Semantic Analysis of Dynamic Connector Based Architecture Styles

Guoxin Su   Mingsheng Ying   Chengqi Zhang

Faculty of Engineering and Information Technology
University of Technology, Sydney

WICSA/ECSA 2012

Outline

Background: where our problem locates

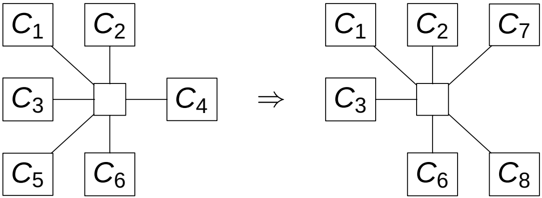
Problem: a motivating example and behavioural properties

Model: formalisation of architectural concepts and properties

Analysis: algorithms for checking desired properties

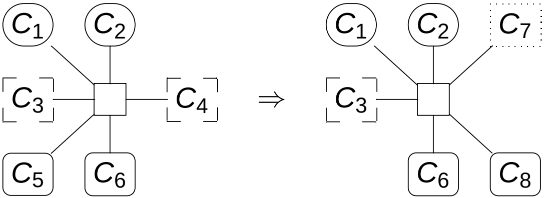
Background

- Dynamism of connector-based architectural styles: insertion and removal of components
- Type- vs instance-level descriptions and component instantiation: parameterisation or semantic conformance
- Behavioural modeling



Background

- Dynamism of connector-based architectural styles: insertion and removal of components
- Type- vs instance-level descriptions and component instantiation: parameterisation or semantic conformance
- Behavioural modeling



Background

- Dynamism of connector-based architectural styles: insertion and removal of components
- Type- vs instance-level descriptions and component instantiation: parameterisation or semantic conformance
- Behavioural modeling

  Syntax of textual specification:

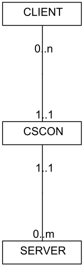
  if $\mathit{state} = x$ **then**

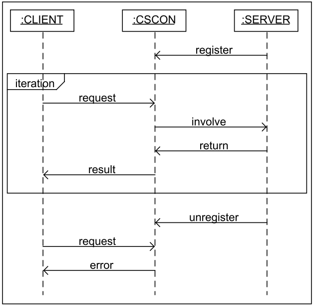
  if pre-conditions **then**

  input/output **and** effects [**and** $\mathit{state} := y$]

A Client-Server System

- The system is connector-based
- A structural veiw and a scenario:





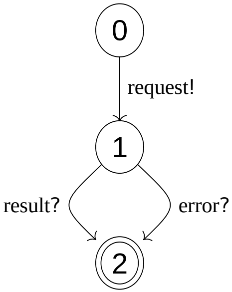
- Components can join in and disconnect to the system dynamically

A Client-Server System

Type-level specification

Component type $\mathit{CLIENT}(c : \mathit{clt}, s : \mathit{sev})$:

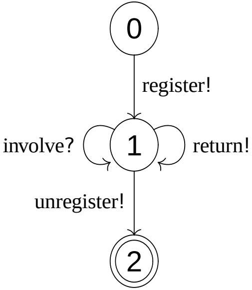
if $\mathit{state} = 0$ **then**
 $\langle \mathit{request}, c, s\rangle!$ **and** $\mathit{state} := 1$
if $\mathit{state} = 1$ **then**
 if true then
  $\langle \mathit{result}, c, s\rangle?$ **and** $\mathit{state} := 2$
 if true then
  $\langle \mathit{error}, c, s\rangle?$ **and** $\mathit{state} := 2$



A Client-Server System

Component type $\mathit{SERVER}(s : \mathit{sev})$:

