Theoretical Foundations of the UML
Lecture 7: Communicating Finite-State Machines

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

moves.rwth-aachen.de/teaching/ss-20/fuml/

May 11, 2020
Outline

1. Introduction
2. Communicating Finite-State Machines
3. Semantics of Communicating Finite-State Machines
4. Emptiness Problem for CFMs
Overview

1. Introduction

2. Communicating Finite-State Machines

3. Semantics of Communicating Finite-State Machines

4. Emptiness Problem for CFMs
• consider (C)MSGs as complete specifications of a system

• MSG \( g \), \( L(g) = \) set of MSCs = set of possible scenarios

Central question: can we obtain a system “realisation” that exhibits all possible scenarios in \( L(g) \)

First question: how do such system “realisations” look like?

- model the behavior of each process by a finite automaton (“local” automaton)
- processes can communicate via unbounded FIFO channels
Consider an MSGs as complete system specifications. They describe a full set of possible system scenarios.

\[ L(G) = \text{set of all possible scenarios} \]
Consider an MSGs as complete system specifications
- they describe a full set of possible system scenarios

Can we obtain “realisations“ that exhibit precisely these scenarios?

central question in the next 3-4 lectures
Consider an MSGs as **complete** system specifications
dthey describe a full set of possible system scenarios

Can we obtain “realisations“ that exhibit precisely these scenarios?

Map MSGs, i.e., scenarios onto an executable **model**

\[ p \rightarrow q \]

- model each process by a **finite-state automaton**
- that communicate via **unbounded directed FIFO channels**

(c) MSG  \[ \rightarrow \] communicating finite-state machine (CFM)
Consider an MSGs as complete system specifications
- they describe a full set of possible system scenarios

Can we obtain “realisations“ that exhibit precisely these scenarios?

Map MSGs, i.e., scenarios onto an executable model
- model each process by a finite-state automaton
- that communicate via unbounded directed FIFO channels

⇒ This yields Communicating Finite-state Machines
Intuition

**Example 1**

Process $p$ and its "realisation" process $q$

```
\[ \begin{array}{c}
\text{"local" automaton of } p \\
\text{"local" automaton of } q
\end{array} \]
```

Global initial state = $(1, A)$

Global final states = $\{ (1, A) \}$

Possible behavior of the CFM:

\[
\begin{align*}
\text{all channels are empty} \\
\text{we are in state } (1, A)
\end{align*}
\]
Example 2

Process $p$

$1 \rightarrow 2$

$2 \rightarrow 1$

$3 \rightarrow 4$

$5 \rightarrow 3$

Process $q$

$A \rightarrow B$

Initial global state $= (1, A)$

Final global states $= \{ (2, B) \}$

MSC is "accepted" by the example CFM (Yannakakis example)
The need for synchronisation messages

Suppose we want to realise

\[
\text{Suppose we want to realise} \quad \{2, B, (3, C)\}
\]

final states

CFM has a deadlock

CFM:

process \( P \)

process \( Q \)
Process $p$ informs process $q$ whether to go "left" or "right".

Automaton for process $p$:

$$F = \{(0,0), (0,0)\}$$

For process $q$:

A deadlock like in the previous example cannot occur.
Introduction

Communicating Finite-State Machines

Semantics of Communicating Finite-State Machines

Emptiness Problem for CFMs
Preliminaries

Definition

Let

- \( \mathcal{P} \) be a finite set of at least two (sequential) processes
- \( \mathcal{C} \) be a finite set of message contents

\( a, b, c \)
Let
• \( \mathcal{P} \) be a finite set of at least two (sequential) processes
• \( \mathcal{C} \) be a finite set of message contents

\[
\text{Definition (communication actions, channels)}
\]
• \( \text{Act}_p^! := \{!(p, q, a) \mid q \in \mathcal{P} \setminus \{p\}, \ a \in \mathcal{C}\} \)
the set of send actions by process \( p \)
Preliminaries

Definition

Let

- $\mathcal{P}$ be a finite set of at least two (sequential) processes
- $\mathcal{C}$ be a finite set of message contents

Definition (communication actions, channels)

- $\text{Act}_p^! := \{!(p, q, a) \mid q \in \mathcal{P} \setminus \{p\}, \ a \in \mathcal{C}\}$
  the set of send actions by process $p$
- $\text{Act}_p^? := \{?(p, q, a) \mid q \in \mathcal{P} \setminus \{p\}, \ a \in \mathcal{C}\}$
  the set of receive actions by process $p$
Preliminaries

