Observer-based Decomposition Assuming Known Interfaces

Bharat Garhewal
TU/e Supervisor: dr. Pieter Cuijpers
ASML Supervisor: dr. Ramon Schiffelers

Technische Universiteit Eindhoven

November 27, 2019

Introduction to decomposition

Decomposition in general

- What is decomposition?

Introduction to decomposition

Decomposition in general

- What is decomposition?
  - Breaking things down...

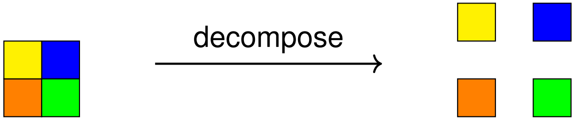


Figure 1: Decomposition

Introduction to decomposition

Decomposition in general

- What is decomposition?
  - Breaking things down...
  - ... in such a way that they can be combined back to the original!

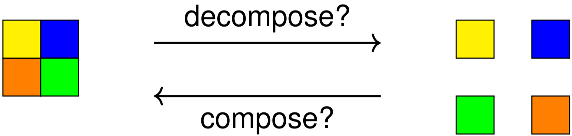


Figure 2: Decomposition

Decomposition in software systems

A software system has...

A specification $\mathcal{S}$ and a (set of) interfaces $\{I_1, \ldots, I_N\}$.

Decomposition in software systems

A software system has...

A specification $\mathcal{S}$ and a (set of) interfaces $\{I_1, \ldots, I_N\}$.

We assume...

The software system is deterministic.

Decomposition in software systems

A software system has...

A specification S and a (set of) interfaces $\{I_1, \ldots, I_N\}$.

We assume...

The software system is deterministic.

We show...

Historical behaviour influences our understanding of the decomposed system.

Decomposition in software systems

A software system has...

A specification S and a (set of) interfaces $\{I_1, \ldots, I_N\}$.

We assume...

The software system is deterministic.

We show...

Historical behaviour influences our understanding of the decomposed system.
What you remember affects your interpretation of the decomposition!

Decomposition in software systems

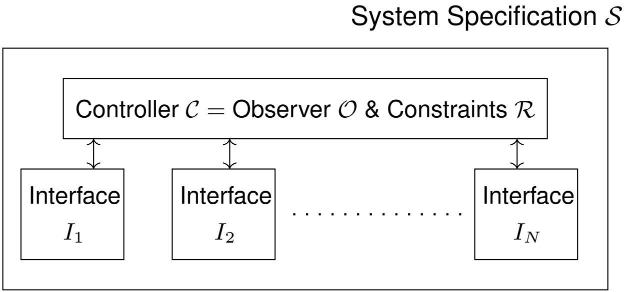
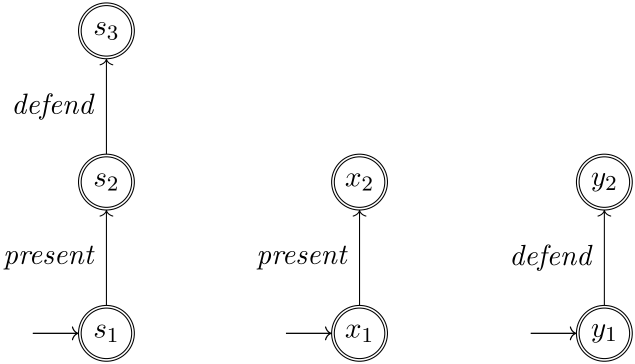