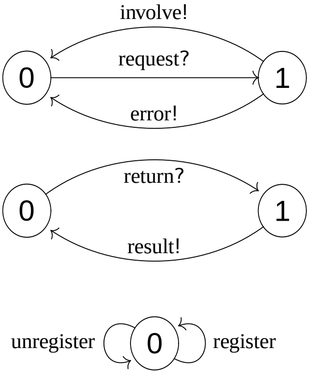
$$
\begin{array}{l}
\textbf{if } \mathit{state} = 0 \textbf{ then} \\
\quad \langle \mathit{register}, s\rangle! \textbf{ and } \mathit{state} := 1 \\
\textbf{if } \mathit{state} = 1 \textbf{ then} \\
\quad \textbf{if true then} \\
\quad\quad \langle \mathit{involve}, x : \mathit{clt}\rangle? \textbf{ and} \\
\quad\quad \mathrm{enqueue}(x, \mathit{Que}) \\
\quad \textbf{if } \mathrm{empty}(\mathit{Que}) = \text{‘n’} \textbf{ then} \\
\quad\quad \textbf{let } y = \mathrm{head}(\mathit{Que}) \textbf{ and} \\
\quad\quad \langle \mathit{return}, y\rangle! \textbf{ and } \mathrm{dequeue}(\mathit{Que}) \\
\quad \textbf{if } \mathrm{empty}(\mathit{Que}) = \text{‘y’} \textbf{ then} \\
\quad\quad \langle \mathit{unregister}, s\rangle! \textbf{ and } \mathit{state} := 2
\end{array}
$$



A Client-Server System

Connector *CSCON*:

if $\mathit{state}_1 = 0$ **then**
  $\langle \mathit{request}, x : \mathit{clt}, y : \mathit{sev} \rangle ?$
if $\mathit{state}_1 = 1$ **then**
  if $y \in \mathit{RegSev}$ **then**
    $\langle \mathit{involve}, x, y \rangle !$ **and** $\mathit{state}_1 := 0$
  else $\langle \mathit{error}, x, y \rangle !$ **and** $\mathit{state}_1 := 0$
if $\mathit{state}_2 = 0$ **then**
  $\langle \mathit{return}, z : \mathit{clt}, w : \mathit{sev} \rangle ?$ **and** $\mathit{state}_2 := 1$
if $\mathit{state}_2 = 1$ **then**
  $\langle \mathit{result}, z, w \rangle !$ **and** $\mathit{state}_2 := 0$
if $\mathit{state}_3 = 0$ **then**
  if true then
    $\langle \mathit{register}, v : \mathit{sev} \rangle ?$ **and**
    $\mathit{RegSev} := \mathit{RegSev} \cup \{v\}$
  if true then
    $\langle \mathit{unregister}, u : \mathit{sev} \rangle ?$ **and**
    $\mathit{RegSev} := \mathit{RegSev} \backslash \{u\}$

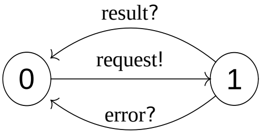


A Client-Server System

Instance-level specification

Component instance Client_1 of CLIENT:

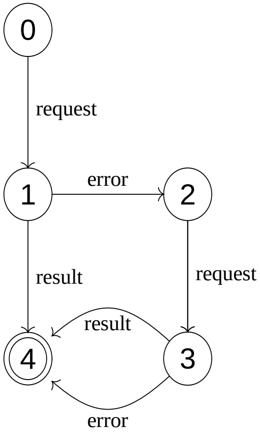
if $\mathit{state} = 1$ **then**
  $\langle \mathit{request}, c_1, s_1 \rangle !$ **and** $\mathit{state} := 2$
if $\mathit{state} = 2$ **then**
  if true then
    $\langle \mathit{result}, c_1, s_1 \rangle ?$ **and** $\mathit{state} := 1$
  if true then
    $\langle \mathit{error}, c_1, s_1 \rangle ?$ **and** $\mathit{state} := 1$



A Client-Server System

Component instance $Client_2$ of $CLIENT$:

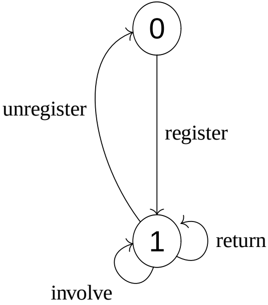
if $state = 0$ **then**
  $\langle request, c_2, s_1 \rangle?$ **and** $state := 1$
if $state = 1$ **then**
  if true then
    $\langle result, c_2, s_1 \rangle?$ **and** $state := 4$
  if true then
    $\langle error, c_2, s_1 \rangle?$ **and** $state := 2$
if $state = 2$ **then**
  choose $any \in sev$ **and**
  $\langle request, c_2, any \rangle!$ **and** $state := 3$
if $state = 3$ **then**
  if true then
    $\langle result, clt_2, any \rangle?$ **and** $state := 4$
  if true then
    $\langle error, clt_2, any \rangle?$ **and** $state := 4$



