A Framework for the Design and Verification of Component-Based Systems

Mila Majster-Cederbaum

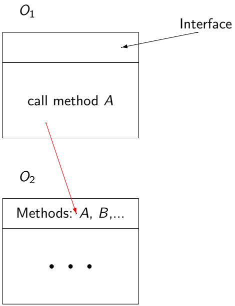
Institute of Computer Science, University of Mannheim, Germany

joint work with G. Gössler, S. Graf, M. Martens, and J. Sifakis

January 21, 2007

System Views

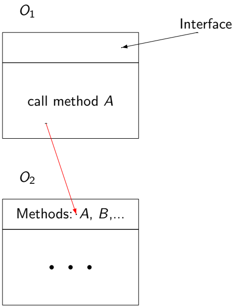
Object-Oriented



O_1 depends on
the existence of O_2

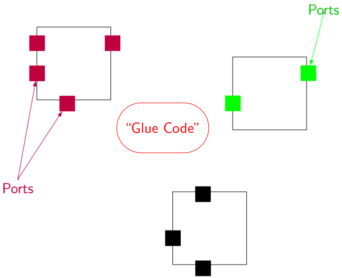
System Views

Object-Oriented



O_1 depends on the existence of O_2

Component-Based



Components do not refer to other components. They offer ports and may be glued together.

Many approaches consider a component as a “black box”.

Many approaches consider a component as a “black box”.

If we want to study properties of component-based systems more information is needed.

Interaction Systems

Interaction Systems

- *Each component* is given by:

Interaction Systems

- *Each component* is given by: a "frame"

Interaction Systems

- *Each component* is given by: a “frame” + “local behavior”.

Interaction Systems

- *Each component* is given by: a "frame" + "local behavior".
- *The glue* is modelled via "connectors".

Interaction Systems

- *Each component* is given by: a "frame" + "local behavior".
- *The glue* is modelled via "connectors".

This means: there are three independent description levels.

Static ingredients of our Model

- a set K of components, w.l.o.g. $K = \{1, ..., n\}$

Static ingredients of our Model

- a set K of components, w.l.o.g. $K = \{1, ..., n\}$
- each component $i \in K$ has a set A_i of ports (actions), $A_i \cap A_j = \emptyset$ for $i \neq j$

Static ingredients of our Model

- a set K of components, w.l.o.g. $K = \{1, ..., n\}$
- each component $i \in K$ has a set A_i of ports (actions), $A_i \cap A_j = \emptyset$ for $i \neq j$
- a connector c is a finite nonempty set of ports, where no two ports belong to the same component, e.g. $c = \{a_1, a_2, a_3\}$, $a_i \in A_i$. A connector designates actions that should be performed conjointly.

Static ingredients of our Model

- a set K of components, w.l.o.g. $K = \{1, ..., n\}$
- each component $i \in K$ has a set A_i of ports (actions), $A_i \cap A_j = \emptyset$ for $i \neq j$
- a connector c is a finite nonempty set of ports, where no two ports belong to the same component, e.g. $c = \{a_1, a_2, a_3\}$, $a_i \in A_i$. A connector designates actions that should be performed conjointly.
- if $\emptyset \neq \alpha \subseteq c$, α is called an interaction. If $a_i \in A_i \cap \alpha$, we say that i participates in α and put $i(\alpha) = a_i$.

Static ingredients of our Model

- a set K of components, w.l.o.g. $K = \{1, ..., n\}$
- each component $i \in K$ has a set A_i of ports (actions), $A_i \cap A_j = \emptyset$ for $i \neq j$
- a connector c is a finite nonempty set of ports, where no two ports belong to the same component, e.g. $c = \{a_1, a_2, a_3\}$, $a_i \in A_i$. A connector designates actions that should be performed conjointly.
- if $\emptyset \neq \alpha \subseteq c$, α is called an interaction. If $a_i \in A_i \cap \alpha$, we say that i participates in α and put $i(\alpha) = a_i$.
- a connector set $C = \{c_1, c_2, \ldots\}$ such that
  1) $c_i \nsubseteq c_j$ 2) $\bigcup_{c \in C} c = \bigcup_{i \in K} A_i$

Static ingredients of our Model

- a set K of components, w.l.o.g. $K = \{1, ..., n\}$
- each component $i \in K$ has a set A_i of ports (actions), $A_i \cap A_j = \emptyset$ for $i \neq j$
- a connector c is a finite nonempty set of ports, where no two ports belong to the same component, e.g. $c = \{a_1, a_2, a_3\}$, $a_i \in A_i$. A connector designates actions that should be performed conjointly.
- if $\emptyset \neq \alpha \subseteq c$, α is called an interaction. If $a_i \in A_i \cap \alpha$, we say that i participates in α and put $i(\alpha) = a_i$.
- a connector set $C = \{c_1, c_2, \ldots\}$ such that
  1) $c_i \not\subseteq c_j$ 2) $\bigcup_{c \in C} c = \bigcup_{i \in K} A_i$
- connectors are also referred to as maximal interactions

Static ingredients of our Model

- a set K of components, w.l.o.g. $K = \{1, ..., n\}$
- each component $i \in K$ has a set A_i of ports (actions), $A_i \cap A_j = \emptyset$ for $i \neq j$
- a connector c is a finite nonempty set of ports, where no two ports belong to the same component, e.g. $c = \{a_1, a_2, a_3\}$, $a_i \in A_i$. A connector designates actions that should be performed conjointly.
- if $\emptyset \neq \alpha \subseteq c$, α is called an interaction. If $a_i \in A_i \cap \alpha$, we say that i participates in α and put $i(\alpha) = a_i$.
- a connector set $C = \{c_1, c_2, \ldots\}$ such that
  1) $c_i \not\subseteq c_j$ 2) $\bigcup_{c \in C} c = \bigcup_{i \in K} A_i$
- connectors are also referred to as maximal interactions
- a set *Comp* of interactions α that are called complete. If $\alpha \subset c$ is complete then α may proceed no matter if the missing actions of c are available or not.