Figure 3: Internal view of a system

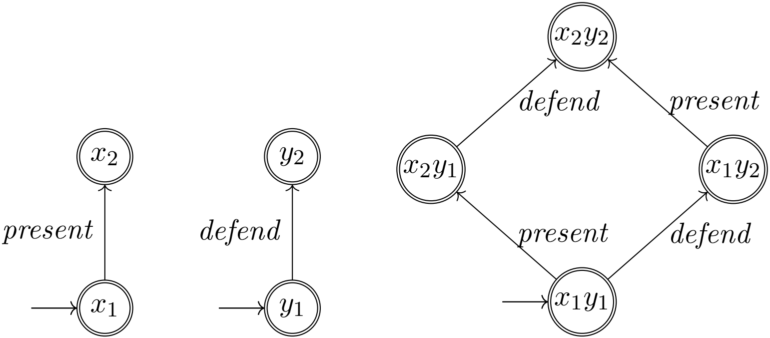
Labelled Transition System



(a) Specification $\mathcal{S}$ (b) Interface I_1 (c) Interface I_2

Figure 4: Unoriginal Example

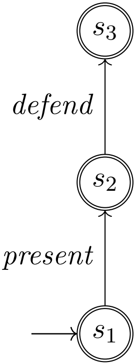
Parallel Composition



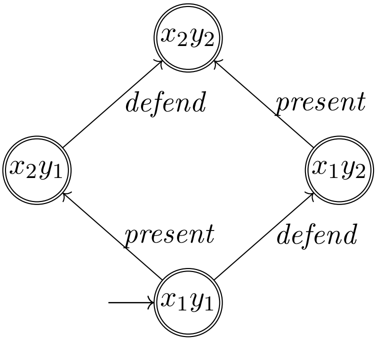
(a) Interface I_1 (b) Interface I_2 (c) Complete Interface $\mathcal{I} = I_1 \parallel_\varnothing I_2$

Figure 5: Interfaces of the Unoriginal Example

Specification and Complete Interface



(a) Specification $\mathcal{S}$



(b) Complete Interface $\mathcal{I}$

Figure 6: Unoriginal Example: Specification with a Complete Interface

Problem Statement

Given a specification $\mathcal{S}$ and set of interfaces $\{I_1, I_2, \ldots, I_N\}$, compute a controller $\mathcal{C} = (\mathcal{R}, \mathcal{O})$ such that:

$$\mathcal{S} \leftrightarrows (I_1 \parallel_\varnothing I_2 \parallel_\varnothing \ldots \parallel_\varnothing I_N) \parallel_\mathcal{R} \mathcal{O}$$

Problem Statement

Given a specification $\mathcal{S}$ and set of interfaces $\{I_1, I_2, \ldots, I_N\}$, compute a controller $\mathcal{C} = (\mathcal{R}, \mathcal{O})$ such that:

$$\mathcal{S} \Leftrightarrow (I_1 \parallel_{\varnothing} I_2 \parallel_{\varnothing} \ldots \parallel_{\varnothing} I_N) \parallel_{\mathcal{R}} \mathcal{O}$$

$$\mathcal{S} \Leftrightarrow \mathcal{I} \parallel_{\mathcal{R}} \mathcal{O}$$

Problem Statement

Given a specification $\mathcal{S}$ and set of interfaces $\{I_1, I_2, \ldots, I_N\}$, compute a controller $\mathcal{C} = (\mathcal{R}, \mathcal{O})$ such that:

$$\mathcal{S} \Leftrightarrow (I_1 \parallel_\varnothing I_2 \parallel_\varnothing \ldots \parallel_\varnothing I_N) \parallel_\mathcal{R} \mathcal{O}$$

$$\mathcal{S} \Leftrightarrow \mathcal{I} \parallel_\mathcal{R} \mathcal{O}$$

Looks similar to Supervisory Control!

Problem Statement

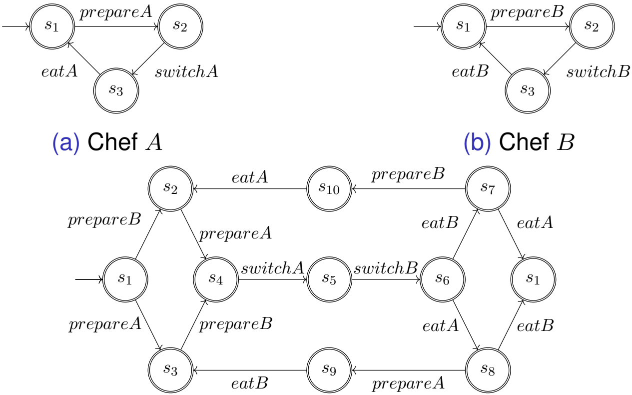
Given a specification $\mathcal{S}$ and set of interfaces $\{I_1, I_2, \ldots, I_N\}$, compute a controller $\mathcal{C} = (\mathcal{R}, \mathcal{O})$ such that:

$$\mathcal{S} \Leftrightarrow (I_1 \parallel_\varnothing I_2 \parallel_\varnothing \ldots \parallel_\varnothing I_N) \parallel_\mathcal{R} \mathcal{O}$$

$$\mathcal{S} \Leftrightarrow \mathcal{I} \parallel_\mathcal{R} \mathcal{O}$$

Looks similar to Supervisory Control!
But first, a running example.