Definition

Let
- \( \mathcal{P} \) be a finite set of at least two (sequential) processes
- \( \mathcal{C} \) be a finite set of message contents

Definition (communication actions, channels)

- \( \text{Act}_p^! := \{!(p, q, a) \mid q \in \mathcal{P} \setminus \{p\}, \ a \in \mathcal{C}\} \)
  the set of send actions by process \( p \)
- \( \text{Act}_p^? := \{?(p, q, a) \mid q \in \mathcal{P} \setminus \{p\}, \ a \in \mathcal{C}\} \)
  the set of receive actions by process \( p \)
- \( \text{Act}_p := \text{Act}_p^! \cup \text{Act}_p^? \)
Preliminaries

Definition

Let
- $\mathcal{P}$ be a finite set of at least two (sequential) processes
- $\mathcal{C}$ be a finite set of message contents

Definition (communication actions, channels)

- $\text{Act}_p^! := \{!(p, q, a) \mid q \in \mathcal{P} \setminus \{p\}, \ a \in \mathcal{C}\}$
  the set of send actions by process $p$
- $\text{Act}_p^? := \{?(p, q, a) \mid q \in \mathcal{P} \setminus \{p\}, \ a \in \mathcal{C}\}$
  the set of receive actions by process $p$
- $\text{Act}_p := \text{Act}_p^! \cup \text{Act}_p^?$
- $\text{Act} := \bigcup_{p \in \mathcal{P}} \text{Act}_p$
Preliminaries

Definition

Let

\[ \mathcal{P} \] be a finite set of at least two (sequential) processes

\[ \mathcal{C} \] be a finite set of message contents

Definition (communication actions, channels)

\[ \text{Act}^!_p := \{ !(p, q, a) \mid q \in \mathcal{P} \setminus \{ p \}, \ a \in \mathcal{C} \} \]
the set of send actions by process \( p \)

\[ \text{Act}^?_p := \{ ?(p, q, a) \mid q \in \mathcal{P} \setminus \{ p \}, \ a \in \mathcal{C} \} \]
the set of receive actions by process \( p \)

\[ \text{Act}_p := \text{Act}^!_p \cup \text{Act}^?_p \]

\[ \text{Act} := \bigcup_{p \in \mathcal{P}} \text{Act}_p \]

\[ \text{Ch} := \{ (p, q) \mid p, q \in \mathcal{P}, \ p \neq q \} \]
“channels“
A communicating finite-state machine (CFM) over $\mathcal{P}$ and $\mathcal{C}$ is a structure

$$A = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathbb{D}, s_{init}, F)$$

where

- $\mathbb{D}$: set of global final states
- $s_{init}$: global initial state
- $F$: synchronisation messages (e.g., left, right)
A communicating finite-state machine (CFM) over $\mathcal{P}$ and $\mathcal{C}$ is a structure 

$$
\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathbb{D}, s_{\text{init}}, F)
$$

where

- $\mathbb{D}$ is a nonempty finite set of synchronization messages (or data)

We often write $s \xrightarrow{\sigma,m_p} s'$ instead of $(s, \sigma, m, s') \in \Delta_p$
A communicating finite-state machine (CFM) over $\mathcal{P}$ and $\mathcal{C}$ is a structure

$$\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathbb{D}, s_{\text{init}}, F)$$

where

- $\mathbb{D}$ is a nonempty finite set of synchronization messages (or data)
- for each $p \in \mathcal{P}$:
  - $S_p$ is a non-empty finite set of local states (the $S_p$ are disjoint)
  - $\Delta_p \subseteq S_p \times \text{Act}_p \times \mathbb{D} \times S_p$ is a set of local transitions

We often write $s \xrightarrow{\sigma,m}_p s'$ instead of $(s, \sigma, m, s') \in \Delta_p$
Communicating finite-state machines

Definition

A communicating finite-state machine (CFM) over \( \mathcal{P} \) and \( \mathcal{C} \) is a structure

\[
\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathcal{D}, s_{\text{init}}, F)
\]

where

- \( \mathcal{D} \) is a nonempty finite set of synchronization messages (or data)
- for each \( p \in \mathcal{P} \):
  - \( S_p \) is a non-empty finite set of local states (the \( S_p \) are disjoint)
  - \( \Delta_p \subseteq S_p \times \text{Act}_p \times \mathcal{D} \times S_p \) is a set of local transitions
- \( s_{\text{init}} \in S_\mathcal{A} \) is the global initial state
  - where \( S_\mathcal{A} := \prod_{p \in \mathcal{P}} S_p \) is the set of global states of \( \mathcal{A} \)