Example - The Dining Philosophers

We model the problem of n philosophers.

Example - The Dining Philosophers

We model the problem of n philosophers.
There are the following types of components:

Example - The Dining Philosophers

We model the problem of n philosophers.
There are the following types of components:

- n components p_i for $0 \leq i \leq n-1$ representing the philosophers. The ports for p_i are $\{activate_i, enter_i, get_i^i, get_i^{i+1}, eat_i, put_i^i, put_i^{i+1}, leave_i\}$.

Example - The Dining Philosophers

We model the problem of n philosophers.
There are the following types of components:

- n components p_i for $0 \leq i \leq n-1$ representing the philosophers. The ports for p_i are $\{activate_i, enter_i, get_i^i, get_i^{i+1}, eat_i, put_i^i, put_i^{i+1}, leave_i\}$.
- n components f_i for $0 \leq i \leq n-1$ representing the forks. The ports for f_i are $\{get_i, put_i\}$.

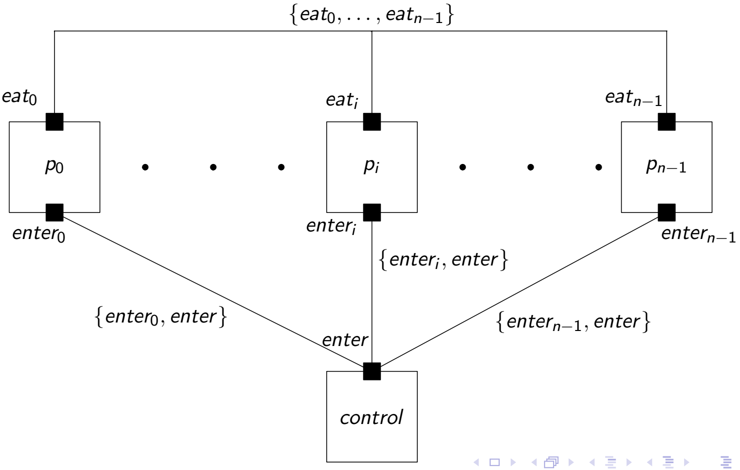
Example - The Dining Philosophers

We model the problem of n philosophers.
There are the following types of components:

- n components p_i for $0 \leq i \leq n-1$ representing the philosophers. The ports for p_i are $\{activate_i, enter_i, get_i^i, get_i^{i+1}, eat_i, put_i^i, put_i^{i+1}, leave_i\}$.
- n components f_i for $0 \leq i \leq n-1$ representing the forks. The ports for f_i are $\{get_i, put_i\}$.
- One component *control*. It controls when a philosopher may enter the room in which the table is located. Its ports are $\{enter, leave\}$.

Example - The Dining Philosophers, Static View

Part of the picture : the philosophers, the control and some connectors. Any nonempty subset of $\{eat_0, eat_1, ..., eat_{n-1}\}$ is declared complete.



Dynamics of our Model

- each component i has a local behavior given by a transition system $T_i = (Q_i, \rightarrow_i)$ where $\rightarrow_i \subseteq Q_i \times A_i \times Q_i$ and A_i is the (local) port set of i. It is assumed that every state offers some action.

Dynamics of our Model

- each component i has a local behavior given by a transition system $T_i = (Q_i, \rightarrow_i)$ where $\rightarrow_i \subseteq Q_i \times A_i \times Q_i$ and A_i is the (local) port set of i. It is assumed that every state offers some action.
- the behavior of the global system is then

$$T = \left(\underbrace{Q_1 \times Q_2 \times \ldots Q_n}_{Q}, \rightarrow \right)$$

with

$$q = (q_1, q_2, \ldots) \overset{\alpha}{\rightarrow} q' = (q'_1, q'_2, \ldots)$$

where α is an interaction and

Dynamics of our Model

- each component i has a local behavior given by a transition system $T_i = (Q_i, \rightarrow_i)$ where $\rightarrow_i \subseteq Q_i \times A_i \times Q_i$ and A_i is the (local) port set of i. It is assumed that every state offers some action.
- the behavior of the global system is then

$$T = \left(\underbrace{Q_1 \times Q_2 \times \ldots Q_n}_{Q}, \rightarrow \right)$$

with

$$q = (q_1, q_2, \ldots) \xrightarrow{\alpha} q' = (q'_1, q'_2, \ldots)$$

where α is an interaction and

- $q_i = q'_i$ if component i does not participate in α

Dynamics of our Model

- each component i has a local behavior given by a transition system $T_i = (Q_i, \rightarrow_i)$ where $\rightarrow_i \subseteq Q_i \times A_i \times Q_i$ and A_i is the (local) port set of i. It is assumed that every state offers some action.
- the behavior of the global system is then

$$T = \left(\underbrace{Q_1 \times Q_2 \times \ldots Q_n}_{Q}, \rightarrow \right)$$

with

$$q = (q_1, q_2, \ldots) \xrightarrow{\alpha} q' = (q'_1, q'_2, \ldots)$$

where α is an interaction and

  - $q_i = q'_i$ if component i does not participate in α
  - $q_i \xrightarrow{a_i} q'_i$ if $a_i \in \alpha$