A Client-Server System

Component instance $Server_1$ of $SERVER$:

$$
\begin{array}{l}
\textbf{if } \textit{state} = 0 \textbf{ then} \\
\quad \textbf{if } \textit{upgrade} = \text{`done'} \textbf{ then} \\
\quad\quad \langle \textit{register}, s_1 \rangle! \textbf{ and } \textit{state} := 1 \\
\textbf{if } \textit{state} = 1 \textbf{ then} \\
\quad \textbf{if } \mathrm{empty}(\textit{Que}) = \text{`y'} \textbf{ and} \\
\quad\quad \textit{upgrade} = \text{`ready'} \textbf{ then} \\
\quad\quad \langle \textit{unregister}, s_1 \rangle! \textbf{ and} \\
\quad\quad \textit{state} := 0 \\
\quad \textbf{if true then} \\
\quad\quad \langle \textit{involve}, x : \textit{clt} \rangle? \textbf{ and} \\
\quad\quad \mathrm{enqueue}(x, \textit{Que}) \\
\quad \textbf{if } \mathrm{empty}(\textit{Que}) = \text{`n'} \textbf{ then} \\
\quad\quad \textbf{let } y = \mathrm{head}(\textit{Que}) \textbf{ and} \\
\quad\quad \langle \textit{return}, y \rangle! \textbf{ and } \mathrm{dequeue}(\textit{Que})
\end{array}
$$



Problems

Basic properties for the client-server system:

- Whether the system is deadlock-free?
- Whether each component, if not terminated, will be deprived of the right to interact with the connector?
- Whether *CSCON* restricts the system's behaviours?
- Whether behaviours of each component, if given a suitable configuration, are realisable?

- Can we know the answers to the above questions without exhausting all possibility of runtime system configurations?
- A pitfall: the semantic variances between component types and instances

Problems

Basic properties for the client-server system:

- Whether the system is deadlock-free?
- Whether each component, if not terminated, will be deprived of the right to interact with the connector?
- Whether *CSCON* restricts the system's behaviours?
- Whether behaviours of each component, if given a suitable configuration, are realisable?

- Can we know the answers to the above questions without exhausting all possibility of runtime system configurations?
- A pitfall: the semantic variances between component types and instances

Process Algebra

Syntax: $\mathcal{N}$ a set of names, $a_i \in \mathcal{N}$.

$$
\begin{array}{rcll}
\lambda & ::= & \langle a_1, \ldots, a_k \rangle & \textit{messages} \\
\alpha, \beta, \gamma & ::= & \lambda? \mid \lambda! \mid \tau & \textit{actions} \\
P, Q & ::= & X \mid \text{nil} \mid P \times Q \mid P \parallel Q & \textit{processes} \\
M, M' & ::= & M + M' \mid \lambda?.P \mid \lambda!.P &
\end{array}
$$

Operation semantics:

$$
\frac{-}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{P \xrightarrow{\alpha} P' \quad X \mathrel{\hat{=}} M}{X \xrightarrow{\alpha} P'}
$$

$$
\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{P \xrightarrow{\alpha} P'}{P \times Q \xrightarrow{\alpha} P' \times Q}
$$

$$
\frac{P \xrightarrow{\tau} P'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q} \qquad \frac{P \xrightarrow{\lambda!} P' \quad Q \xrightarrow{\lambda?} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}
$$

where M is '$\parallel$'- and '$\times$'-free.

- From behavioural specification to PA processes
- $\mathit{CSCON}, \mathit{CLIENT}, \mathit{SERVER}, \mathit{client}_1$, ect. as PA processes

*Recursive equations for Server_1

(1) if $\mathit{Que} = \epsilon$ and $\mathit{update} =$ 'ready', then

$$X[2,\epsilon] \doteq \sum_{a \in \mathit{clt}} \langle \mathit{involve}, a\rangle?.\ X[3,a] + \langle \mathit{unregister}, s\rangle!.\ X[3,\epsilon]$$

(2) if $\mathit{Que} \neq \epsilon$, then

$$X[2,\mathit{Que}] \doteq \sum_{a \in \mathit{clt}} \langle \mathit{involve}, a\rangle?.\ X[2,\mathit{Que}_a] + \langle \mathit{return}, c\rangle!.\ X[2,\mathit{Que}']$$

where $\mathit{Que}_a = \mathrm{enqueue}(a, \mathit{Que})$ such that $a \in \mathit{clt}$, $c = \mathrm{head}(\mathit{Que})$, and $\mathit{Que}' = \mathrm{dequeue}(\mathit{Que})$.