Infinitely Hungry Chefs



(a) Chef A

(b) Chef B

(c) Kitchen $\mathcal{S} \not\approx A \parallel_{\varnothing} B$ (s_1 is repeated)

Figure 7: The stars of our kitchen

Supervisory Control Theory (SCT)

What does it do?

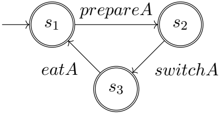
1. Plant P
2. Requirement R
3. Supervisor $S : S/P \models R$

Supervisory Control Theory (SCT)

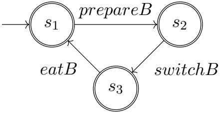
What does it do?

1. Plant $P :=$ Complete Interface $\mathcal{I}$
2. Requirement $R :=$ System Specification $\mathcal{S}$
3. Supervisor $S : S/P \models R$
   $S = P \parallel_{\varnothing} R = \mathcal{I} \parallel_{\varnothing} \mathcal{S} = \mathcal{S}$

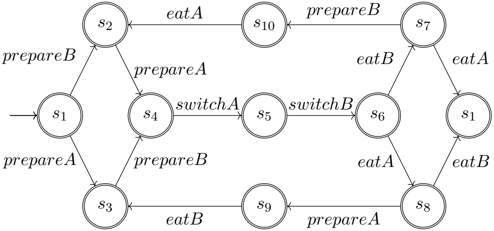
Applying SCT on the Infinitely Hungry Chefs



(a) Chef A



(b) Chef B



(c) Kitchen $\mathcal{S}$ = Supervisor S

Figure 8: Result of SCT: Supervisor is identical to the specification

SCT v/s Us

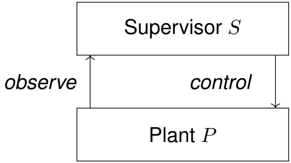


Figure 9: SCT

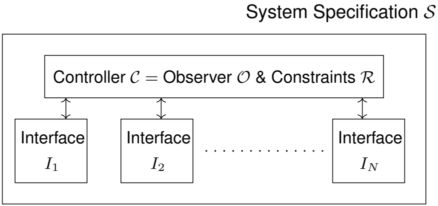
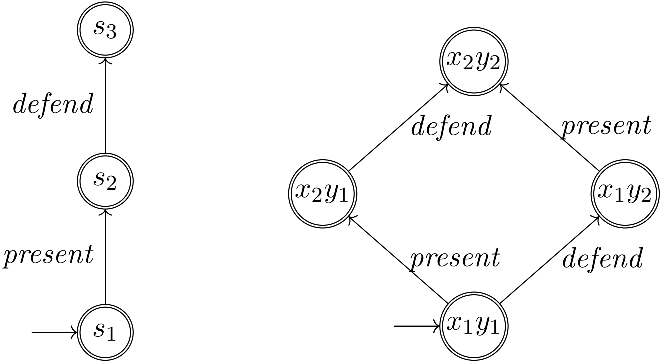