Definition Interaction Systems

An **Interaction System** is given by

$$Sys = (K, C, Comp, T)$$

where

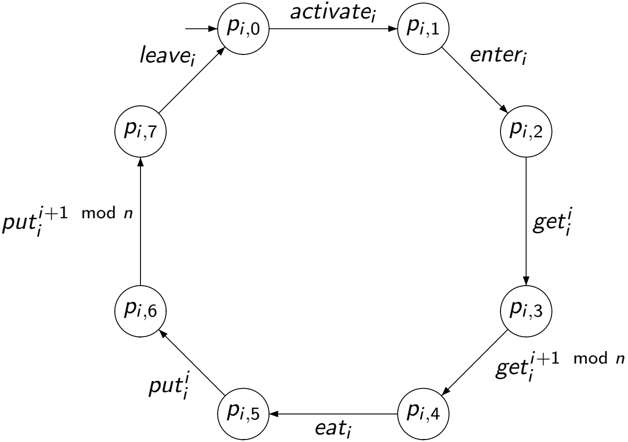
K, C, and *Comp* constitute the static part of the system

and

T constitutes the dynamic part of the system.

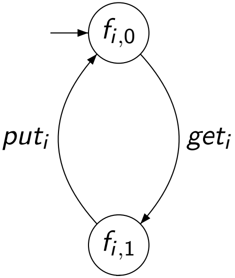
Example - The Dining Philosophers, Dynamics

The behavior of philosopher p_i is given by:



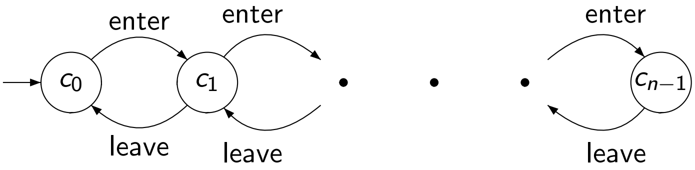
Example - The Dining Philosophers, Dynamics

The behavior of fork f_i is given by:



Example - The Dining Philosophers, Dynamics

The behavior of *control* is given by:



Example - The Dining Philosophers, Connectors

There are the following connectors:

- $\{eat_0, \ldots, eat_{n-1}\}$ and any nonempty subset is complete

Example - The Dining Philosophers, Connectors

There are the following connectors:

- $\{eat_0, \ldots, eat_{n-1}\}$ and any nonempty subset is complete
- $\{activate_0, \ldots, activate_{n-1}\}$

Example - The Dining Philosophers, Connectors

There are the following connectors:

- $\{eat_0, \ldots, eat_{n-1}\}$ and any nonempty subset is complete
- $\{activate_0, \ldots, activate_{n-1}\}$
- $\{enter, enter_i\}$

Example - The Dining Philosophers, Connectors

There are the following connectors:

- $\{eat_0, \ldots, eat_{n-1}\}$ and any nonempty subset is complete
- $\{activate_0, \ldots, activate_{n-1}\}$
- $\{enter, enter_i\}$
- $\{leave, leave_i\}$

Example - The Dining Philosophers, Connectors

There are the following connectors:

- $\{eat_0, \ldots, eat_{n-1}\}$ and any nonempty subset is complete
- $\{activate_0, \ldots, activate_{n-1}\}$
- $\{enter, enter_i\}$
- $\{leave, leave_i\}$
- $\{get_i^i, get_i\}$

Example - The Dining Philosophers, Connectors

There are the following connectors:

- $\{eat_0, \ldots, eat_{n-1}\}$ and any nonempty subset is complete
- $\{activate_0, \ldots, activate_{n-1}\}$
- $\{enter, enter_i\}$
- $\{leave, leave_i\}$
- $\{get_i^i, get_i\}$
- $\{put_i^i, put_i\}$

Example - The Dining Philosophers, Connectors

There are the following connectors:

- $\{eat_0, \dots, eat_{n-1}\}$ and any nonempty subset is complete
- $\{activate_0, \dots, activate_{n-1}\}$
- $\{enter, enter_i\}$
- $\{leave, leave_i\}$
- $\{get_i^i, get_i\}$
- $\{put_i^i, put_i\}$
- $\left\{get_i^{i+1 \mod n}, get_{i+1 \mod n}\right\}$

Example - The Dining Philosophers, Connectors

There are the following connectors:

- $\{eat_0, \ldots, eat_{n-1}\}$ and any nonempty subset is complete
- $\{activate_0, \ldots, activate_{n-1}\}$
- $\{enter, enter_i\}$
- $\{leave, leave_i\}$
- $\{get_i^i, get_i\}$
- $\{put_i^i, put_i\}$
- $\left\{get_i^{i+1 \mod n}, get_{i+1 \mod n}\right\}$
- $\left\{put_i^{i+1 \mod n}, put_{i+1 \mod n}\right\}$

Example - The Dining Philosophers, Connectors

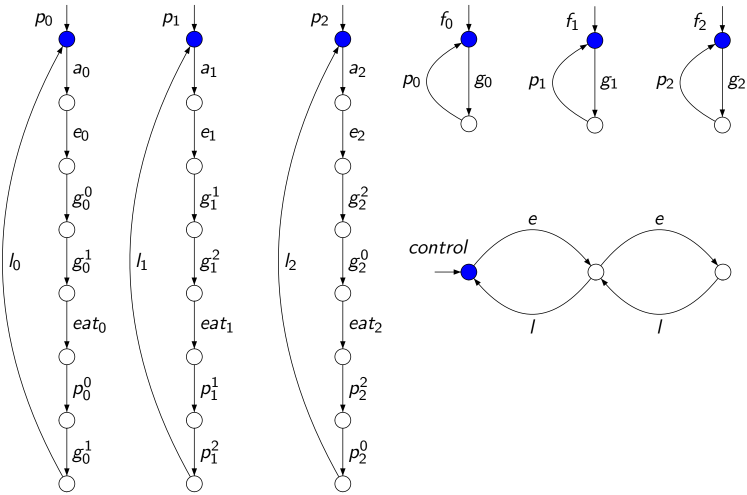
There are the following connectors:

- $\{eat_0, \ldots, eat_{n-1}\}$ and any nonempty subset is complete
- $\{activate_0, \ldots, activate_{n-1}\}$
- $\{enter, enter_i\}$
- $\{leave, leave_i\}$
- $\{get_i^i, get_i\}$
- $\{put_i^i, put_i\}$
- $\left\{get_i^{i+1 \mod n}, get_{i+1 \mod n}\right\}$
- $\left\{put_i^{i+1 \mod n}, put_{i+1 \mod n}\right\}$

for $0 \leq i \leq n-1$

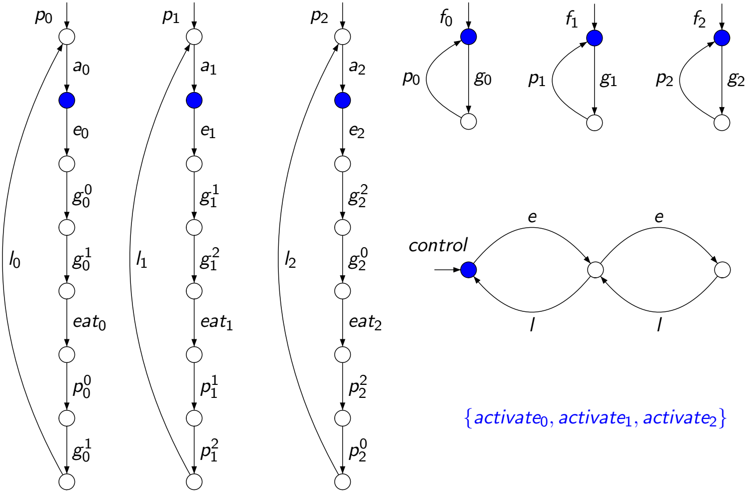
Example - The Dining Philosophers, Global Transitions

The behavior for $n = 3$:



Example - The Dining Philosophers

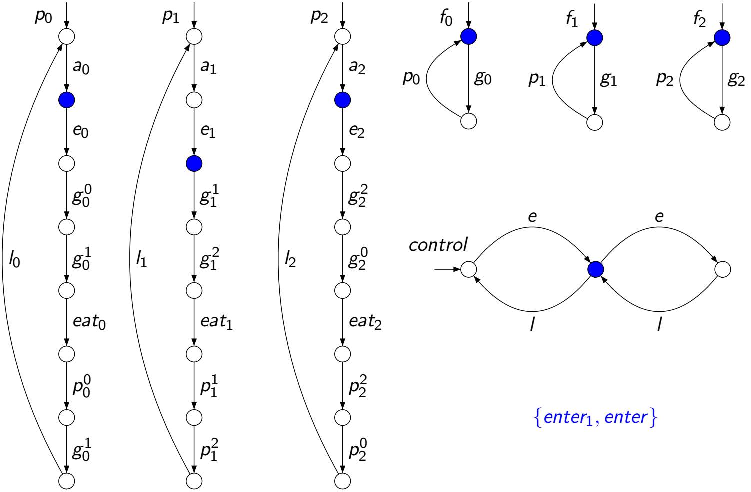
The behavior for $n = 3$:



$\{activate_0, activate_1, activate_2\}$

Example - The Dining Philosophers

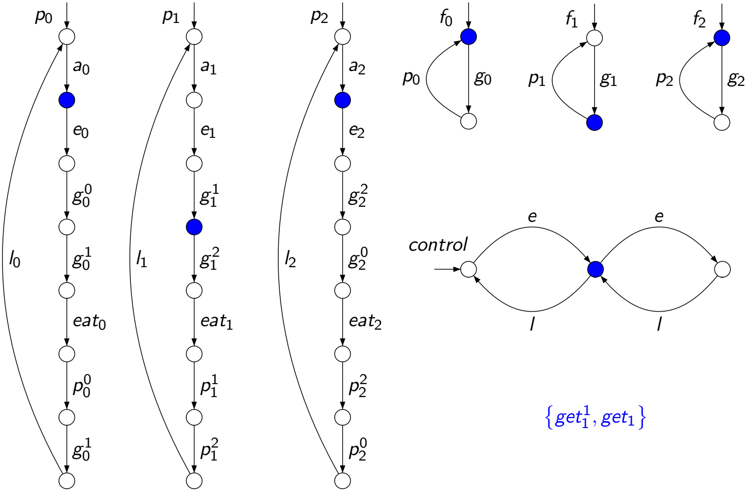
The behavior for $n = 3$:



$\{enter_1, enter\}$

Example - The Dining Philosophers

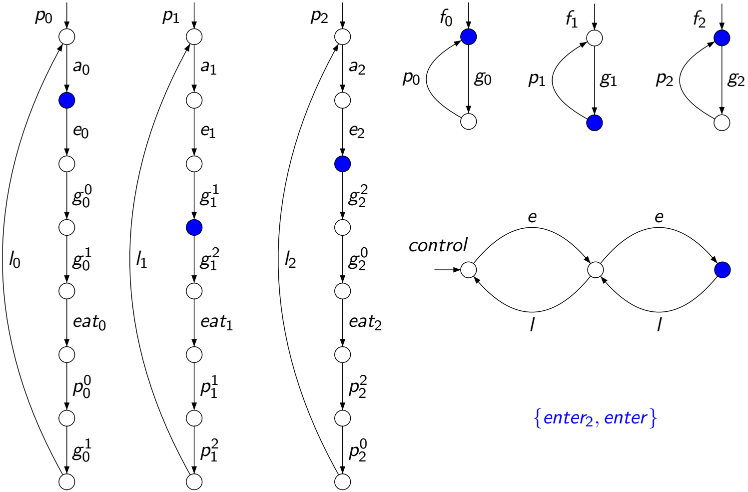
The behavior for $n = 3$:



$\{get_1^1, get_1\}$

Example - The Dining Philosophers

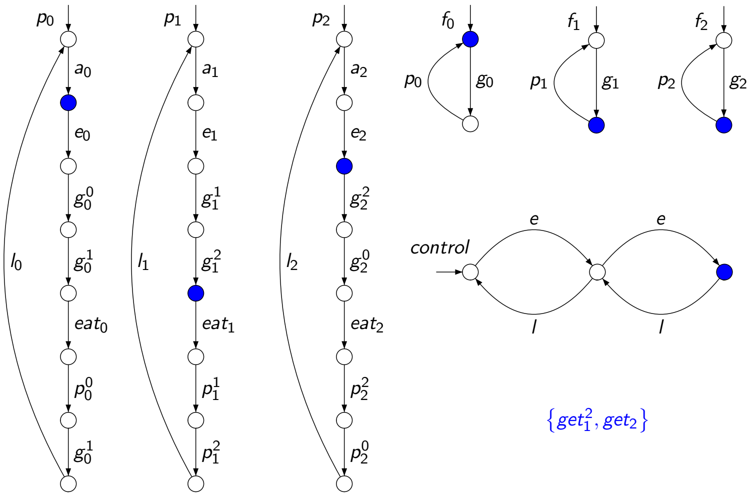
The behavior for $n = 3$:



$\{enter_2, enter\}$

Example - The Dining Philosophers

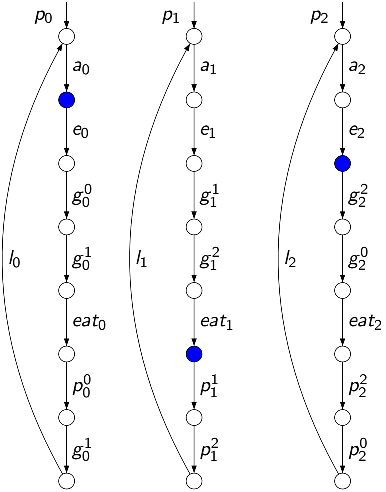
The behavior for $n = 3$:

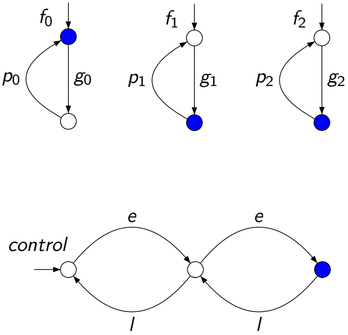


$\{get_1^2, get_2\}$

Example - The Dining Philosophers

The behavior for $n = 3$:





$\{eat_1\}$ and so on

Properties of Interaction Systems

Properties of Interaction Systems

Interesting properties of interaction systems are

Properties of Interaction Systems

Interesting properties of interaction systems are

- local/global deadlock-freedom

Properties of Interaction Systems

Interesting properties of interaction systems are

- local/global deadlock-freedom
- liveness of components

Properties of Interaction Systems

Interesting properties of interaction systems are

- local/global deadlock-freedom
- liveness of components
- progress of components

Properties of Interaction Systems

Interesting properties of interaction systems are

- local/global deadlock-freedom
- liveness of components
- progress of components
- robustness against failure of components

Properties of Interaction Systems

Interesting properties of interaction systems are

- local/global deadlock-freedom
- liveness of components
- progress of components
- robustness against failure of components
- availability

Properties of Interaction Systems

Interesting properties of interaction systems are

- local/global deadlock-freedom
- liveness of components
- progress of components
- robustness against failure of components
- availability

Here we treat liveness.

P-Deadlock-Freedom

A predicate P on the state space Q is inductive if

1. $P \not\equiv \mathit{false}$
2. $P(q) \wedge q \xrightarrow{\alpha} q' \Rightarrow P(q')$ for $\alpha \in C \cup \mathit{Comp}$

P-Deadlock-Freedom

A predicate P on the state space Q is inductive if

1. $P \not\equiv \mathit{false}$
2. $P(q) \wedge q \xrightarrow{\alpha} q' \Rightarrow P(q')$ for $\alpha \in C \cup \mathit{Comp}$

From now on let P be an inductive predicate on Q.

P-Deadlock-Freedom

A predicate P on the state space Q is inductive if

1. $P \not\equiv \mathit{false}$
2. $P(q) \wedge q \xrightarrow{\alpha} q' \Rightarrow P(q')$ for $\alpha \in C \cup \mathit{Comp}$

From now on let P be an inductive predicate on Q.

Sys is called P-deadlock-free if for every global state q with $P(q) = \mathit{true}$ there is a transition

$$q \xrightarrow{\alpha} q'$$

with $\alpha \in C \cup \mathit{Comp}$.

Liveness

Let $Sys = (K, C, Comp, T)$ be a P-deadlock-free interaction system.

Liveness

Let $Sys = (K, C, Comp, T)$ be a P-deadlock-free interaction system.
A run is an infinite transition sequence

$$\sigma := q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \xrightarrow{\alpha_2} \ldots \qquad \alpha_i \in C \cup Comp.$$

Liveness

Let $Sys = (K, C, Comp, T)$ be a P-deadlock-free interaction system.
A run is an infinite transition sequence

$$\sigma := q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \xrightarrow{\alpha_2} \ldots \qquad \alpha_i \in C \cup Comp.$$

$K' \subseteq K$ is called is called P-live if *every run* with $P(q_0)$

$$\sigma := q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \xrightarrow{\alpha_2} \ldots \qquad \alpha_i \in C \cup Comp$$

of Sys encompasses an infinite number of transitions labelled with an interaction where some $i \in K'$ participates.