Main Concepts

An informal glimpse

- an architecture type $=$ component types + a connector
- an architecture (instance) $=$ components + a connector

Definition (Components, connectors, component types)

Components are '$\|$'-free processes and *connectors* are '$\|$'- and '$\times$'-free processes. *Component types* are '$\|$'-free abstract processes of the form

$$\mathcal{I} = Q(x_1 : A_1, \ldots, x_m : A_m)$$

where (1) $A_i \subseteq \mathcal{N}$ $(1 \leq i \leq m)$ are name spaces, and (2) x_i $(1 \leq i \leq m)$ are formal parameters of $\mathcal{I}$ with x_1 being a distinguished one (which, informally specking, is reserved for the name of an instance of $\mathcal{I}$).

Main Concepts

An informal glimpse

- an architecture type $=$ component types + a connector
- an architecture (instance) $=$ components + a connector

Definition (Components, connectors, component types)

Components are '$\|$'-free processes and *connectors* are '$\|$'- and '$\times$'-free processes. *Component types* are '$\|$'-free abstract processes of the form

$$\mathcal{I} = Q(x_1 : A_1, \ldots, x_m : A_m)$$

where (1) $A_i \subseteq \mathcal{N}$ ($1 \leq i \leq m$) are name spaces, and (2) x_i ($1 \leq i \leq m$) are formal parameters of $\mathcal{I}$ with x_1 being a distinguished one (which, informally specking, is reserved for the name of an instance of $\mathcal{I}$).

Main Concepts

Definition (Architecture types and instances)

A (dynamic, connector-based) *architecture type* is represented as the tuple

$$\mathcal{A}^t = \langle \mathcal{I}_1, \ldots, \mathcal{I}_n, C \rangle$$

An *architecture instance* of $\mathcal{A}^t$ is the tuple

$$\mathcal{A} = \langle P_1^1, \ldots P_1^{m_1}, \ldots, P_n^{m_n}, C \rangle$$

where P_i^j *conforms* to $\mathcal{I}_i$.

Example

$$\begin{aligned} CStype &= \langle CLIENT, SERVER, CSCON \rangle \\ CSsystem &= \langle Client_1, Client_2, \ldots, Server_1, \ldots, CSCON \rangle \end{aligned}$$

Main Concepts

Definition (Canonical components)

If $a \in A_1$, we call

$$\mathcal{I}\langle a \rangle = \sum_{a_2 \in A_2, \ldots, a_2 \in A_m} Q\langle a, a_2, \ldots a_m \rangle$$

a *canonical component* of $\mathcal{I}$.

Definition (Component conformance)

P *conforms* to $\mathcal{I}\langle a \rangle$, denoted $\mathcal{I}\langle a \rangle \vdash P$, if there is $R \subseteq Proc \times Proc$ such that $\langle \mathcal{I}\langle a \rangle, P \rangle \in R$ and for each $\langle P_1, P_2 \rangle \in R$:

- if $P_1 = nil$ then $\langle \mathcal{I}\langle a \rangle, P_2 \rangle \in R$ or $P_2 = nil$;
- if $P_1 \xrightarrow{\alpha} P_1'$ and $P_1 \neq \mathcal{I}\langle a \rangle$ and $P_2 \xrightarrow{\alpha} P_2'$ and $\langle P_1', P_2' \rangle \in R$ for some P_2';
- if $P_2 \xrightarrow{\alpha} P_2'$ then $P_1 \xrightarrow{\alpha} P_1'$ and $\langle P_1', P_2' \rangle \in R$ for some P_1'.