Figure 10: Our work

Specification and Complete Interface



(a) Specification $\mathcal{S}$ (b) Complete Interface $\mathcal{I}$

Figure 11: Unoriginal Example: Specification with a Complete Interface

Constraints

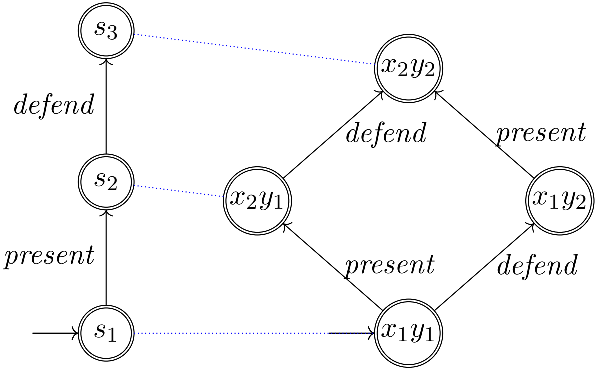


Figure 12: Unoriginal Example: Simulation Relation

Constraints

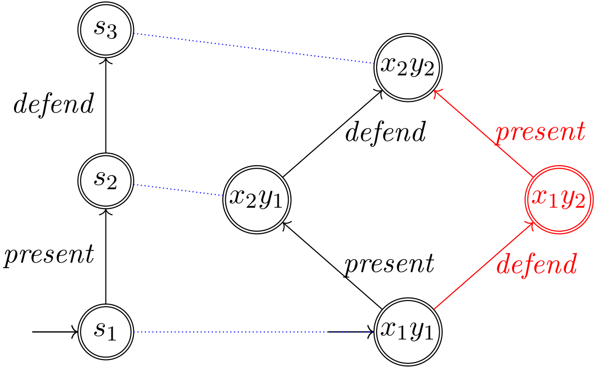


Figure 12: Unoriginal Example constrained by $\mathcal{R} = \{(defend, (x_1y_1))\}$, red is constrained, and the blue is bisimilar ($\Leftrightarrow$)

Constraints

Definition (Constrained Parallel Composition)

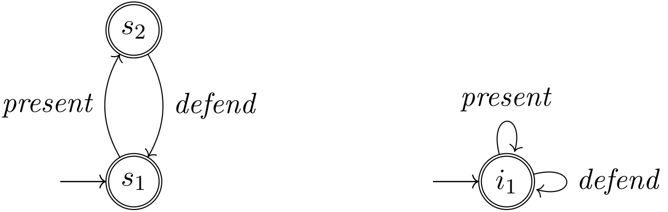
The parallel composition of LTSs $A = (Q_A, \Sigma_A, \rightarrow_A, q_{0A})$ and $B = (Q_B, \Sigma_B, \rightarrow_B, q_{0B})$ constrained by $\mathcal{R} \subseteq (\Sigma_A \cup \Sigma_B) \times Q_A \times Q_B$ is defined as:

$$A \parallel_{\mathcal{R}} B = (Q_A \times Q_B, \Sigma_A \cup \Sigma_B, \rightarrow, (q_{0A}, q_{0B})),$$

where $\rightarrow \subseteq (Q_A \times Q_B) \times (\Sigma_A \cup \Sigma_B) \times (Q_A \times Q_B)$ is the transition relation. The transition $(q_1, q_2) \xrightarrow{e} (q_1', q_2')$ is contained in $\rightarrow$ iff the transition satisfies one of the following properties:

1. $q_1 \xrightarrow{e}_A q_1'$, $q_2 \xrightarrow{e}_B q_2'$, $e \in \Sigma_A \cap \Sigma_B$, and $(e, (q_1, q_2)) \notin \mathcal{R}$, or
2. $q_1 \xrightarrow{e}_A q_1'$, $q_2 = q_2'$, $e \in \Sigma_A \setminus \Sigma_B$, and $(e, (q_1, q_2)) \notin \mathcal{R}$, or
3. $q_1 = q_1'$, $q_2 \xrightarrow{e}_B q_2'$, $e \in \Sigma_B \setminus \Sigma_A$, and $(e, (q_1, q_2)) \notin \mathcal{R}$.

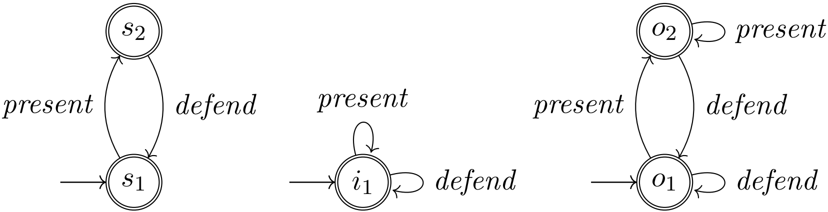
Observer



(a) Specification $\mathcal{S}$ (b) Complete Interface $\mathcal{I}$

Figure 12: Unoriginal Example: Insane version

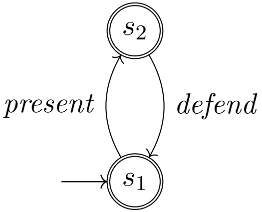
Observer



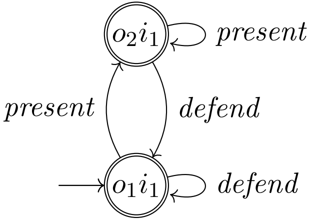
(a) Specification $\mathcal{S}$ (b) Comp. Interface $\mathcal{I}$ (c) Observer $\mathcal{O}$

