in this talk.

rec x. Login.(InvalidLogin.x ⊕ ValidLogin.rec y. Query.Catalog.(y + Logout + rec z. Purchase. Accepted ⊕ InvalidPayment.(z + Logout) ⊕ OutOfStock.(y + Logout)))
A formal contract language

contracts \( \sigma ::= \)
\[
\begin{align*}
0 & \quad (\text{void}) \\
\alpha . \sigma & \quad (\text{action prefix}) \\
\sigma + \sigma & \quad (\text{external choice}) \\
\sigma \oplus \sigma & \quad (\text{internal choice})
\end{align*}
\]

actions \( \alpha ::= \)
\[
\begin{align*}
a & \quad (\text{receive}) \\
\bar{a} & \quad (\text{send})
\end{align*}
\]
Names represent types, operations, \ldots
A formal contract language

\[
\text{contracts} \quad \sigma ::= \\
0 \quad (\text{void}) \\
\alpha.\sigma \quad (\text{action prefix}) \\
\sigma + \sigma \quad (\text{external choice}) \\
\sigma \oplus \sigma \quad (\text{internal choice}) \\
\]

\[
\text{actions} \quad \alpha ::= \\
a \quad (\text{receive}) \\
\overline{a} \quad (\text{send}) \\
\]

Names represent types, operations, \ldots

Two questions:

1. When does a client fit a server of given contract? \quad (compliance)
2. When is a contract more general than another? \quad (subcontracting)
A formal contract language

contracts  \( \sigma \) ::= 

0 \hspace{1cm} (void)
\( \alpha \cdot \sigma \) \hspace{1cm} (action prefix)
\( \sigma + \sigma \) \hspace{1cm} (external choice)
\( \sigma \oplus \sigma \) \hspace{1cm} (internal choice)

actions  \( \alpha \) ::= 
a \hspace{1cm} (receive)
\( \overline{a} \) \hspace{1cm} (send)

Names represent types, operations, ...
A formal contract language

\[
\text{contracts} \quad \sigma ::= \\
\quad 0 \quad (\text{void}) \\
\quad \alpha.\sigma \quad (\text{action prefix}) \\
\quad \sigma + \sigma \quad (\text{external choice}) \\
\quad \sigma \oplus \sigma \quad (\text{internal choice})
\]

\[
\text{actions} \quad \alpha ::= \\
\quad a \quad (\text{receive}) \\
\quad \overline{a} \quad (\text{send})
\]

Two questions:

1. When does a client fit a server of given contract? (compliance)
   When it successfully achieves every possible interaction with it

2. When is a contract more general than another? (subcontracting)
   When all the clients of the other comply with it
The contract of a process describes

1. WHICH actions the process offers

\[
\begin{align*}
\sigma_1 \xrightarrow{\alpha} \sigma'_1 & \quad \sigma_2 \xrightarrow{\alpha} \sigma'_2 \quad \sigma_1 + \sigma_2 \xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2 \\
\sigma_1 \xrightarrow{\alpha} \sigma'_1 & \quad \sigma_2 \xrightarrow{\alpha} \sigma'_2 \quad \sigma_1 \oplus \sigma_2 \xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2
\end{align*}
\]

2. HOW actions are offered

\[
\begin{align*}
\emptyset \Downarrow \emptyset \\
\alpha.\sigma \Downarrow \{\alpha\} \\
(\sigma + \sigma') \Downarrow R \cup R' & \quad \text{if } \sigma \Downarrow R \text{ and } \sigma' \Downarrow R' \\
(\sigma \oplus \sigma') \Downarrow R & \quad \text{if either } \sigma \Downarrow R \text{ or } \sigma' \Downarrow R
\end{align*}
\]
### Semantics

The contract of a process describes

1. **WHICH actions the process offers**

   \[ \sigma_1 \xrightarrow{\alpha} \sigma'_1 \quad \sigma_2 \xrightarrow{\alpha} \sigma'_2 \]
   \[ \sigma_1 + \sigma_2 \xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2 \]

   \[ \alpha \cdot \sigma \xrightarrow{\alpha} \sigma \]

   \[ \sigma_1 \xrightarrow{\alpha} \sigma'_1 \quad \sigma_2 \xrightarrow{\alpha} \sigma'_2 \]
   \[ \sigma_1 \oplus \sigma_2 \xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2 \]

2. **HOW actions are offered**

   \[ 0 \downarrow \emptyset \]
   \[ \alpha \cdot \sigma \downarrow \{ \alpha \} \]
   \[ (\sigma + \sigma') \downarrow R \cup R' \quad \text{if } \sigma \downarrow R \text{ and } \sigma' \downarrow R' \]
   \[ (\sigma \oplus \sigma') \downarrow R \quad \text{if either } \sigma \downarrow R \text{ or } \sigma' \downarrow R \]

---

Castagna, Gesbert, and Padovani

A Theory of Contracts for Web Services
Semantics

The contract of a process describes

1. WHICH actions the process offers

\[ \alpha \cdot \sigma \xrightarrow{\alpha} \sigma \]

\[ \sigma_1 \xrightarrow{\alpha} \sigma'_1 \quad \sigma_2 \xrightarrow{\alpha} \sigma'_2 \]

\[ \sigma_1 + \sigma_2 \xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2 \]

\[ \sigma_1 \oplus \sigma_2 \xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2 \]

(\(\sigma \xrightarrow{\alpha}: \text{“}\sigma\text{ can emit }\alpha\text{”}\))

\[ \sigma_1 \xrightarrow{\alpha} \sigma'_1 \quad \sigma_2 \xrightarrow{\check{\alpha}} \]

\[ \sigma_1 + \sigma_2 \xrightarrow{\alpha} \sigma'_1 \]

\[ \sigma_1 \xrightarrow{\alpha} \sigma'_1 \quad \sigma_2 \xrightarrow{\check{\alpha}} \]

\[ \sigma_1 \oplus \sigma_2 \xrightarrow{\alpha} \sigma'_1 \]

2. HOW actions are offered

\[ \emptyset \Downarrow \emptyset \]

\[ \alpha \cdot \sigma \Downarrow \{ \alpha \} \]

\[ (\sigma + \sigma') \Downarrow R \cup R' \quad \text{if } \sigma \Downarrow R \text{ and } \sigma' \Downarrow R' \]

\[ (\sigma \oplus \sigma') \Downarrow R \quad \text{if either } \sigma \Downarrow R \text{ or } \sigma' \Downarrow R \]
Semantics

The contract of a process describes

1. **WHICH actions the process offers**

   \[ \alpha \cdot \sigma \xrightarrow{\alpha} \sigma \]

   \[
   \begin{align*}
   \sigma_1 &\xrightarrow{\alpha} \sigma'_1 \\
   \sigma_2 &\xrightarrow{\alpha} \sigma'_2 \\
   \sigma_1 + \sigma_2 &\xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2 \\
   \sigma_1 \oplus \sigma_2 &\xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2
   \end{align*}
   \]

   \[ (\sigma \xrightarrow{\alpha}: \text{“}\sigma \text{ can emit } \alpha\text{”} ) \]

   \[
   \begin{align*}
   \sigma_1 &\xrightarrow{\alpha} \sigma'_1 \\
   \sigma_2 &\not\xrightarrow{\alpha} \\
   \sigma_1 + \sigma_2 &\xrightarrow{\alpha} \sigma'_1 \\
   \sigma_1 \oplus \sigma_2 &\xrightarrow{\alpha} \sigma'_1
   \end{align*}
   \]