We often write \( s \xrightarrow{\sigma,m}_p s' \) instead of \( (s, \sigma, m, s') \in \Delta_p \)
Communicating finite-state machines

Definition

A communicating finite-state machine (CFM) over \( \mathcal{P} \) and \( \mathcal{C} \) is a structure

\[
\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathbb{D}, s_{\text{init}}, F)
\]

where

- \( \mathbb{D} \) is a nonempty finite set of synchronization messages (or data)
- for each \( p \in \mathcal{P} \):
  - \( S_p \) is a non-empty finite set of local states (the \( S_p \) are disjoint)
  - \( \Delta_p \subseteq S_p \times \text{Act}_p \times \mathbb{D} \times S_p \) is a set of local transitions
- \( s_{\text{init}} \in S_A \) is the global initial state
  - where \( S_A := \prod_{p \in \mathcal{P}} S_p \) is the set of global states of \( \mathcal{A} \)
- \( F \subseteq S_A \) is the set of global final states

We often write \( s \xrightarrow{\sigma,m} p s' \) instead of \( (s, \sigma, m, s') \in \Delta_p \)
Communicating finite-state machines

Example

CFM $A$ over $\mathcal{P} = \{1, 2\}$ and $\mathcal{C} = \{\text{req, ack}\}$

- $\mathbb{D} = \{\text{yellow, purple, white}\}$
- $S_1 = \{s_0, s_1, s_2\}$
- $S_2 = \{t_0, t_1, t_2\}$

- $\Delta_1: s_0 \overset{!(1, 2, \text{req})}{\rightarrow} 1 s_0 \ldots$
- $\Delta_2: t_0 \overset{?(2, 1, \text{req})}{\rightarrow} 2 t_1 \ldots$
- $s_{init} = (s_0, t_0)$
- $F = \{(s_2, t_2)\}$
Communicating finite-state machines

Example

![Diagram of communicating finite-state machines]

Joost-Pieter Katoen  Theoretical Foundations of the UML
Communicating finite-state machines

Example

```
! (1, 2, req) \\
\rightarrow

s_0

\uparrow

! (1, 2, req) \\
\rightarrow

s_1

\uparrow

! (1, 2, req) \\
\rightarrow

s_2

? (1, 2, ack) \\
\rightarrow

t_0

? (2, 1, req) \\
\rightarrow

t_1

? (2, 1, req) \\
\rightarrow

t_2

! (2, 1, ack) \\
\rightarrow

!(1, 2, req)

... \\
... \\
... \\
\text{linearisation}
```

CMSC
Communicating finite-state machines

Example

![Diagram of communicating finite-state machines]

Joost-Pieter Katoen  Theoretical Foundations of the UML
Communicating finite-state machines

Example

\[
\begin{align*}
\text{\texttt{s}}_0 & \quad \text{\texttt{s}}_1 \\
?\text{\texttt{(1, 2, req)}} & \quad ?\text{\texttt{(1, 2, ack)}} \\
!\text{\texttt{(1, 2, req)}} & \quad !\text{\texttt{(1, 2, ack)}} \\
?\text{\texttt{(1, 2, ack)}} & \quad ?\text{\texttt{(2, 1, req)}} \\
!\text{\texttt{(1, 2, req)}} & \quad !\text{\texttt{(1, 2, req)}} \\
?\text{\texttt{(2, 1, req)}} & \quad ?\text{\texttt{(2, 1, ack)}} \\
!\text{\texttt{(1, 2, req)}} & \quad !\text{\texttt{(1, 2, req)}} \\
?\text{\texttt{(2, 1, ack)}} & \quad ?\text{\texttt{(2, 1, req)}} \\
!\text{\texttt{(2, 1, ack)}} & \quad !\text{\texttt{(1, 2, req)}} \\
?\text{\texttt{(2, 1, req)}} & \quad ?\text{\texttt{(2, 1, ack)}} \\
!\text{\texttt{(2, 1, req)}} & \quad !\text{\texttt{(1, 2, req)}} \\
\end{align*}
\]
Communicating finite-state machines

Example

![Diagram of communicating finite-state machines]

Joost-Pieter Katoen  Theoretical Foundations of the UML
Communicating finite-state machines