Figure 12: Insanely Unoriginal Example with an observer

Observer



(a) Specification $\mathcal{S}$



(b) Composition $\mathcal{I} \parallel_{\varnothing} \mathcal{O}$

Figure 12: Insanely Unoriginal Example with a composed observer

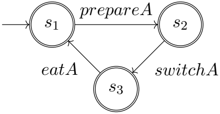
Observer

(a) Specification $\mathcal{S}$ (b) Composition $\mathcal{I} \parallel_{\varnothing} \mathcal{O}$

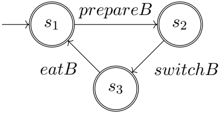
Figure 12: Insanely Unoriginal Example: $\mathcal{S} \Leftrightarrow \mathcal{I} \parallel_{\mathcal{R}} \mathcal{O}$

Constraints $\mathcal{R} = \{(defend, (i_1, o_1)), (present, (i_1, o_2))\}$

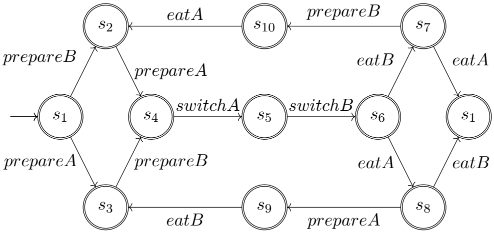
Infinitely Hungry Chefs



(a) Chef A



(b) Chef B



(c) Kitchen $\mathcal{S} \not\simeq A \parallel_{\varnothing} B$ (s_1 is repeated)

Figure 13: The stars of our kitchen

Infinitely Hungry Chefs

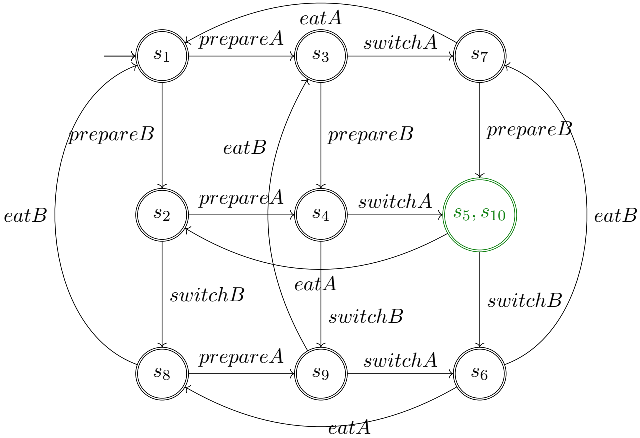


Figure 13: Complete Chefs: $\mathcal{I} = A \parallel_{\varnothing} B$

Algorithms: Introduction

What do we know?

1. We need a simulation relation f_r from specification $\mathcal{S}$ to complete interface $\mathcal{I}$.
2. If f_r is injective, then $\mathcal{O} = 1_{\Sigma_\mathcal{I}}$ (single state with self-loops).
3. If f_r is non-injective, then make it injective using an observer $\mathcal{O}$.
4. Compute constraints $\mathcal{R}$ using f_r, $\mathcal{S}$, $\mathcal{I}$, and $\mathcal{O}$.
5. Finally, $\mathcal{S} \Leftrightarrow \mathcal{I} \parallel_\mathcal{R} \mathcal{O}$.

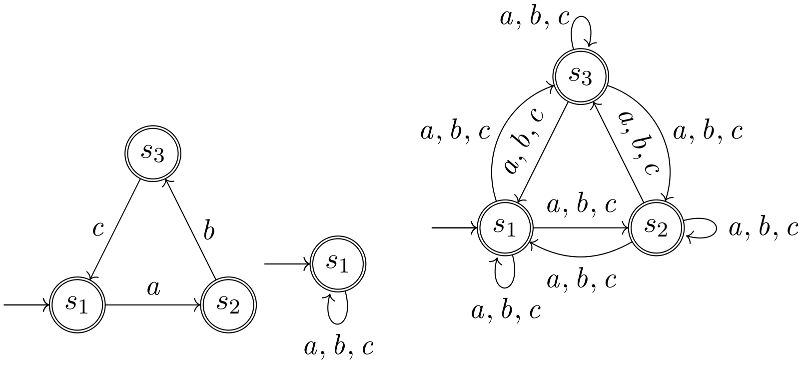
Find Observer

Definition (Observer)

An observer $\mathcal{O}$ for a specification $\mathcal{S}$ and complete interface $\mathcal{I}$ is an LTS such that the following properties hold:

1. $\mathcal{O}$ should observe all events from $\mathcal{S}$,
2. $\mathcal{O}$ should be input-enabled,
3. There exists an injective simulation relation $f_r \subseteq Q_{\mathcal{S}} \times Q_{\mathcal{I}\|_{\varnothing}\mathcal{O}}$.