Main Concepts

Theorem (Properties of $\vdash$)

1. $\mathcal{I}\langle a\rangle \vdash \mathcal{I}\langle a\rangle$,
2. $\mathcal{I}\langle a\rangle \vdash P_1 \;\&\; \mathcal{I}\langle a\rangle \vdash P_2 \Rightarrow \mathcal{I}\langle a\rangle \vdash P_1 + P_2$,
3. $\mathcal{I}\langle a\rangle \vdash P_1 \;\&\; P_1 \simeq P_2 \Rightarrow \mathcal{I}\langle a\rangle \vdash P_2$,
4. $\mathcal{I}\langle a\rangle \vdash P \Rightarrow \mathcal{I}\langle a\rangle \vdash P^*$,
5. $\vdash$ *is decidable.*

Example

$$CLIENT\langle c_1 : clt\rangle \vdash Client_1$$
$$CLIENT\langle c_2 : clt\rangle \vdash Client_2$$
$$SERVER\langle s_1 : sev\rangle \vdash Server_1$$

Main Concepts

Definition (Component configurations)

A *component configuration* of $\mathcal{A}^t$ is a process of the form

$$F = P_1\langle a_1\rangle \times \ldots \times P_n\langle a_n\rangle$$

such that, for each $1 \leq i \neq j \leq n$, $a_i \neq a_j$ and $\mathcal{I}_i\langle a_i\rangle \vdash P_i$ for some interface $\mathcal{I}_i$ of $\mathcal{A}^t$.

Definition (Architectures revisited)

The semantics of an architecture $\mathcal{A}$ can be considered as the process $F \parallel C$.

Example

$$CSsystem = (client_1 \times client_2 \times \ldots \times server_1 \times \ldots) \parallel CSCON$$

Properties

Definition (Deadlock-freedom)

(1) An architecture instance $\mathcal{A} = F \,\|\, C$ is *deadlock-free*, if the following proposition holds: if $\mathcal{A} \xrightarrow{*} F' \,\|\, C'$ and $F' \longrightarrow$, then $F' \,\|\, C' \longrightarrow$. (2) An architecture type $\mathcal{A}^t$ is deadlock-free if each instance of $\mathcal{A}^t$ is deadlock-free.

Definition (Non-starvation)

$\mathcal{A} = (F \times P) \,\|\, C$ is *non-starving*, if the following holds: if $\mathcal{A} \xrightarrow{*} (F' \times P') \,\|\, C'$ and $P' \longrightarrow$, then there are F'' and C'' such that $F' \,\|\, C' \xrightarrow{*} F'' \,\|\, C''$ and $P' \,\|\, C'' \longrightarrow$. (2) An architecture type $\mathcal{A}^t$ is non-starving if each instance of $\mathcal{A}^t$ is non-starving.

Lemma

Non-starvation implies deadlock-freedom.

Properties

Conservation: behaviours of architecture instances are refined by the connector.

Definition (Conservation)

An architecture type $\mathcal{A}^t$ is *conservative*, if, for each $\widetilde{\alpha}$ such that $C \xrightarrow{\widetilde{\alpha}}$, there is a configuration F such that $F \xrightarrow{\sharp\widetilde{\alpha}}$ where $\sharp\widetilde{\alpha}$ is the dual sequence of $\widetilde{\alpha}$ w.r.t. $\{?, !\}$.

Completeness: the connector does not exclude behaviours of components.

Definition (Completeness)

$\mathcal{A}^t$ is complete if the following proposition holds: for each component P and $P' \in Proc(P)$, if $P' \xrightarrow{\alpha}$, then $(F \times P) \parallel C \xrightarrow{*} (F' \times P') \parallel C'$ for some F, F', C' such that $C' \xrightarrow{\sharp\alpha}$.

The Method

The method is to construct a specific architecture instance which can mimic just all behaviours of possible components.

Definition (Construction procedure)

For any given component P, we choose a new process identifier X_P. The *iteration* of P, denoted by P^*, is obtained by substituting *nil* in P by X_P, and the recursive equation for X_P is $X_P \stackrel{\circ}{=} P^*$. A canonical configuration and a canonical architecture instance are respectively defined as

$$F_c^* = X_{P_{c,1}} \times \ldots \times X_{P_{c,k}} \qquad \mathcal{A}_c^* = F_c^* \parallel C$$

where $P_{c,1}, \ldots, P_{c,k}$ enumerate all canonical components of $\mathcal{A}^t$.

The Method

N.B. The number of canonical components of an architecture type is the number of *possible* components. For example, the number of canonical components of *CStype* is $|clt| + |sev|$. But the number of possible configurations for *CStype* is $2^{|clt|+|sev|}$.

The following lemma says that the iteration of a canonical component's behaviours are just enough to mimic all of its components' behaviours in some sense.