Example

\[
\begin{align*}
(1, 2, \text{req}) & \to s_0 \\
(1, 2, \text{ack}) & \to s_1 \\
(1, 2, \text{req}) & \to s_2 \\
(2, 1, \text{req}) & \to t_0 \\
(2, 1, \text{ack}) & \to t_1 \\
(2, 1, \text{req}) & \to t_2 \\
(1, 2, \text{req}) & \to ?(2, 1, \text{req}) \\
(1, 2, \text{ack}) & \to !(2, 1, \text{ack}) \\
(2, 1, \text{req}) & \to !(2, 1, \text{ack}) \\
(2, 1, \text{ack}) & \to !(2, 1, \text{ack}) \\
(1, 2, \text{req}) & \to !(1, 2, \text{req}) \\
(1, 2, \text{ack}) & \to !(1, 2, \text{ack}) \\
(2, 1, \text{req}) & \to !(2, 1, \text{req}) \\
(2, 1, \text{ack}) & \to !(2, 1, \text{ack}) \\
\end{align*}
\]
Example

Communicating finite-state machines
Communicating finite-state machines

Example

![Diagram of communicating finite-state machines]

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ack</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

!(1, 2, req) !(1, 2, req) ?(2, 1, req) !(2, 1, ack) ?(2, 1, req) !(2, 1, ack) ?(1, 2, ack)
Communicating finite-state machines

Example

\[
\begin{array}{c}
(1, 2, \text{req}) \rightarrow s_0 \\
(1, 2, \text{ack}) \rightarrow s_1 \\
(1, 2, \text{req}) \rightarrow s_2 \\
(2, 1, \text{req}) \rightarrow t_0 \\
(2, 1, \text{ack}) \rightarrow t_1 \\
(2, 1, \text{req}) \rightarrow t_2 \\
\end{array}
\]
Communicating finite-state machines

Example

!\((1, 2, \text{req})\) \(s_0\)

?\((1, 2, \text{ack})\)

\(s_1\)

?\((1, 2, \text{req})\) !\((2, 1, \text{ack})\)

\(s_2\)

?\((1, 2, \text{ack})\)

\(t_0\)

?\((2, 1, \text{req})\) !\((2, 1, \text{ack})\)

\(t_1\)

?\((1, 2, \text{ack})\)

\(t_2\)

?\((2, 1, \text{req})\) !\((2, 1, \text{ack})\)

!\((1, 2, \text{req})\)

!(1, 2, req) !(1, 2, req) ?(2, 1, req) !(2, 1, ack) ?(2, 1, req) !(2, 1, ack) ?(1, 2, ack) !(1, 2, req) ?(1, 2, ack)
Communicating finite-state machines

Example

\[
\begin{align*}
&(1, 2, \text{req}) \quad \text{s}_0 \\
&(2, 1, \text{req}) \quad \text{s}_1 \\
&(1, 2, \text{req}) \quad \text{s}_2
\end{align*}
\]

\[
\begin{align*}
&(1, 2, \text{ack}) \quad \text{s}_0 \\
&(2, 1, \text{ack}) \quad \text{s}_1 \\
&(1, 2, \text{ack}) \quad \text{s}_2
\end{align*}
\]

\[
\begin{align*}
&(1, 2, \text{req}) \quad \text{t}_0 \\
&(2, 1, \text{req}) \quad \text{t}_1 \\
&(1, 2, \text{req}) \quad \text{t}_2
\end{align*}
\]

\[
\begin{align*}
&(1, 2, \text{ack}) \quad \text{t}_0 \\
&(2, 1, \text{ack}) \quad \text{t}_1 \\
&(1, 2, \text{ack}) \quad \text{t}_2
\end{align*}
\]
Communicating finite-state machines

Example

\[ !\text{req}(1, 2) \rightarrow s_0 \]
\[ ?\text{ack}(1, 2) \rightarrow s_1 \]
\[ !\text{req}(1, 2) \rightarrow s_2 \]
\[ ?\text{req}(2, 1) \rightarrow t_0 \]
\[ ?\text{req}(1, 2) \rightarrow t_1 \]
\[ !\text{req}(1, 2) \rightarrow s_0 \]
\[ !\text{req}(2, 1) \rightarrow s_1 \]
\[ !\text{req}(1, 2) \rightarrow s_2 \]
\[ ?\text{req}(2, 1) \rightarrow t_0 \]
\[ ?\text{req}(1, 2) \rightarrow t_1 \]
\[ ?\text{req}(1, 2) \rightarrow t_2 \]

Joost-Pieter Katoen  Theoretical Foundations of the UML
Communicating finite-state machines