NP-Hardness

Testing properties is expensive ("state-space-explosion").

NP-Hardness

Testing properties is expensive ("state-space-explosion").

We have shown e.g. that deciding

NP-Hardness

Testing properties is expensive ("state-space-explosion").

We have shown e.g. that deciding

- deadlock-freedom

NP-Hardness

Testing properties is expensive ("state-space-explosion").

We have shown e.g. that deciding

- deadlock-freedom
- liveness

NP-Hardness

Testing properties is expensive ("state-space-explosion").

We have shown e.g. that deciding

- deadlock-freedom
- liveness

is NP-hard.

NP-Hardness

Testing properties is expensive (“state-space-explosion”).

We have shown e.g. that deciding

- deadlock-freedom
- liveness

is NP-hard.

Proposed solutions:

NP-Hardness

Testing properties is expensive ("state-space-explosion").

We have shown e.g. that deciding

- deadlock-freedom
- liveness

is NP-hard.

Proposed solutions:

- *establish conditions* that can be tested in polynomial time and imply the desired properties

NP-Hardness

Testing properties is expensive ("state-space-explosion").

We have shown e.g. that deciding

- deadlock-freedom
- liveness

is NP-hard.

Proposed solutions:

- *establish conditions* that can be tested in polynomial time and imply the desired properties
- exploit compositionality

A Sufficient Criterion for Liveness

Let Sys be an interaction system with set K of components with alphabets A_i, where $i \in K$.

A Sufficient Criterion for Liveness

Let Sys be an interaction system with set K of components with alphabets A_i, where $i \in K$. $A \subset A_j$ is inevitable in T_j if every infinite path in T_j encompasses an infinite number of transitions labelled with some action in A.

A Sufficient Criterion for Liveness

Let Sys be an interaction system with set K of components with alphabets A_i, where $i \in K$. $A \subset A_j$ is inevitable in T_j if every infinite path in T_j encompasses an infinite number of transitions labelled with some action in A. Let

$$G = (K, \rightarrow)$$

A Sufficient Criterion for Liveness

Let *Sys* be an interaction system with set K of components with alphabets A_i, where $i \in K$. $A \subset A_j$ is inevitable in T_j if every infinite path in T_j encompasses an infinite number of transitions labelled with some action in A. Let

$$G = (K, \rightarrow)$$

where

$$i \rightarrow j \Leftrightarrow A_j \setminus \underbrace{excl\,(i)\,[j]}_{\substack{\text{all } a_j \text{ that occur} \\ \text{in some } \alpha \\ \text{in which } i \text{ does} \\ \text{does not participate}}} \text{ is inevitable in } \underbrace{T_j}_{\text{local condition}}$$

A Sufficient Criterion for Liveness

Let *Sys* be an interaction system with set K of components with alphabets A_i, where $i \in K$. $A \subset A_j$ is inevitable in T_j if every infinite path in T_j encompasses an infinite number of transitions labelled with some action in A. Let

$$G = (K, \rightarrow)$$

where

$$i \rightarrow j \Leftrightarrow A_j \backslash \underbrace{excl(i)[j]}_{\text{all } a_j \text{ that occur in some } \alpha \text{ in which } i \text{ does does not participate}} \text{ is inevitable in } \underbrace{T_j}_{\text{local condition}}$$

If $i \rightarrow j$ then j will, when it proceeds, eventually need the cooperation of i.

A Sufficient Criterion for Liveness

Consider now a path in the graph G:

$$k \longrightarrow j_1 \longrightarrow j_2 \longrightarrow \dots j_r.$$

A Sufficient Criterion for Liveness

Consider now a path in the graph G:

$$k \longrightarrow j_1 \longrightarrow j_2 \longrightarrow \ldots j_r.$$

1. Observation: If j_r participates infinitely often in a run σ then by a simple induction argument k participates infinitely often in σ, too.

A Sufficient Criterion for Liveness

Consider now a path in the graph G:

$$k \longrightarrow j_1 \longrightarrow j_2 \longrightarrow \ldots j_r.$$

1. Observation: If j_r participates infinitely often in a run σ then by a simple induction argument k participates infinitely often in σ, too.

2. Observation: If Sys is finite and deadlock-free then in any run σ there must be some component j' that participates infinitely often in σ. If there is a path from k to j' in G then k participates infinitely often in that run σ.

A Sufficient Criterion for Liveness

Consider now a path in the graph G:

$$k \rightarrow j_1 \rightarrow j_2 \rightarrow \dots j_r.$$

1. Observation: If j_r participates infinitely often in a run σ then by a simple induction argument k participates infinitely often in σ, too.

2. Observation: If *Sys* is finite and deadlock-free then in any run σ there must be some component j' that participates infinitely often in σ. If there is a path from k to j' in G then k participates infinitely often in that run σ.

Hence, as a first result: If $Reach(k) = K$, then k is live.

A Sufficient Criterion for Liveness

Consider now a path in the graph G:

$$k \longrightarrow j_1 \longrightarrow j_2 \longrightarrow \ldots j_r.$$

1. Observation: If j_r participates infinitely often in a run σ then by a simple induction argument k participates infinitely often in σ, too.

2. Observation: If Sys is finite and deadlock-free then in any run σ there must be some component j' that participates infinitely often in σ. If there is a path from k to j' in G then k participates infinitely often in that run σ.

Hence, as a first result: If $Reach(k) = K$, then k is live.

But we can do better.