2. **HOW actions are offered**

   \[
   \begin{align*}
   \emptyset &\downarrow \emptyset \\
   \alpha \cdot \sigma &\downarrow \{\alpha\} \\
   (\sigma + \sigma') &\downarrow R \cup R' & \text{if } \sigma \downarrow R \text{ and } \sigma' \downarrow R' \\
   (\sigma \oplus \sigma') &\downarrow R & \text{if either } \sigma \downarrow R \text{ or } \sigma' \downarrow R
   \end{align*}
   \]
Semantics

The contract of a process describes

1. WHICH actions the process offers

\[ \sigma_1 \xrightarrow{\alpha} \sigma'_1 \quad \sigma_2 \xrightarrow{\alpha} \sigma'_2 \]
\[ \sigma_1 + \sigma_2 \xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2 \]
\[ \sigma_1 \xrightarrow{\alpha} \sigma'_1 \quad \sigma_2 \xrightarrow{\alpha} \sigma'_2 \]
\[ \sigma_1 \oplus \sigma_2 \xrightarrow{\alpha} \sigma'_1 \oplus \sigma'_2 \]

\[ \alpha \cdot \sigma \xrightarrow{\alpha} \sigma \]

(\( \sigma \xrightarrow{\alpha} \): “\( \sigma \) can emit \( \alpha \)”)

2. HOW actions are offered

\[ 0 \downarrow \emptyset \]
\[ \alpha \cdot \sigma \downarrow \{ \alpha \} \]
\[ (\sigma + \sigma') \downarrow \mathcal{R} \cup \mathcal{R}' \quad \text{if } \sigma \downarrow \mathcal{R} \text{ and } \sigma' \downarrow \mathcal{R}' \]
\[ (\sigma \oplus \sigma') \downarrow \mathcal{R} \quad \text{if either } \sigma \downarrow \mathcal{R} \text{ or } \sigma' \downarrow \mathcal{R} \]
Semantics

For instance

\[ a \oplus b \xrightarrow{a} 0 \]
\[ a \oplus b \xrightarrow{b} 0 \]

but

\[ a \oplus b \downarrow \{a\} \]
\[ a \oplus b \downarrow \{b\} \]
\[ a + b \xrightarrow{a} 0 \]
\[ a + b \xrightarrow{b} 0 \]
\[ a + b \downarrow \{a, b\} \]
Relating clients and services: **compliance**

A client \( \rho \) complies with a service \( \sigma \) \((\rho \vdash \sigma)\) if it successfully achieves every possible interaction with the service

- \( a.\lozenge + b.\lozenge \vdash \bar{a} \oplus \bar{b} \)
- \( a.\lozenge + b.\lozenge \vdash \bar{a} \)
- \( a.\lozenge \oplus b.\lozenge \vdash \bar{a}.c + \bar{b}.d \)
- \( a.\lozenge \oplus b.\lozenge \not\vdash \bar{a} \oplus \bar{b} \)

(\( \lozenge \) indicates client’s satisfaction).
A client $\rho$ complies with a service $\sigma$ ($\rho \vdash \sigma$) if it successfully achieves every possible interaction with the service:

- $a.\bigcirc + b.\bigcirc \vdash \overline{a} \oplus \overline{b}$
- $a.\bigcirc + b.\bigcirc \vdash \overline{a}$
- $a.\bigcirc \oplus b.\bigcirc \vdash \overline{a}.c + \overline{b}.d$
- $a.\bigcirc \oplus b.\bigcirc \not\vdash \overline{a} \oplus \overline{b}$

($\bigcirc$ indicates client’s satisfaction).
A client $\rho$ complies with a service $\sigma$ ($\rho \vdash \sigma$) if it successfully achieves every possible interaction with the service:

- $a.\triangleright + b.\triangleright \vdash \bar{a} \oplus \bar{b}$
- $a.\triangleright + b.\triangleright \vdash \bar{a}$
- $a.\triangleright \oplus b.\triangleright \vdash \bar{a}.c + \bar{b}.d$
- $a.\triangleright \oplus b.\triangleright \not\vdash \bar{a} \oplus \bar{b}$

($\triangleright$ indicates client’s satisfaction).

Formally $\rho \vdash \sigma$ iff

1. $\rho \Downarrow R$, $\sigma \Downarrow S$, and $\overline{R} \cap S = \emptyset$ imply $\triangleright \in R$
2. $\rho \xrightarrow{\bar{a}} \rho'$ and $\sigma \xrightarrow{\alpha} \sigma'$ imply $\rho' \vdash \sigma'$
Relating different services: subcontract

Intuition

A client that works with a server $\sigma$ will also work with a server $\tau$ that “does more”: $\sigma \preceq \tau$

1. Is more deterministic:
   \[
   \overline{a} \oplus b.c \preceq \overline{a}
   \]
   \[
   \text{InvalidLogin} \oplus \text{ValidLogin} \preceq \text{ValidLogin}
   \]

2. Offers more choices:
   \[
   \overline{a} \preceq \overline{a} + b.d
   \]
   \[
   \text{Logout} + \text{Purchase} \preceq \text{Logout} + \text{Purchase} + \text{SaveForLater} \quad \text{[width extension]}
   \]

3. Offers longer interaction patterns:
   \[
   \overline{a} \preceq \overline{a}.b.d
   \]
   \[
   \text{Purchase.} \overline{\text{Accepted}} \preceq \text{Purchase.} \overline{\text{Accepted}}.\overline{\text{Invoice}} \quad \text{[depth extension]}
   \]
Relating different services: **subcontract**

**Intuition**

A client that works with a server $\sigma$ will also work with a server $\tau$ that “does more”:

1. **Is more deterministic:**
   
   \[
   \overline{a} \oplus \overline{b}.c \preceq \overline{a}
   \]

   \[
   \text{InvalidLogin} \oplus \text{ValidLogin} \preceq \text{ValidLogin}
   \]

2. **Offers more choices:**
   
   \[
   \overline{a} \preceq \overline{a} + \overline{b}.d
   \]

   \[
   \text{Logout} + \text{Purchase} \preceq \text{Logout} + \text{Purchase} + \text{SaveForLater}
   \]

3. **Offers longer interaction patterns:**
   
   \[
   \overline{a} \preceq \overline{a}.\overline{b}.d
   \]

   \[
   \text{Purchase.A} \preceq \text{Purchase.A}.\text{Accepted}.\text{Invoice}
   \]
Relating different services: subcontract

**Intuition**

A client that works with a server $\sigma$ will also work with a server $\tau$ that “does more”: $\sigma \preceq \tau$

1. **Is more deterministic:**
   \[
   \overline{a} \oplus \overline{b.c} \preceq \overline{a}
   \]
   \[
   \text{InvalidLogin} \oplus \text{ValidLogin} \preceq \text{ValidLogin}
   \]

2. **Offers more choices:**
   \[
   \overline{a} \preceq \overline{a + b.d}
   \]
   \[
   \text{Logout} + \text{Purchase} \preceq \text{Logout} + \text{Purchase} + \text{SaveForLater}
   \]

3. **Offers longer interaction patterns:**
   \[
   \overline{a} \preceq \overline{a.b.d}
   \]
   \[
   \text{Purchase.A} + \text{Accepted} \preceq \text{Purchase.A.A.} + \text{Invoice}
   \]
Relating different services: subcontract

Intuition

A client that works with a server $\sigma$ will also work with a server $\tau$ that “does more”: $\sigma \preceq \tau$

1. **Is more deterministic:**
   \[
   \bar{a} \oplus \bar{b}.c \preceq \bar{a}
   \]
   \[
   \text{InvalidLogin} \oplus \text{ValidLogin} \preceq \text{ValidLogin}
   \]