Find Observer



(a) Specification $\mathcal{S}$ (b) Flower $1_{\mathcal{S}}$ (c) Exploded Flower $\clubsuit_{\mathcal{S}}$

Figure 14: Exploding a flower

Find Observer

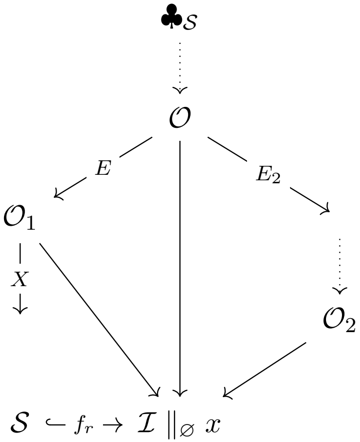


Figure 14: Locally Minimal observer

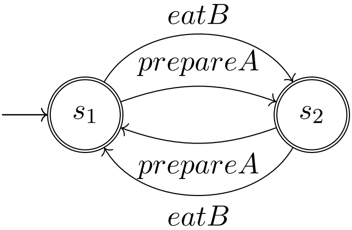
Find Observer

Algorithm 1: `FindObserver`

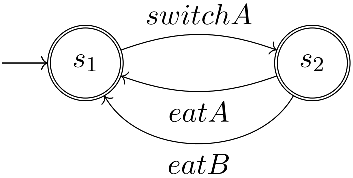
Data: Pre-observer $\mathcal{O} = \clubsuit_{\mathcal{S}}$, Complete Interface $\mathcal{I}$, Specification $\mathcal{S}$
Result: Observer $\mathcal{O}$

if $f_r \subseteq Q_{\mathcal{S}} \times Q_{\mathcal{I}\|_{\varnothing}\mathcal{O}}$ *doesn't exist and* $\mathcal{O} \neq \clubsuit_{\mathcal{S}}$ **then**
  return $\perp$
if $\mathcal{O} \cong 1_{\mathcal{O}}$ **then**
  return $\mathcal{O}$
for $E \in \mathcal{P}(\rightarrow_{\mathcal{O}})$ **do**
  $\mathcal{O}' := \mathcal{O}$
  $\mathcal{O} :=$ `FindObserver`$(delTransitions(\mathcal{O}, E), \mathcal{I}, \mathcal{S})$
  if $\mathcal{O}$ *is a DLTS* **then**
    return $\mathcal{O}$
  else
    $\mathcal{O} := \mathcal{O}'$
  end
end

Observers for the Infinitely Hungry Chefs



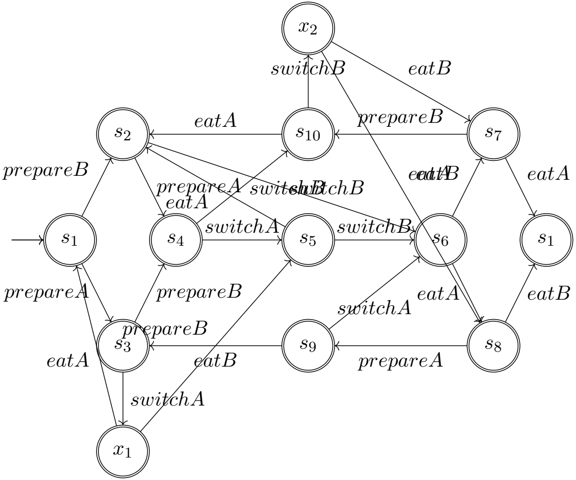
(a) Observer $\mathcal{O}_1$



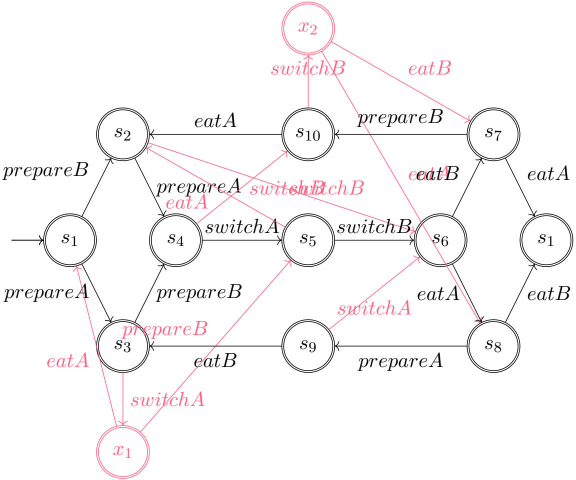
(b) Observer $\mathcal{O}_2$

Figure 14: Two locally minimal observers (self-loops omitted)