A Sufficient Criterion for Liveness

Let $k \in K$

$$R_0(k) = \{j \mid j \text{ reachable from } k \text{ in } G\}$$

A Sufficient Criterion for Liveness

Let $k \in K$

$$R_0(k) = \{j \mid j \text{ reachable from } k \text{ in } G\}$$

$$R_{i+1}(k) = \{h \mid \forall \alpha \in C \cup Comp : h(\alpha) \neq \emptyset \Rightarrow \exists j \in R_i(k) : j(\alpha) \neq \emptyset\} \cup R_i(k)$$

A Sufficient Criterion for Liveness

Let $k \in K$

$$R_0(k) = \{j \mid j \text{ reachable from } k \text{ in } G\}$$
$$R_{i+1}(k) = \{h \mid \forall \alpha \in C \cup Comp : h(\alpha) \neq \emptyset \Rightarrow \exists j \in R_i(k) : j(\alpha) \neq \emptyset\} \;\cup\; R_i(k)$$

$$\Rightarrow R_0(k) \subseteq R_1(k) \subseteq R_2(k) \subseteq \ldots$$

A Sufficient Criterion for Liveness

Let $k \in K$

$$R_0(k) = \{j \mid j \text{ reachable from } k \text{ in } G\}$$
$$R_{i+1}(k) = \{h \mid \forall \alpha \in C \cup Comp : h(\alpha) \neq \emptyset \Rightarrow \exists j \in R_i(k) : j(\alpha) \neq \emptyset\} \;\cup\; R_i(k)$$

$$\Rightarrow R_0(k) \subseteq R_1(k) \subseteq R_2(k) \subseteq \ldots$$

Consider a run σ where $h \in R_1(k)$ occurs infinitely often. Then, as the system is finite, there must be some α with which h occurs infinitely often in that run. Hence there must be some component $j \in R_0(k)$ with $j(\alpha) \neq \emptyset$. Hence j participates infinitely often in σ.

A Sufficient Criterion for Liveness

Let $k \in K$

$$
\begin{aligned}
R_0(k) &= \{j \mid j \text{ reachable from } k \text{ in } G\} \\
R_{i+1}(k) &= \{h \mid \forall \alpha \in C \cup Comp : h(\alpha) \neq \emptyset \Rightarrow \\
&\qquad \exists j \in R_i(k) : j(\alpha) \neq \emptyset\} \ \cup \ R_i(k)
\end{aligned}
$$

$$
\Rightarrow R_0(k) \subseteq R_1(k) \subseteq R_2(k) \subseteq \ldots
$$

Consider a run σ where $h \in R_1(k)$ occurs infinitely often. Then, as the system is finite, there must be some α with which h occurs infinitely often in that run. Hence there must be some component $j \in R_0(k)$ with $j(\alpha) \neq \emptyset$. Hence j participates infinitely often in σ.

By induction on i we obtain

A Sufficient Criterion for Liveness

Theorem

Let Sys be a finite P-deadlock-free interaction system and $k \in K$ a component. If

$$K = \bigcup_{i \geq 0} R_i(k)$$

then k is P-live in Sys.

A Sufficient Criterion for Liveness

Theorem

Let Sys be a finite P-deadlock-free interaction system and $k \in K$ a component. If

$$K = \bigcup_{i \geq 0} R_i(k)$$

then k is P-live in Sys.

Cost: graph construction and "reachability" - polynomial in $|T_i|$, $|K|$, and $|C \cup Comp|$.

Example - The Dining Philosophers (continued)

It can be shown that modelling the dining philosophers as above is P-deadlock-free where P describes reachability from the designated initial state. This is due to the control component.

Example - The Dining Philosophers (continued)

It can be shown that modelling the dining philosophers as above is P-deadlock-free where P describes reachability from the designated initial state. This is due to the control component.

How can it be guaranteed that no philosopher starves?

Example - The Dining Philosophers (continued)

It can be shown that modelling the dining philosophers as above is P-deadlock-free where P describes reachability from the designated initial state. This is due to the control component.

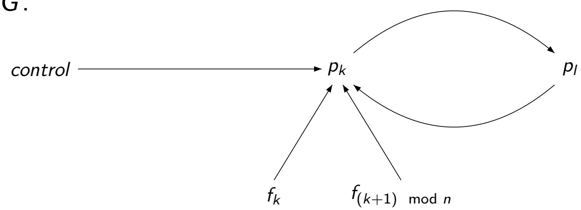
How can it be guaranteed that no philosopher starves?

It suffices to ensure that every philosopher is live because of the linearity of the behavior of the philosophers.

Example - The Dining Philosophers(continued)

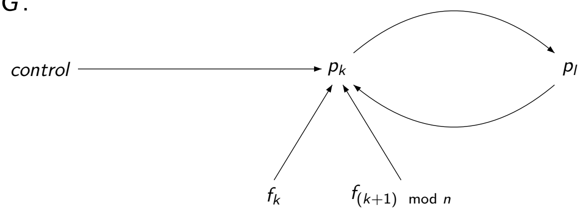
The theorem can be used to show that every philosopher is live.

Part of G:



Example - The Dining Philosophers(continued)

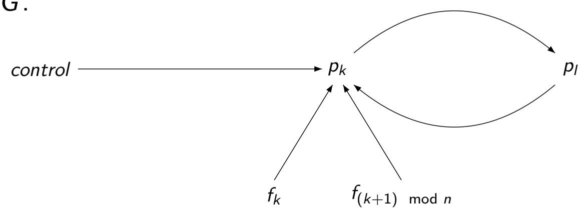
The theorem can be used to show that every philosopher is live.
Part of G:



control is not in $R_0(p_k)$.