2. **Offers more choices:**
   \[
   \bar{a} \preceq \bar{a} + \bar{b}.d
   \]
   \[
   \text{Logout+Purchase} \preceq \text{Logout+Purchase+SaveForLater}
   \]

3. **Offers longer interaction patterns:**
   \[
   \bar{a} \preceq \bar{a}.\bar{b}.d
   \]
   \[
   \text{Purchase.Accepted} \preceq \text{Purchase.Accepted.Invoice}
   \]
Relating different services: **subcontract**

**Intuition**

A client that works with a server $\sigma$ will also work with a server $\tau$ that “does more”: $\sigma \preceq \tau$

1. **Is more deterministic:**

   \[ \overline{a} \oplus \overline{b.c} \preceq \overline{a} \]

   \[
   \text{InvalidLogin} \oplus \text{ValidLogin} \preceq \text{ValidLogin}
   \]

2. **Offers more choices:**

   \[ \overline{a} \preceq \overline{a + b.d} \]

   \[
   \text{Logout} + \text{Purchase} \preceq \text{Logout} + \text{Purchase} + \text{SaveForLater} \quad [\text{width extension}]
   \]

3. **Offers longer interaction patterns:**

   \[ \overline{a} \preceq \overline{a.b.d} \]

   \[
   \text{Purchase}.\text{Accepted} \preceq \text{Purchase}.\text{Accepted}.\text{Invoice} \quad [\text{depth extension}]
   \]
Relating different services: **subcontract**

**Intuition**

A client that works with a server $\sigma$ will also work with a server $\tau$ that “does more”: $\sigma \preceq \tau$

1. **Is more deterministic:**

   \[ \overline{a} \oplus \overline{b.c} \preceq \overline{a} \]

   $\text{InvalidLogin} \oplus \text{ValidLogin} \preceq \text{ValidLogin}$

2. **Offers more choices:**

   \[ \overline{a} \preceq \overline{a + b.d} \]

   $\text{Logout} + \text{Purchase} \preceq \text{Logout} + \text{Purchase} + \text{SaveForLater}$  [width extension]

3. **Offers longer interaction patterns:**

   \[ \overline{a} \preceq \overline{a \cdot b.d} \]

   $\text{Purchase.Accepted} \preceq \text{Purchase.Accepted.Invoice}$  [depth extension]
Relating different services: subcontract

Intuition
A client that works with a server $\sigma$ will also work with a server $\tau$ that “does more”: $\sigma \preceq \tau$

1. Is more deterministic:
   $\overline{a} \oplus \overline{b}.c \preceq \overline{a}$
   $\text{InvalidLogin} \oplus \text{ValidLogin} \preceq \text{ValidLogin}$

2. Offers more choices:
   $\overline{a} \preceq \overline{a} + \overline{b}.d$
   $\text{Logout} + \text{Purchase} \preceq \text{Logout} + \text{Purchase} + \text{SaveForLater}$ [width extension]

3. Offers longer interaction patterns:
   $\overline{a} \preceq \overline{a}.\overline{b}.d$
   $\text{Purchase.Accepted} \preceq \text{Purchase.Accepted.Invoice}$ [depth extension]
Relating different services: subcontract

**Intuition**

A client that works with a server $\sigma$ will also work with a server $\tau$ that “does more”: $\sigma \preceq \tau$

1. **Is more deterministic:**

$$\overline{a} \oplus \overline{b}.c \preceq \overline{a}$$

$\text{InvalidLogin} \oplus \text{ValidLogin} \preceq \text{ValidLogin}$

2. **Offers more choices:**

$$\overline{a} \preceq \overline{a} + \overline{b}.d$$

Logout+Purchase $\preceq$ Logout+Purchase+SaveForLater [width extension]

3. **Offers longer interaction patterns:**

$$\overline{a} \preceq \overline{a}.\overline{b}.d$$

Purchase.\text{Accepted} $\preceq$ Purchase.\text{Accepted}.\text{Invoice} [depth extension]

Is “$\preceq$” an (inverse) subtyping relation? Apparent mismatch in 1 and 2
Compliance and Subcontract mismatch

- Note:

\[ \overline{a} \oplus \overline{b}.c \quad \preceq \quad a \quad \preceq \quad \overline{a} + \overline{b}.d \]

- but for a client \( a \otimes + b.\overline{c}.\otimes \):

\[ a \otimes + b.\overline{c}.\otimes \not\preceq \overline{a} \oplus \overline{b}.c \quad a \otimes + b.\overline{c}.\otimes \not\preceq \overline{a} + \overline{b}.d \]

- Can we replace a server \( \overline{a} \oplus \overline{b}.c \) for a \( \overline{a} + \overline{b}.d \) one (i.e. \( \overline{a} \oplus \overline{b}.c :\rightarrow \overline{a} + \overline{b}.d \))?

- YES if “\( :\rightarrow \)” uses explicit coercions (rather than implicit ones)

Filter-out foreign actions
(e.g. \( \overline{b} \) and \( d \) for \( \otimes \))
Compliance and Subcontract mismatch

- Note:

\[ \bar{a} \oplus \bar{b}.c \prec \bar{a} \preceq \bar{a} + \bar{b}.d \]

- but for a client \( a \otimes b \bar{c} \cdot \oplus \):

\[ a \otimes b \bar{c} \cdot \oplus \rightarrow \bar{a} \oplus \bar{b}.c \quad a \otimes b \bar{c} \cdot \oplus \not\rightarrow \bar{a} + \bar{b}.d \]

- Can we replace a server \( \bar{a} \oplus \bar{b}.c \) for a \( \bar{a} + \bar{b}.d \) one (i.e. \( \bar{a} \oplus \bar{b}.c :> \bar{a} + \bar{b}.d \))? 

- YES if “:>” uses explicit coercions (rather than implicit ones)

Filter-out foreign actions (e.g. \( \bar{b} \) and \( d \) for \( \otimes \))
Compliance and Subcontract mismatch

- Note:
  \[
  \overline{a} \oplus \overline{b}.c \preceq a \preceq \overline{a} + \overline{b}.d
  \]

  but for a client \(a \quad \forall + b.\overline{c}.\forall\):
  \[
  a \quad \forall + b.\overline{c}.\forall \manyrightarrow \overline{a} \oplus \overline{b}.c \quad a \quad \forall + b.\overline{c}.\forall \manyrightarrow \overline{a} + \overline{b}.d
  \]

- Can we replace a server \(\overline{a} \oplus \overline{b}.c\) for a \(\overline{a} + \overline{b}.d\) one
  (i.e. \(\overline{a} \oplus \overline{b}.c :> \overline{a} + \overline{b}.d\))?

  YES if “:>” uses explicit coercions (rather than implicit ones)

Filter-out foreign actions
  (e.g. \(\overline{b}\) and \(d\) for \(\exists\))
Compliance and Subcontract mismatch

- Note:

\[ \overline{a} \oplus \overline{b}.c \quad \preceq \quad \overline{a} \quad \preceq \quad \overline{a} + \overline{b}.d \]

- but for a client \( a.\oplus + b.\overline{c}.\oplus \):

\[ a.\oplus + b.\overline{c}.\oplus \nRightarrow \overline{a} \oplus \overline{b}.c \quad a.\oplus + b.\overline{c}.\oplus \nRightarrow \overline{a} + \overline{b}.d \]

- Can we replace a server \( \overline{a} \oplus \overline{b}.c \) for a \( \overline{a} + \overline{b}.d \) one (i.e. \( \overline{a} \oplus \overline{b}.c \nRightarrow \overline{a} + \overline{b}.d \))?  