Constraints for observer $\mathcal{O}_2$



Constraints for observer $\mathcal{O}_2$



Make Constraints

Algorithm 2: `FindConstraints`

Data: Specification $\mathcal{S}$, Complete Interface $\mathcal{I}$, Observer $\mathcal{O}$, Injective simulation relation $f_r \subseteq \mathcal{S} \times \mathcal{I} \parallel_{\varnothing} \mathcal{O}$, projections $\pi_{\mathcal{I}} \subseteq Q_{\mathcal{I}} \times Q_{\mathcal{O}} \times Q_{\mathcal{I}}$ and $\pi_{\mathcal{O}} \subseteq Q_{\mathcal{I}} \times Q_{\mathcal{O}} \times Q_{\mathcal{O}}$

Result: Set of constraints $\mathcal{R}$

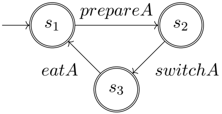
Create empty set B of elements of type $Q_{\mathcal{I} \parallel_{\varnothing} \mathcal{O}} \times \Sigma_{\mathcal{I}}$

for $(s, p) : f_r$ **do**

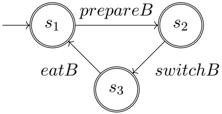
$\quad B(p) := \{e \in \Sigma_{\mathcal{I}} \mid t \xrightarrow{e}_{\mathcal{I} \parallel_{\varnothing} \mathcal{O}} \textit{ and } s \not\xrightarrow{e}_{\mathcal{S}}\}$

end

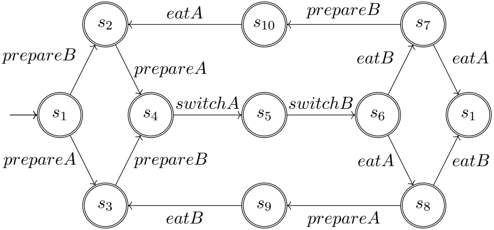
$\mathcal{R} = \{(e, g(i)) \mid (e, i) \in B^{-1}\}$ where $g(i) = \{(\pi_{\mathcal{I}}(x), \pi_{\mathcal{O}}(x)) | x \in i\}$



(a) Chef A



(b) Chef B



(c) Kitchen $\mathcal{S} \not\cong A \parallel_{\varnothing} B$ (s_1 is repeated)

Figure 16: The stars of our kitchen

Solution 1

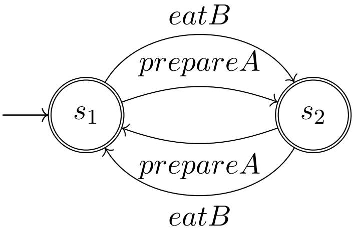


Figure 17: Observer $\mathcal{O}_1$

$$\begin{aligned}
&(eatA, (\mathcal{O}._1 s_2, A.s_3, B.s_2)) \\
&(swtichB, \{(\mathcal{O}_1.s_1, A.s_3, B.s_2), \\
&\qquad (\mathcal{O}_1.s_1, A.s_1, B.s_2), \\
&\qquad (\mathcal{O}_1.s_2, A.s_2, B.s_2)\}) \\
&(switchA, \{(\mathcal{O}_1.s_2, A.s_2, B.s_1), \\
&\qquad (\mathcal{O}_1.s_1, A.s_2, B.s_3)\})
\end{aligned}$$