Example - The Dining Philosophers(continued)

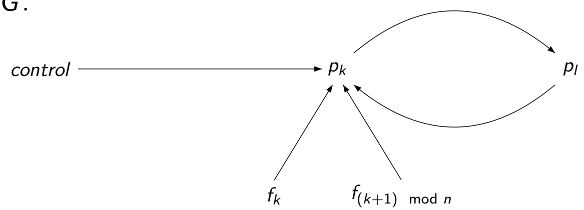
The theorem can be used to show that every philosopher is live.
Part of G:



control is not in $R_0(p_k)$. The interactions in which *control* participates are $\{enter, enter_j\}$, $\{leave, leave_j\}$ $j = 0, ..., n-1$.

Example - The Dining Philosophers(continued)

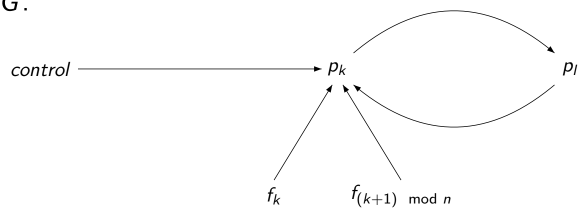
The theorem can be used to show that every philosopher is live.
Part of G:



control is not in $R_0(p_k)$. The interactions in which *control* participates are $\{enter, enter_j\}$, $\{leave, leave_j\}$ $j = 0, ..., n-1$. For any such α there is a philosopher $\in R_0(p_k)$ that participates in α.

Example - The Dining Philosophers(continued)

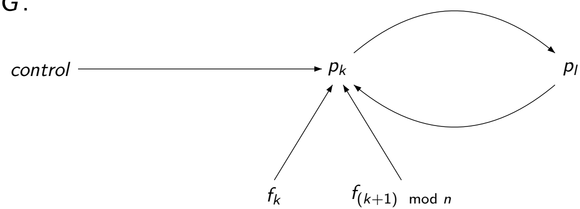
The theorem can be used to show that every philosopher is live.
Part of G:



control is not in $R_0(p_k)$. The interactions in which *control* participates are $\{enter, enter_j\}$, $\{leave, leave_j\}$ $j = 0, ..., n-1$. For any such α there is a philosopher $\in R_0(p_k)$ that participates in α. Similarly for the forks.

Example - The Dining Philosophers(continued)

The theorem can be used to show that every philosopher is live.
Part of G:



control is not in $R_0(p_k)$. The interactions in which *control* participates are $\{enter, enter_j\}$, $\{leave, leave_j\}$ $j = 0, ..., n-1$. For any such α there is a philosopher $\in R_0(p_k)$ that participates in α. Similarly for the forks.

$$\Rightarrow \bigcup_{i \geq 0} R_i(p_k) = K$$

Hence philosopher p_k is live in *Sys*.

Conclusion

- We introduced a framework in which properties of component based systems can be investigated. Testing of properties is expensive.

Conclusion

- We introduced a framework in which properties of component based systems can be investigated. Testing of properties is expensive.
- We established a criterion that ensures liveness in interaction systems and can be tested in polynomial time.

Conclusion

- We introduced a framework in which properties of component based systems can be investigated. Testing of properties is expensive.
- We established a criterion that ensures liveness in interaction systems and can be tested in polynomial time.
- Various other properties have been/are currently investigated:

Conclusion

- We introduced a framework in which properties of component based systems can be investigated. Testing of properties is expensive.
- We established a criterion that ensures liveness in interaction systems and can be tested in polynomial time.
- Various other properties have been/are currently investigated:
  - local/ global deadlock-freedom

Conclusion

- We introduced a framework in which properties of component based systems can be investigated. Testing of properties is expensive.
- We established a criterion that ensures liveness in interaction systems and can be tested in polynomial time.
- Various other properties have been/are currently investigated:
  - local/ global deadlock-freedom
  - liveness of an interaction α or a subset $K' \subseteq K$ of components

Conclusion

- We introduced a framework in which properties of component based systems can be investigated. Testing of properties is expensive.
- We established a criterion that ensures liveness in interaction systems and can be tested in polynomial time.
- Various other properties have been/are currently investigated:
  - local/ global deadlock-freedom
  - liveness of an interaction α or a subset $K' \subseteq K$ of components
  - local progress of $K' \subseteq K$

Conclusion

- We introduced a framework in which properties of component based systems can be investigated. Testing of properties is expensive.
- We established a criterion that ensures liveness in interaction systems and can be tested in polynomial time.
- Various other properties have been/are currently investigated:
  - local/ global deadlock-freedom
  - liveness of an interaction α or a subset $K' \subseteq K$ of components
  - local progress of $K' \subseteq K$
  - robustness of deadlock-freedom in case of failure/removal of components or ports

Conclusion

- We introduced a framework in which properties of component based systems can be investigated. Testing of properties is expensive.
- We established a criterion that ensures liveness in interaction systems and can be tested in polynomial time.
- Various other properties have been/are currently investigated:
  - local/ global deadlock-freedom
  - liveness of an interaction α or a subset $K' \subseteq K$ of components
  - local progress of $K' \subseteq K$
  - robustness of deadlock-freedom in case of failure/removal of components or ports
  - availability of interactions

Conclusion

Work in progress concerning compositionality

Conclusion

Work in progress concerning compositionality

- Define an operator for composing interaction systems

Conclusion

Work in progress concerning compositionality

- Define an operator for composing interaction systems
- Establish conditions under which desirable properties are preserved under composition

Conclusion

Work in progress concerning compositionality

- Define an operator for composing interaction systems
- Establish conditions under which desirable properties are preserved under composition

In addition we introduce probabilities to be able to make statements of the type:

With probability p no deadlock will occur.