  - YES if “\( \nRightarrow \)” uses explicit coercions (rather than implicit ones)

Filter-out foreign actions  
(e.g. \( \overline{b} \) and \( d \) for \( \bullet \))
Compliance and Subcontract mismatch

- Note:

\[ \overline{a} \oplus \overline{b}.c \triangleleft \overline{a} \triangleleft \overline{a} + \overline{b}.d \]

- but for a client \( a.\Diamond + b.\overline{c}.\Diamond \):

\[ a.\Diamond + b.\overline{c}.\Diamond \not\sqsubseteq \overline{a} \oplus \overline{b}.c \quad a.\Diamond + b.\overline{c}.\Diamond \not\sqcup \overline{a} + \overline{b}.d \]

- Can we replace a server \( \overline{a} \oplus \overline{b}.c \) for a \( \overline{a} + \overline{b}.d \) one
(i.e. \( \overline{a} \oplus \overline{b}.c :> \overline{a} + \overline{b}.d \))?

- YES if “:>” uses explicit coercions (rather than implicit ones)

Filter-out foreign actions
(e.g. \( \overline{b} \) and \( d \) for 2)
Gluing compliance and subcontracting: Filters

\[ \text{filters} \quad f \ ::= \ \bigsqcup_{\alpha \in \Lambda} \alpha \cdot f_{\alpha} \]

Transition relation of filters

\[ \bigsqcup_{\alpha \in \Lambda} \alpha \cdot f_{\alpha} \xrightarrow{\beta} f_{\beta} \quad \text{if } \beta \in \Lambda \]

Contract coercion through a filter

\[
\begin{align*}
    f(0) & = 0 \\
    f(\alpha \cdot \sigma) & = 0 & \text{if } f \xrightarrow{\alpha} \\
    f(\alpha \cdot \sigma) & = \alpha \cdot f'(\sigma) & \text{if } f \xrightarrow{\alpha} f' \\
    f(\sigma_1 + \sigma_2) & = f(\sigma_1) + f(\sigma_2) \\
    f(\sigma_1 \oplus \sigma_2) & = f(\sigma_1) \oplus f(\sigma_2)
\end{align*}
\]

Property

\[ \sigma \preceq \tau \land \rho \vdash \sigma \iff \sigma \vdash f(\tau) \quad \text{for some filter } f \]
Gluing compliance and subcontracting: Filters

filters  \[ f ::= \bigsqcup_{\alpha \in \Lambda} \alpha \cdot f_{\alpha} \]

Transition relation of filters

\[ \bigsqcup_{\alpha \in \Lambda} \alpha \cdot f_{\alpha} \xrightarrow{\beta} f_{\beta} \quad \text{if } \beta \in \Lambda \]

Contract coercion through a filter

\[
\begin{align*}
    f(0) &= 0 \\
    f(\alpha \cdot \sigma) &= 0 & \text{if } f \xrightarrow{\alpha} \\
    f(\alpha \cdot \sigma) &= \alpha \cdot f'(\sigma) & \text{if } f \xrightarrow{\alpha} f'
\end{align*}
\]
\[
\begin{align*}
    f(\sigma_1 + \sigma_2) &= f(\sigma_1) + f(\sigma_2) \\
    f(\sigma_1 \oplus \sigma_2) &= f(\sigma_1) \oplus f(\sigma_2)
\end{align*}
\]

Property

\[ \sigma \preceq \tau \land \rho \vdash \sigma \iff \sigma \vdash f(\tau) \text{ for some filter } f \]
Gluing compliance and subcontracting: Filters

\[ f ::= \bigcap_{\alpha \in \Lambda} \alpha.f_{\alpha} \]

Transition relation of filters

\[ \bigcap_{\alpha \in \Lambda} \alpha.f_{\alpha} \xrightarrow{\beta} f_{\beta} \quad \text{if } \beta \in \Lambda \]

Contract coercion through a filter

\[
\begin{align*}
  f(0) &= 0 \\
  f(\alpha \cdot \sigma) &= 0 & \text{if } f \xrightarrow{\alpha} \\
  f(\alpha \cdot \sigma) &= \alpha \cdot f'(\sigma) & \text{if } f \xrightarrow{\alpha} f' \\
  f(\sigma_1 + \sigma_2) &= f(\sigma_1) + f(\sigma_2) \\
  f(\sigma_1 \oplus \sigma_2) &= f(\sigma_1) \oplus f(\sigma_2)
\end{align*}
\]

Property

\[ \sigma \preceq \tau \quad \land \quad \rho \vdash \sigma \quad \iff \quad \sigma \vdash f(\tau) \quad \text{for some filter } f \]
Gluing compliance and subcontracting: Filters

filters

\[ f ::= \bigwedge_{\alpha \in \Lambda} \alpha.f_{\alpha} \]

Transition relation of filters

\[ \bigwedge_{\alpha \in \Lambda} \alpha.f_{\alpha} \xrightarrow{\beta} f_{\beta} \quad \text{if} \ \beta \in \Lambda \]

Contract coercion through a filter

\[
\begin{align*}
  f(0) &= 0 \\
  f(\alpha.\sigma) &= 0 & \text{if } f \trianglerighteq_{\alpha} \\
  f(\alpha.\sigma) &= \alpha.f'(\sigma) & \text{if } f \xrightarrow{\alpha} f'
\end{align*}
\]

\[
\begin{align*}
  f(\sigma_1 + \sigma_2) &= f(\sigma_1) + f(\sigma_2) \\
  f(\sigma_1 \oplus \sigma_2) &= f(\sigma_1) \oplus f(\sigma_2)
\end{align*}
\]

Property

\[ \sigma \preceq \tau \ \land \ \rho \vdash \sigma \ \iff \ \sigma \vdash f(\tau) \quad \text{for some filter } f \]
Results

Filters are “proofs” of subcontracting

\[ f : \sigma \leq \tau \]

- deduction system for subcontracting: \( f : \sigma \leq \tau \),
- algebraic theory for filters
- existence and effectiveness of a most general filter (via cut-elimination, yields subcontracting coherence)
- subcontracting decidability

Compliance characterises \textit{must testing}

\[ \sigma \sqsubseteq_{\text{must}} \tau \iff \rho \not\vdash \sigma \text{ implies } \rho \not\vdash \tau \]

Corollary

\[ \sigma \leq \tau \iff \rho \sqsubseteq_{\text{must}} f(\tau) \text{ for some filter } f \]
Results

Filters are “proofs” of subcontracting

\[ f : \sigma \leq \tau \]

- deduction system for subcontracting: \( f : \sigma \leq \tau \),
- algebraic theory for filters
- existence and effectiveness of a most general filter
  (via cut-elimination, yields subcontracting coherence)
- subcontracting decidability

Compliance characterises must testing

\[ \sigma \sqsubseteq_{must} \tau \iff \rho \not\vdash \sigma \text{ implies } \rho \not\vdash \tau \]

Corollary

\[ \sigma \leq \tau \iff \rho \sqsubseteq_{must} f(\tau) \text{ for some filter } f \]
Results

Filters are “proofs” of subcontracting

\[ f : \sigma \leq \tau \]

- deduction system for subcontracting: \( f : \sigma \leq \tau \),
- algebraic theory for filters
- existence and effectiveness of a most general filter (via cut-elimination, yields subcontracting coherence)
- subcontracting decidability

Compliance characterises must testing

\[ \sigma \sqsubseteq_{\text{must}} \tau \iff \rho \vdash \sigma \text{ implies } \rho \vdash \tau \]

Corollary

\[ \sigma \leq \tau \iff \rho \sqsubseteq_{\text{must}} f(\tau) \text{ for some filter } f \]
Results