Lemma

- *If* $P \xrightarrow{\widetilde{\alpha}}$ *and* $\mathcal{I}\langle a\rangle \vdash P$*, then* $\mathcal{I}\langle a\rangle^* \xrightarrow{\widetilde{\alpha}}$*;*
- *If* $\mathcal{I}\langle a\rangle^* \xrightarrow{\widetilde{\alpha}}$*, then there is* P *such that* $P \xrightarrow{\widetilde{\alpha}}$ *and* $\mathcal{I}\langle a\rangle \vdash P$*.*

Deadlock-Freedom

Theorem

$\mathcal{A}^t$ is deadlock-free if and only if the depth-first search algorithm on the right returns 'yes'.

Significance

It searches in $\mathcal{A}^*_c$'s state space but verifies $\mathcal{A}^t$'s property.

Data: $\mathcal{A}^*_c$
Output: 'yes' or 'no'

Let $bool = 1$
foreach $(P_1 \times \ldots \times P_k) \,\|\, C' \in Proc(\mathcal{A}^*_c)$ **do**
  if $P_1 \times \ldots \times P_k = F^*_c$ **then**
    if $X_{P_{c,i}} \,\|\, C' \not\longrightarrow, \exists 1 \leq i \leq k$ **then**
      $bool := 0$
      break
  else
    Let $P'_i = P_i\{\text{nil}/X_{\mathcal{I}_i\langle a\rangle}\}, \forall 1 \leq i \leq k$
    if $(P'_1 \times \ldots \times P'_k) \,\|\, C' \not\longrightarrow$ **then**
      $bool := 0$
      break

if $bool = 1$ **then return** 'yes'
else return 'no'

Non-Starvation

Data: $\mathcal{A}_c^*$
Output: 'yes' or 'no'

Let $bool = 1$
foreach $(P_1 \times \ldots \times P_k) \parallel C' \in Proc(\mathcal{A}_c^*)$ **do**
  Let $P_i' = P_i\{\mathrm{nil}/X_{\mathcal{I}_i\langle a \rangle}\}, \forall 1 \leq i \leq k$
  foreach $1 \leq i \leq k$ **do**
    Let $F_i = P_1' \times \ldots P_{i-1}' \times P_{i+1}' \times \ldots P_k'$
    if *there are* F_i' *and* C'' *such that*

$$F_i \parallel C' \xrightarrow{*} F_i' \parallel C''$$

$$P_i \parallel C'' \longrightarrow$$

    then
      skip
    else
      $bool := 0$
      break

if $bool = 1$ **then return** 'yes'
else return 'no'

Theorem

$\mathcal{A}^t$ is non-starving if and only if the depth-first search algorithm on the left returns 'yes'.

Significance

$\mathcal{A}_c^* \rightsquigarrow \mathcal{A}^t$.

Conservation (I)

Definition (Determinism)

An architecture type $\mathcal{A}^t$ is *deterministic* if its connector and all of its canonical components are deterministic.

Theorem

Suppose $\mathcal{A}^t$ is deterministic. $\mathcal{A}^t$ is conservative if and only if the depth-first search algorithm on the right returns 'yes'.

Significance

$\mathcal{A}^*_c \rightsquigarrow \mathcal{A}^t$.

Data: $\mathcal{A}^*_c$
Output: 'yes' or 'no'

Let $bool = 1$
foreach $(P_1 \times \ldots \times P_k) \,\|\, C' \in Proc(\mathcal{A}^*_c)$ **do**
  foreach α s.t. $C' \xrightarrow{\alpha}$ **do**
    if $(P_1 \times \ldots \times P_k) \not\xrightarrow{\sharp\alpha}$ **then**
      $bool := 0$
      break

if $bool = 1$ **then return** 'yes'
else return 'no'

Completeness (I)

Theorem

Suppose $\mathcal{A}^t$ is deterministic. $\mathcal{A}^t$ is complete if and only if the algorithm on the right returns 'yes'.

Data: $\mathcal{A}_c^*$
Output: 'yes' or 'no'

Let $bool = 1$
foreach $P \in Proc(P_{c,i})$, $1 \leq i \leq k$ **do**
    let $S = \{C' \mid F_c^* \parallel C \xrightarrow{*} F \parallel C', F[i] = P\}$
    if *there exists* α *s.t.* $P \xrightarrow{\alpha}$ *and* $C' \not\xrightarrow{\sharp\alpha}$ *for all* $C' \in S$ **then**
        $bool := 0$
        break
if $bool = 1$ **then return** 'yes'
else return 'no'

Significance

- $\mathcal{A}_c^* \rightsquigarrow \mathcal{A}^t$,
- Compositionality: it checks the canonical components on the one-by-one basis.