$R_{\mathcal{O}_1}$

Solution 1

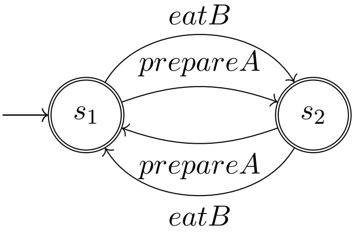


Figure 17: Observer $\mathcal{O}_1$

$$
\begin{aligned}
&(eatA, (\mathcal{O}._1 s_2, A.s_3, B.s_2)) \\
&(swtichB, \{(\mathcal{O}_1.s_1, A.s_3, B.s_2), \\
&\qquad (\mathcal{O}_1.s_1, A.s_1, B.s_2), \\
&\qquad (\mathcal{O}_1.s_2, A.s_2, B.s_2)\}) \\
&(switchA, \{(\mathcal{O}_1.s_2, A.s_2, B.s_1), \\
&\qquad (\mathcal{O}_1.s_1, A.s_2, B.s_3)\})
\end{aligned}
$$

$R_{\mathcal{O}_1}$

1. Chef A cannot switch before she is prepared ($\mathcal{O}_1.s_1$) and before chef B has eaten ($B.s_3$) after switching, and
2. Chef A cannot switch before she is prepared ($\mathcal{O}_1.s_2$) and before chef B has eaten and while chef B is prepared ($B.s_1$).

Solution 2

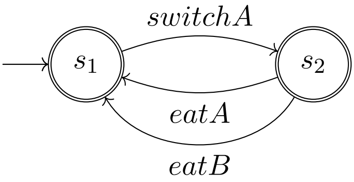


Figure 18: Observer $\mathcal{O}_2$

$$\begin{aligned}
&(eatA, (\mathcal{O}_2.s_2, A.s_3, B.s_2))\\
&(swtichB, \{(\mathcal{O}_2.s_1, A.s_1, B.s_2),\\
&\quad(\mathcal{O}_2.s_1, A.s_2, B.s_2),\\
&\quad(\mathcal{O}_2.s_1, A.s_3, B.s_2)\})\\
&(switchA, \{(\mathcal{O}_2.s_1, A.s_2, B.s_1),\\
&\quad(\mathcal{O}_2.s_1, A.s_2, B.s_3)\})\\
&R_{\mathcal{O}_2}
\end{aligned}$$

Solution 2

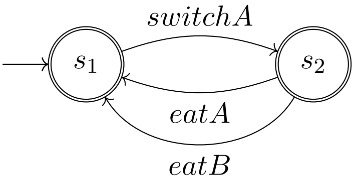


Figure 18: Observer $\mathcal{O}_2$

$$
\begin{aligned}
&(eatA, (\mathcal{O}_2.s_2, A.s_3, B.s_2))\\
&(swtichB, \{(\mathcal{O}_2.s_1, A.s_1, B.s_2),\\
&\qquad (\mathcal{O}_2.s_1, A.s_2, B.s_2),\\
&\qquad (\mathcal{O}_2.s_1, A.s_3, B.s_2)\})\\
&(switchA, \{(\mathcal{O}_2.s_1, A.s_2, B.s_1),\\
&\qquad (\mathcal{O}_2.s_1, A.s_2, B.s_3)\})
\end{aligned}
$$

$$R_{\mathcal{O}_2}$$

1. Chef A cannot switch when chef B is not prepared ($B.s_1$) or when chef B has already switched his dish ($B.s_3$).

Which one is more 'intuitive'?

Solution 1

1. Chef A cannot switch before she is prepared ($\mathcal{O}_1.s_1$) and before chef B has eaten ($B.s_3$) after switching, and
2. Chef A cannot switch before she is prepared ($\mathcal{O}_1.s_2$) and before chef B has eaten and while chef B is prepared ($B.s_1$).

Solution 2

1. Chef A cannot switch when chef B is not prepared ($B.s_1$) or when chef B has already switched his dish ($B.s_3$).

Which one is more 'intuitive'?

Solution 2

1. Chef A cannot switch when chef B is not prepared $(B.s_1)$ or when chef B has already switched his dish $(B.s_3)$.

Original

1. Chef A can switch her dish only after both the chefs have finished preparing their dishes,
2. Chef B catch his dish only after chef A has switched his dish.

Future Work

Intuitive

More 'intuitive' or more 'interesting'?

Mechanical

Construct hierarchical observers?

Supervisory Control Theory

Perhaps some techniques in SCT can help us out?

Wrap-up

A software system has...

A specification S and a (set of) interfaces $\{I_1, \ldots, I_N\}$.

We assume...

The software system is deterministic.

We show...

What you remember influences your interpretation of the decomposition!