Filters are “proofs” of subcontracting

\[ f : \sigma \leq \tau \]

- deduction system for subcontracting: \( f : \sigma \leq \tau \),
- algebraic theory for filters
- existence and effectiveness of a most general filter (via cut-elimination, yields subcontracting coherence)
- subcontracting decidability

Compliance characterises \textit{must testing}

\[ \sigma \sqsubseteq_{\text{must}} \tau \iff \rho \vdash \sigma \text{ implies } \rho \vdash \tau \]

\textbf{Corollary}

\[ \sigma \leq \tau \iff \rho \sqsubseteq_{\text{must}} f(\tau) \text{ for some filter } f \]
Results

Filters are “proofs” of subcontracting

\[ f : \sigma \leq \tau \]

- deduction system for subcontracting: \( f : \sigma \leq \tau \),
- algebraic theory for filters
- existence and effectiveness of a most general filter
  (via cut-elimination, yields subcontracting coherence)
- subcontracting decidability

Compliance characterises must testing

\[ \sigma \sqsubseteq_{must} \tau \iff \rho \vdash \sigma \text{ implies } \rho \vdash \tau \]

Corollary

\[ \sigma \leq \tau \iff \rho \sqsubseteq_{must} f(\tau) \text{ for some filter } f \]
Filters are “proofs” of subcontracting

\[ f : \sigma \preceq \tau \]

- deduction system for subcontracting: \( f : \sigma \preceq \tau \),
- algebraic theory for filters
- existence and effectiveness of a most general filter (via cut-elimination, yields subcontracting coherence)
- subcontracting decidability

Compliance characterises *must testing*

\[ \sigma \sqsubseteq_{\text{must}} \tau \iff \rho \not\vdash \sigma \text{ implies } \rho \not\vdash \tau \]

**Corollary**

\[ \sigma \preceq \tau \iff \rho \sqsubseteq_{\text{must}} f(\tau) \text{ for some filter } f \]
Results

Filters are “proofs” of subcontracting

\[ f : \sigma \leq \tau \]

- deduction system for subcontracting: \( f : \sigma \leq \tau \),
- algebraic theory for filters
- existence and effectiveness of a most general filter (via cut-elimination, yields subcontracting coherence)
- subcontracting decidability

Compliance characterises \textit{must testing}

\[ \sigma \must \leq \tau \iff \rho \not\vdash \sigma \text{ implies } \rho \not\vdash \tau \]

Corollary

\[ \sigma \leq \tau \iff \rho \must f(\tau) \text{ for some filter } f \]
Some details

- Identity:
  \[ I_\sigma \overset{\text{def}}{=} \bigsqcup_{\alpha \to \sigma', \alpha. I_{\sigma'}} \]

- Intersection: composes two filters
  \[ f \land g \overset{\text{def}}{=} \bigsqcup_{\alpha \in A \cap B} \alpha.(f_\alpha \land g_\alpha) \]

- Union: merges two filters
  \[ f \lor g \overset{\text{def}}{=} \bigsqcup_{\alpha \in A \cup B} \alpha. \begin{cases} f_\alpha \lor g_\alpha, & \alpha \in A \cap B \\ f_\alpha, & \alpha \in A \setminus B \\ g_\alpha, & \alpha \in B \setminus A \end{cases} \]
\[(\sigma = \tau \equiv l_\sigma : \sigma \leq \tau \text{ and } l_\tau : \tau \leq \sigma)\]

\[
\begin{align*}
\sigma + \sigma &= \sigma \\
\sigma + \tau &= \tau + \sigma \\
\sigma + (\sigma' + \sigma'') &= (\sigma + \sigma') + \sigma'' \\
\sigma + (\sigma' \oplus \sigma'') &= (\sigma + \sigma') \oplus (\sigma + \sigma'') \\
\sigma + \mathbf{0} &= \sigma \\
\alpha.\sigma + \alpha.\tau &= \alpha.(\sigma \oplus \tau) \\
\alpha.\sigma \oplus \alpha.\tau &= \alpha.(\sigma \oplus \tau)
\end{align*}
\]

(MUST)
\[l_\sigma : \sigma \oplus \tau \leq \sigma\]
(DepthExt)
\[\mathbf{0} : \mathbf{0} \leq \sigma\]
(WEAKENING)
\[
\begin{array}{c}
f : \sigma \leq \tau \\
g \land l_\tau \leq f
\end{array} \\
f \lor g : \sigma \leq \tau
\]
(TRANSITIVITY)
\[
\begin{array}{c}
f : \sigma \leq \sigma' \\
g : \sigma' \leq \sigma''
\end{array} \\
f \land g : \sigma \leq \sigma''
\]
(PREFIX)
\[
\begin{array}{c}
f : \sigma \leq \tau
\end{array} \\
\alpha.f : \alpha.\sigma \leq \alpha.\tau
\]
(INTCHOICE)
\[
\begin{array}{c}
f : \sigma \leq \sigma' \\
f : \tau \leq \tau'
\end{array} \\
f : \sigma \oplus \tau \leq \sigma' \oplus \tau'
\]
(EXTCHOICE)
\[
\begin{array}{c}
f : \sigma \leq \sigma' \\
f : \tau \leq \tau'
\end{array} \\
f : \sigma + \tau \leq \sigma' + \tau'
\]
Some properties

- The sought subtyping relations

\[(\text{MUST})\]
\[l_\sigma : \sigma \oplus \tau \leq \sigma\]

\[(\text{DepExt+Prefix})\]
\[\alpha.0 : \alpha.0 \leq \alpha.\sigma\]

\[(\text{DepExt+ExtCh+Weak})\]
\[l_\sigma \land l_\tau \leq 0\]
\[l_\sigma : \sigma \leq \sigma + \tau\]

- Canonical filter for \(\sigma \leq \tau\) (algorithmic)

\[\bigvee \left\{ f \mid f : \sigma \leq \tau \text{ and } f \leq l_\tau \right\}\]

- Characterisation of must testing for canonical filters

\[\sigma \sqsubseteq_{\text{must}} \tau \iff l_\tau : \sigma \leq \tau\]
Application to languages

Our contracts work for any language as long as they come equipped with:

1. An LTS

\[ P \xrightarrow{\mu} P' \]

\( \mu \) is either a visible action or an invisible \( \tau \) action

2. A type system

\[ \vdash P : \sigma \]

\( \sigma \) is a contract

3. The latter abstracts the former:

1. If \( \vdash P : \sigma \) and \( \sigma \xrightarrow{\alpha} \), then \( P \xrightarrow{\alpha} \)
2. If \( \vdash P : \sigma \) and \( P \xrightarrow{\mu} P' \) then \( \vdash P' : \sigma' \) and
   - if \( \mu = \tau \), then \( \sigma \xrightarrow{\text{must}} \sigma' \)
   - if \( \mu = \alpha \), then \( \sigma \xrightarrow{\alpha} \xrightarrow{\text{must}} \sigma' \)

\( \xrightarrow{\text{must}} \) measures non-determinism
Our contracts work for any language as long as they come equipped with:

1. **An LTS**

   \[ P \xrightarrow{\mu} P' \]

   \(\mu\) is either a visible action or an invisible \(\tau\) action

2. **A type system**

   \[ \vdash P : \sigma \]