Assumptions, Definitions, Theorems

Two further *assumptions* :

- Different canonical components share no actions ($Act(P_{c,i}) \cap Act(P_{c,j}) = \varnothing$ if $i \neq j$);
- Actions of components are dual to actions of the connector, and vice versa ($\bigcup_{i=1}^{k} Act(P_{c,i}) = \{\sharp\alpha \mid \alpha \in Act(C)\}$).

Let $\widetilde{\alpha}|A$ be the projection of $\widetilde{\alpha}$ on $A \subseteq Act$. $Q \xrightarrow{\widetilde{\gamma}}_A Q'$ if and only if there is $\widetilde{\alpha}$ such that $Q \xrightarrow{\widetilde{\alpha}} Q'$ and $\widetilde{\gamma} = \widetilde{\alpha}|A$.

Definition

(1) $C \preccurlyeq P_{c,i}$ if $C \xrightarrow{\widetilde{\alpha}}_{Act(P_{c,i})}$ implies $P^*_{c,i} \xrightarrow{\sharp\widetilde{\alpha}}$; (2) $C \succcurlyeq P_{c,i}$ if $P^*_{c,i} \xrightarrow{\widetilde{\alpha}}$ implies $C \xrightarrow{\sharp\widetilde{\alpha}}_{Act(P_{c,i})}$.

Theorem

$\mathcal{A}^t$ is conservative (resp. complete) if and only if $C \preccurlyeq P_{c,i}$ (resp. $C \succcurlyeq P_{c,i}$) for each canonical component $P_{c,i}$ of $\mathcal{A}^t$.

Conservation (II)

Theorem

With the previous three assumptions, $C \preccurlyeq P_{c,i}$ if and only if the algorithm on the right returns 'yes'.

Significance

- $\mathcal{A}^*_c \rightsquigarrow \mathcal{A}^t$,
- Compositional checking.

Data: $C, P_{c,i}$
Output: 'yes' or 'no'

Construct a graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$ such that

- $\mathbf{V} = Proc(P^*_{c,i}) \times Proc(C)$
- $\mathbf{E} = \{\langle\langle P_1, C_1\rangle, \langle P_2, C_2\rangle\rangle \mid P_1 \xrightarrow{\alpha} P_2, C_1 \xrightarrow{\sharp\alpha}_{Act(P_{c,i})} C_2, \alpha \in Act(P_{c,i})\}$

Let $bool = 1$
foreach $\langle P, C'\rangle$ *reachable from* $\langle P^*_{c,i}, C\rangle$ *in* $\mathbf{G}$ **do**
  if *there is* γ *such that*
  - $P \xrightarrow{\gamma}$
  - $C' \overset{\sharp\gamma}{\not\longrightarrow}_{Act(P_{c,i})}$

  then
    $bool := 0$
    break

if $bool = 1$ **then return** 'yes'
else return 'no'

Completeness (II)

Data: $C, P_{c,i}$
Output: 'yes' or 'no'

Construct a graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$ as in the previous algorithm
Let *bool* $= 1$
foreach $\langle P, C' \rangle$ *reachable from* $\langle P^*_{c,i}, C \rangle$ *in* **G do**

  if *there is* γ *such that*

  - $P \overset{\gamma}{\not\longrightarrow}$
  - $C' \xrightarrow{\sharp\gamma}_{Act(P_{c,i})}$

  then

    bool $:= 0$
    break

if *bool* $= 1$ **then return** 'yes'
else return 'no'

Theorem

With the previous three assumptions plus the conservation of $\mathcal{A}^t$, $C \succcurlyeq P_{c,i}$ if and only if the algorithm on the left returns 'yes'.

Significance

- $\mathcal{A}^*_c \leadsto \mathcal{A}^t$,
- Finer-grained compositional checking.

Summary

- We propose a semantic model for the dynamic connector-based architecture styles;
- We show that the analysis of several basic properties of these architecture styles depends on architecture instances with fixed configurations, reducing the verification state space.

Outlook: the bridge between our semantic model and a mature ADL?