   \(\sigma\) is a contract

3. **The latter abstracts the former:**

   - If \(\vdash P : \sigma\) and \(\sigma \xrightarrow{\alpha} \), then \(P \xrightarrow{\alpha}\)
   - If \(\vdash P : \sigma\) and \(P \xrightarrow{\mu} P'\) then \(\vdash P' : \sigma'\) and
     - if \(\mu = \tau\), then \(\sigma \xrightarrow{\alpha} \sigma'\)
     - if \(\mu = \alpha\), then \(\sigma \xrightarrow{\alpha} \sigma'\)

\(\xrightarrow{\text{must}}\) measures non-determinism
Application to languages

Our contracts work for any language as long as they come equipped with:

1. An LTS

\[ P \xrightarrow{\mu} P' \]

\( \mu \) is either a visible action or an invisible \( \tau \) action

2. A type system

\[ \vdash P : \sigma \]

\( \sigma \) is a contract

3. The latter abstracts the former:

1. If \( \vdash P : \sigma \) and \( \sigma \xrightarrow{\alpha} \), then \( P \xrightarrow{\alpha} \)

2. If \( \vdash P : \sigma \) and \( P \xrightarrow{\mu} P' \) then \( \vdash P' : \sigma' \) and
   - if \( \mu = \tau \), then \( \sigma \sqsubseteq_{\text{must}} \sigma' \)
   - if \( \mu = \alpha \), then \( \sigma \xrightarrow{\alpha} \sqsubseteq_{\text{must}} \sigma' \)

\[ \sqsubseteq_{\text{must}} \] measures non-determinism

Castagna, Gesbert, and Padovani  A Theory of Contracts for Web Services
Application to languages

Our contracts work for any language as long as they come equipped with:

1 An LTS

\[ P \xrightarrow{\mu} P' \]

\( \mu \) is either a visible action or an invisible \( \tau \) action

2 A type system

\( \vdash P : \sigma \)

\( \sigma \) is a contract

3 The latter abstracts the former:

1 If \( \vdash P : \sigma \) and \( \sigma \xrightarrow{\alpha} \), then \( P \xrightarrow{\alpha} \)

2 If \( \vdash P : \sigma \) and \( P \xrightarrow{\mu} P' \) then \( \vdash P' : \sigma' \) and

   - if \( \mu = \tau \), then \( \sigma \xrightarrow{\mu} \sigma' \)
   - if \( \mu = \alpha \), then \( \sigma \xrightarrow{\alpha} \sigma' \)

\( \xrightarrow{\text{must}} \) measures non-determinism
Process compliance

By the LTS we define sessions and compliance for processes:

- If $P \xrightarrow{\alpha} P'$ and $Q \xrightarrow{\bar{\alpha}} Q'$ then $P \parallel Q \rightarrow P' \parallel Q'$ (plus $\tau$-moves).

- Client $P$ complies with server $Q$ (noted $P \vdash Q$) if:
  - if $P \xrightarrow{\mu}$, then $\mu = \emptyset$ or
  - $P \parallel Q \rightarrow P' \parallel Q'$ and $P' \vdash Q'$

Theorem (Process compliance):

If $\vdash P : \rho$ and $\vdash Q : \sigma$ and $\rho \vdash \sigma$, then $P \vdash Q$. 

Castagna, Gesbert, and Padovani
Process compliance

By the LTS we define sessions and compliance for processes

- If $P \xrightarrow{\alpha} P'$ and $Q \xrightarrow{\bar{\alpha}} Q'$ then $P \parallel Q \rightarrow P' \parallel Q'$ (plus $\tau$-moves)

- Client $P$ complies with server $Q$ (noted $P \vdash Q$) if
  - if $P \mu \rightarrow$, then $\mu = \emptyset$ or
  - $P \parallel Q \rightarrow P' \parallel Q'$ and $P' \vdash Q'$

Theorem (Process compliance)

If $\vdash P : \rho$ and $\vdash Q : \sigma$ and $\rho \dashv \vdash \sigma$, then $P \vdash Q$
Process compliance

By the LTS we define sessions and compliance for processes

- If $P \xrightarrow{\alpha} P'$ and $Q \xrightarrow{-\alpha} Q'$ then $P \parallel Q \rightarrow P' \parallel Q'$ (plus $\tau$-moves)

- Client $P$ complies with server $Q$ (noted $P \vdash Q$) if
  - if $P \xrightarrow{\mu}$, then $\mu = \emptyset$ or
  - $P \parallel Q \rightarrow P' \parallel Q'$ and $P' \vdash Q'$

Theorem (Process compliance)

*If $\vdash P : \rho$ and $\vdash Q : \sigma$ and $\rho \vdash \sigma$, then $P \vdash Q$*
**Process Filtering**

Add filters to the language: \( f[P] \)

**Transition rules for filters**

\[
\begin{align*}
(FILTER1) \quad & P \xrightarrow{\alpha} P' \quad f \xrightarrow{\alpha} f' \\
& f[P] \xrightarrow{\alpha} f'[P']
\end{align*}
\]

\[
\begin{align*}
(FILTER2) \quad & P \xrightarrow{\tau} P' \\
& f[P] \xrightarrow{\tau} f[P']
\end{align*}
\]

**Typing rules for filters**

\[
(T-FILTER) \quad \vdash P : \sigma \\
\vdash f[P] : f(\sigma)
\]

“Subject reduction” still holds

**Theorem**

Let \( P \vdash p \) with \( p \vdash \sigma \). If \( Q \vdash \sigma \) and \( f : \sigma \rightarrow \tau \), then \( P \vdash f[Q] \).
Process Filtering

Add filters to the language: $f[P]$

Transition rules for filters

\[
\begin{align*}
\text{(FILTER1)} & \\
 P & \xrightarrow{\alpha} P' & f & \xrightarrow{\alpha} f' \\
\hline
 f[P] & \xrightarrow{\alpha} f'[P']
\end{align*}
\]

\[
\begin{align*}
\text{(FILTER2)} & \\
 P & \xrightarrow{\tau} P' \\
\hline
 f[P] & \xrightarrow{\tau} f'[P']
\end{align*}
\]

Typing rules for filters

\[
\begin{align*}
\text{(T-FILTER)} & \\
\vdash P : \sigma \\
\hline
\vdash f[P] : f(\sigma)
\end{align*}
\]

“Subject reduction” still holds

Theorem (Process filtering)

Let $\vdash P : \rho$ with $\rho \vdash \sigma$. If $\vdash Q : \tau$ and $f : \sigma \leq \tau$, then $P \vdash f[Q]$.
Process Filtering

Add filters to the language: \( f[P] \)

Transition rules for filters

\[
\begin{align*}
\text{(FILTER1)} & \quad P \xrightarrow{\alpha} P' \\
& \quad f \xrightarrow{\alpha} f' \\
\hline
& \quad f[P] \xrightarrow{\alpha} f'[P']
\end{align*}
\]

\[
\begin{align*}
\text{(FILTER2)} & \quad P \xrightarrow{\tau} P' \\
& \quad f[P] \xrightarrow{\tau} f[P']
\end{align*}
\]

Typing rules for filters

\[
\begin{align*}
\text{(T-FILTER)} & \quad \vdash P : \sigma \\
& \quad \vdash f[P] : f(\sigma)
\end{align*}
\]

“Subject reduction” still holds

Theorem (Process filtering)

Let \( \vdash P : \rho \) with \( \rho \vdash \sigma \). If \( \vdash Q : \tau \) and \( f : \sigma \preceq \tau \), then \( P \vdash f[Q] \)
Process Filtering

Add filters to the language: $f[P]$

**Transition rules for filters**

\[(\text{FILTER1})\]

\[
P \xrightarrow{\alpha} P' \quad f \xrightarrow{\alpha} f' \quad \quad f[P] \xrightarrow{\alpha} f'[P']
\]

\[(\text{FILTER2})\]

\[
P \xrightarrow{\tau} P' \quad \quad f[P] \xrightarrow{\tau} f[P']
\]

**Typing rules for filters**

\[(\text{T-FILTER})\]

\[
\vdash P : \sigma \\

\vdash f[P] : f(\sigma)
\]

“Subject reduction” still holds

**Theorem (Process filtering)**

Let $\vdash P : \rho$ with $\rho \vdash \sigma$. If $\vdash Q : \tau$ and $f : \sigma \leq \tau$, then $P \vdash f[Q]$
**Application to service discovery**

Search for services compliant with a client $\rho$:

\[ \text{discover}(\rho) = \left\{ (\sigma, f) \mid \rho \vdash f(\sigma) \right\} \]

Call one of the services in the result by using the associated filter.

More efficient using subcontracts and caching

1. Compute $\rho \upharpoonright$ the “canonical server” of $\rho$
2. Return all compatible servers:

\[ \text{discover}(\rho) = \left\{ (\sigma, f) \mid f : \rho \upharpoonright \preceq \sigma \right\} \quad \text{(cached)} \]

Finer-grained searches

Define some minimal behaviour that must not be filtered out:

\[ \text{discover}(\rho, g) = \left\{ (\sigma, f) \mid f : \rho \upharpoonright \preceq \sigma \land g \preceq f \right\} \]

E.g.: provide at least Login.ValidLogin.Query.Catalog.Purchase.Accepted so as to avoid, say, Login.InvalidLogin.
Application to service discovery

Search for services compliant with a client $\rho$:

$$\text{discover}(\rho) = \{ (\sigma, f) \mid \rho \vdash f(\sigma) \}$$

Call one of the services in the result by using the associated filter.

More efficient using subcontracts and caching

1. Compute $\rho^\bot$ the “canonical server” of $\rho$
2. Return all compatible servers:
   $$\text{discover}(\rho) = \{ (\sigma, f) \mid f : \rho^\bot \preceq \sigma \}$$  (cached)

Finer-grained searches

Define some minimal behaviour that must not be filtered out:

$$\text{discover}(\rho, g) = \{ (\sigma, f) \mid f : \rho^\bot \preceq \sigma \land g \preceq f \}$$

E.g.: provide at least $\text{Login}.\text{ValidLogin}.\text{Query}.\text{Catalog}.\text{Purchase}.\text{Accepted}$

so as to avoid, say, $\text{Login}.\text{InvalidLogin}$.
Application to service discovery

Search for services compliant with a client $\rho$:

$$\text{discover}(\rho) = \{(\sigma, f) \mid \rho \vdash f(\sigma)\}$$

Call one of the services in the result by using the associated filter.

More efficient using subcontracts and caching

1. Compute $\rho^\perp$ the “canonical server” of $\rho$
2. Return all compatible servers:

$$\text{discover}(\rho) = \{(\sigma, f) \mid f : \rho^\perp \preceq \sigma\} \quad \text{(cached)}$$

Finer-grained searches

Define some minimal behaviour that must not be filtered out:

$$\text{discover}(\rho, g) = \{(\sigma, f) \mid f : \rho^\perp \preceq \sigma \land g \preceq f\}$$

E.g.: provide at least $\text{Login.ValidLogin.Query.Catalog.Purchase.Accepted}$ so as to avoid, say, $\text{Login.InValidLogin}$. 
Application to service discovery

Search for services compliant with a client $\rho$:
\[
discover(\rho) = \{(\sigma, f) \mid \rho \vdash f(\sigma)\}
\]
Call one of the services in the result by using the associated filter.

More efficient using subcontracts and caching

1. Compute $\rho^\perp$ the “canonical server” of $\rho$
2. Return all compatible servers:
\[
discover(\rho) = \{(\sigma, f) \mid f : \rho^\perp \preceq \sigma\}
\]
(cached)

Finer-grained searches
Define some minimal behaviour that must not be filtered out:
\[
discover(\rho, g) = \{(\sigma, f) \mid f : \rho^\perp \preceq \sigma \land g \preceq f\}
\]

E.g.: provide at least Login.ValidLogin.Query.Catalog.Purchase.Accepted so as to avoid, say, Login.InvalidLogin.
Application to service discovery

Search for services compliant with a client $\rho$:

$$\text{discover}(\rho) = \{(\sigma, f) \mid \rho \vdash f(\sigma)\}$$

Call one of the services in the result by using the associated filter.

More efficient using subcontracts and caching

1. Compute $\rho^\perp$ the “canonical server” of $\rho$
2. Return all compatible servers:

$$\text{discover}(\rho) = \{(\sigma, f) \mid f : \rho^\perp \preceq \sigma\}$$

(cached)

Finer-grained searches

Define some minimal behaviour that must not be filtered out:

$$\text{discover}(\rho, g) = \{(\sigma, f) \mid f : \rho^\perp \preceq \sigma \land g \preceq f\}$$

E.g.: provide at least $\text{Login.ValidLogin.Query.Catalog.Purchase.Accepted}$ so as to avoid, say, $\text{Login.InvalidLogin}$.
Application to service discovery

Search for services compliant with a client $\rho$:

$\text{discover}(\rho) = \{ (\sigma, f) \mid \rho \vdash f(\sigma) \}$

Call one of the services in the result by using the associated filter.

More efficient using subcontracts and caching

1. Compute $\rho^\perp$ the “canonical server” of $\rho$
2. Return all compatible servers:

$\text{discover}(\rho) = \{ (\sigma, f) \mid f : \rho^\perp \preceq \sigma \}$ (cached)

Finer-grained searches

Define some minimal behaviour that must not be filtered out:

$\text{discover}(\rho, g) = \{ (\sigma, f) \mid f : \rho^\perp \preceq \sigma \land g \preceq f \}$

E.g.: provide at least \underline{Login.\hspace{1em}ValidLogin.\hspace{1em}Query.\hspace{1em}Catalog.\hspace{1em}Purchase.\hspace{1em}Accepted} so as to avoid, say, \underline{Login.\hspace{1em}InvalidLogin}.
Application to service discovery

Search for services compliant with a client $\rho$:

$$\text{discover}(\rho) = \{(\sigma, f) \mid \rho \vdash f(\sigma)\}$$

Call one of the services in the result by using the associated filter.

More efficient using subcontracts and caching

1. Compute $\rho \perp$ the “canonical server” of $\rho$
2. Return all compatible servers:

$$\text{discover}(\rho) = \{(\sigma, f) \mid f : \rho \perp \preceq \sigma\}$$ (cached)

Finer-grained searches

Define some minimal behaviour that must not be filtered out:

$$\text{discover}(\rho, g) = \{(\sigma, f) \mid f : \rho \perp \preceq \sigma \land g \preceq f\}$$

E.g.: provide at least \text{Login.ValidLogin.Query.Catalog.Purchase.Accepted} so as to avoid, say, \text{Login.InvalidLogin}.
Application to service discovery

Search for services compliant with a client $\rho$:

$$\text{discover}(\rho) = \{(\sigma, f) \mid \rho \vdash f(\sigma)\}$$

Call one of the services in the result by using the associated filter.

More efficient using subcontracts and caching

1. Compute $\rho^\bot$ the “canonical server” of $\rho$
2. Return all compatible servers:

$$\text{discover}(\rho) = \{(\sigma, f) \mid f : \rho^\bot \preceq \sigma\}$$

(cached)

Finer-grained searches

Define some minimal behaviour that must not be filtered out:

$$\text{discover}(\rho, g) = \{(\sigma, f) \mid f : \rho^\bot \preceq \sigma \land g \preceq f\}$$

E.g.: provide at least $\text{Login. ValidLogin. Query. Catalog. Purchase. Accepted}$ so as to avoid, say, $\text{Login. InvalidLogin}$. 

Castagna, Gesbert, and Padovani

A Theory of Contracts for Web Services
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients
- Technical device: filters
- Filters as explicit coercions between contracts and as proofs for subcontracting.
- Algorithmic counterpart obtained via cut-elimination.
- Theory independent from the language(s) used to implement clients and services.
- Applications: specification and discovery of services.
- Future work
  - Recursion (almost done)
  - Application to WS-BPEL (almost done: pretty easy)
  - Integration with session types (in progress)
  - Choreography, Messages, and higher order channels
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients

- Technical device: filters
  - Filters as explicit coercions between contracts and as proofs for subcontracting.
  - Algorithmic counterpart obtained via cut-elimination.
  - Theory independent from the language(s) used to implement clients and services.
  - Applications: specification and discovery of services.

Future work
- Recursion (almost done)
- Application to WS-BPEL (almost done: pretty easy)
- Integration with session types (in progress)
- Choreography, Messages, and higher order channels
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients

- Technical device: filters

- Filters as explicit coercions between contracts and as proofs for subcontracting.

- Algorithmic counterpart obtained via cut-elimination.

- Theory independent from the language(s) used to implement clients and services.

- Applications: specification and discovery of services.

- Future work
  - Recursion (almost done)
  - Application to WS-BPEL (almost done: pretty easy)
  - Integration with session types (in progress)
  - Choreography, Messages, and higher order channels
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients

- Technical device: filters

- Filters as explicit coercions between contracts and as proofs for subcontracting.

- Algorithmic counterpart obtained via cut-elimination.

- Theory independent from the language(s) used to implement clients and services.

- Applications: specification and discovery of services.

- Future work
  - Recursion (almost done)
  - Application to WS-BPEL (almost done: pretty easy)
  - Integration with session types (in progress)
  - Choreography, Messages, and higher order channels
Conclusion

- **The approach reconciles two desiderata:**
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients
- **Technical device: filters**
- **Filters as explicit coercions between contracts and as proofs for subcontracting.**
- **Algorithmic counterpart obtained via cut-elimination.**
- **Theory independent from the language(s) used to implement clients and services.**
- **Applications:** specification and discovery of services.
- **Future work**
  - Recursion *(almost done)*
  - Application to WS-BPEL *(almost done: pretty easy)*
  - Integration with session types *(in progress)*
  - Choreography, Messages, and higher order channels
The approach reconciles two desiderata:
- upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
- upgrade transparency for old clients

Technical device: filters

Filters as explicit coercions between contracts and as proofs for subcontracting.

Algorithmic counterpart obtained via cut-elimination.

Theory independent from the language(s) used to implement clients and services.

Applications: specification and discovery of services.

Future work
- Recursion (almost done)
- Application to WS-BPEL (almost done: pretty easy)
- Integration with session types (in progress)
- Choreography, Messages, and higher order channels
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients

- Technical device: filters

- Filters as explicit coercions between contracts and as proofs for subcontracting.

- Algorithmic counterpart obtained via cut-elimination.

- Theory independent from the language(s) used to implement clients and services.

- Applications: specification and discovery of services.

- Future work
  - Recursion (almost done)
  - Application to WS-BPEL (almost done: pretty easy)
  - Integration with session types (in progress)
  - Choreography, Messages, and higher order channels
Conclusion

The approach reconciles two desiderata:
- upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
- upgrade transparency for old clients

Technical device: filters

Filters as explicit coercions between contracts and as proofs for subcontracting.

Algorithmic counterpart obtained via cut-elimination.

Theory independent from the language(s) used to implement clients and services.

Applications: specification and discovery of services.

Future work
- Recursion (almost done)
- Application to WS-BPEL (almost done: pretty easy)
- Integration with session types (in progress)
- Choreography, Messages, and higher order channels
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients
- Technical device: filters
- Filters as explicit coercions between contracts and as proofs for subcontracting.
- Algorithmic counterpart obtained via cut-elimination.
- Theory independent from the language(s) used to implement clients and services.
- Applications: specification and discovery of services.
- Future work
  - Recursion *(almost done)*
  - Application to WS-BPEL *(almost done: pretty easy)*
  - Integration with session types *(in progress)*
  - Choreography, Messages, and higher order channels
Conclusion

The approach reconciles two desiderata:
- upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
- upgrade transparency for old clients

Technical device: filters

Filters as explicit coercions between contracts and as proofs for subcontracting.

Algorithmic counterpart obtained via cut-elimination.

Theory independent from the language(s) used to implement clients and services.

Applications: specification and discovery of services.

Future work
- Recursion (almost done)
- Application to WS-BPEL (almost done: pretty easy)
- Integration with session types (in progress)
- Choreography, Messages, and higher order channels

Full details in the forthcoming POPL 2008 paper Castagna, Gesbert, and Padovani
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients

- Technical device: filters
- Filters as explicit coercions between contracts and as proofs for subcontracting.
- Algorithmic counterpart obtained via cut-elimination.
- Theory independent from the language(s) used to implement clients and services.

- Applications: specification and discovery of services.

- Future work
  - Recursion (almost done)
  - Application to WS-BPEL (almost done: pretty easy)
  - Integration with session types (in progress)
  - Choreography, Messages, and higher order channels
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients
- Technical device: filters
- Filters as explicit coercions between contracts and as proofs for subcontracting.
- Algorithmic counterpart obtained via cut-elimination.
- Theory independent from the language(s) used to implement clients and services.
- Applications: specification and discovery of services.
- Future work
  - Recursion (almost done)
  - Application to WS-BPEL (almost done: pretty easy)
  - Integration with session types (in progress)
  - Choreography, Messages, and higher order channels

Full details in the forthcoming POPL 2008 paper

Castagna, Gesbert, and Padovani
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients
- Technical device: filters
- Filters as explicit coercions between contracts and as proofs for subcontracting.
- Algorithmic counterpart obtained via cut-elimination.
- Theory independent from the language(s) used to implement clients and services.
- Applications: specification and discovery of services.
- Future work
  - Recursion (almost done)
  - Application to WS-BPEL (almost done: pretty easy)
  - Integration with session types (in progress)
  - Choreography, Messages, and higher order channels

Full details in the forthcoming POPL 2008 paper

Castagna, Gesbert, and Padovani
A Theory of Contracts for Web Services
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients
- Technical device: filters
- Filters as explicit coercions between contracts and as proofs for subcontracting.
- Algorithmic counterpart obtained via cut-elimination.
- Theory independent from the language(s) used to implement clients and services.
- Applications: specification and discovery of services.
- Future work
  - Recursion (almost done)
  - Application to WS-BPEL (almost done: pretty easy)
  - Integration with session types (in progress)
  - Choreography, Messages, and higher order channels

Full details in the forthcoming POPL 2008 paper

Castagna, Gesbert, and Padovani
Conclusion

- The approach reconciles two desiderata:
  - upgrade services by more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones;
  - upgrade transparency for old clients
- Technical device: filters
- Filters as explicit coercions between contracts and as proofs for subcontracting.
- Algorithmic counterpart obtained via cut-elimination.
- Theory independent from the language(s) used to implement clients and services.
- Applications: specification and discovery of services.
- Future work
  - Recursion (almost done)
  - Application to WS-BPEL (almost done: pretty easy)
  - Integration with session types (in progress)
  - Choreography, Messages, and higher order channels

Full details in the forthcoming POPL 2008 paper