Example

\[
\begin{align*}
!(1, 2, \text{req}) & \quad s_0 \\
?& (1, 2, \text{ack}) \\
?& (2, 1, \text{req}) \\
!(2, 1, \text{ack}) & \quad t_0 \\
?& (2, 1, \text{req}) \\
!(2, 1, \text{ack}) & \quad t_1 \\
?& (1, 2, \text{req}) \\
!(2, 1, \text{ack}) & \quad t_2 \\
?& (2, 1, \text{req}) \\
\end{align*}
\]
Communicating finite-state machines

Example

MSC \( M \)

\( M \) is accepted by this CFM.
Overview

1. Introduction
2. Communicating Finite-State Machines
3. Semantics of Communicating Finite-State Machines
4. Emptiness Problem for CFMs
Formal semantics of CFMs

Let $A = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathcal{D}, s_{init}, F)$ be a CFM over $\mathcal{P}$ and $\mathcal{C}$.

**Definition (configurations)**

**Configurations of $A$:**

$Conf_A := S_A \times \{ \eta \mid \eta : \text{Ch} \to (\mathcal{C} \times \mathcal{D})^* \}$

- $\eta : \text{Ch} \to (\mathcal{C} \times \mathcal{D})^*$
- "the content of all channels of the CFM"
- $\eta((p,q)) = \varepsilon$
- $\eta((p,q)) = (a,\bullet),(b,\bullet)$
Let $A = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathcal{D}, s_{init}, F)$ be a CFM over $\mathcal{P}$ and $\mathcal{C}$.

**Definition (configurations)**

Configurations of $A$: $Conf_A := S_A \times \{ \eta \mid \eta : Ch \rightarrow (\mathcal{C} \times \mathcal{D})^* \}$

**Definition (global step)**

$\rightarrow_A \subseteq Conf_A \times Act \times \mathcal{D} \times Conf_A$ is defined as follows:

\[
( (s_1, s_i, s_k), \eta ) \xrightarrow{\eta(s_i, s_j, a), m} ( (s_1, s_i, s_k), \eta' ) \]

$Ch \rightarrow (\mathcal{C} \times \mathcal{D})^*$

\[
?(s_i, s_j, a), m
\]

$|p| = k$
Formal semantics of CFMs

Let $A = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathcal{D}, s_{\text{init}}, F)$ be a CFM over $\mathcal{P}$ and $\mathcal{C}$.

**Definition (configurations)**

Configurations of $A$: $\text{Conf}_A := S_A \times \{ \eta \mid \eta : Ch \rightarrow (\mathcal{C} \times \mathcal{D})^* \}$

**Definition (global step)**

$\rightarrow_A \subseteq \text{Conf}_A \times \text{Act} \times \mathcal{D} \times \text{Conf}_A$ is defined as follows:

- sending a message: $((s, \eta), !(p, q, a), m, (s', \eta')) \in \rightarrow_A$ if
  
  \begin{align*}
  & (s[p], !(p, q, a), m, s'[p]) \in \Delta_p \\
  & \eta' = \eta[p, q] := (a, m) \cdot \eta((p, q)) \\
  & s[r] = s'[r] \text{ for all } r \in \mathcal{P} \setminus \{p\}
  \end{align*}
Formal semantics of CFMs

Let $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathbb{D}, s_{\text{init}}, F)$ be a CFM over $\mathcal{P}$ and $\mathcal{C}$.

Definition (configurations)

Configurations of $\mathcal{A}$: $\text{Conf}_\mathcal{A} := S_\mathcal{A} \times \{\eta \mid \eta : \text{Ch} \to (\mathcal{C} \times \mathbb{D})^*\}$

Definition (global step)

$\longrightarrow_\mathcal{A} \subseteq \text{Conf}_\mathcal{A} \times \text{Act} \times \mathbb{D} \times \text{Conf}_\mathcal{A}$ is defined as follows:

- sending a message: \((\overline{s}, \eta), !(p, q, a), m, (\overline{s}', \eta')\) $\in \longrightarrow_\mathcal{A}$ if
  - \((\overline{s}[p], !(p, q, a), m, \overline{s}'[p])\) $\in \Delta_p$
  - \(\eta' = \eta[(p, q) := a, m \cdot \eta((p, q))]\)
  - \(\overline{s}[r] = \overline{s}'[r]\) for all $r \in \mathcal{P} \setminus \{p\}$

- receipt of a message: \((\overline{s}, \eta), ?(p, q, a), m, (\overline{s}', \eta')\) $\in \longrightarrow_\mathcal{A}$ if
  - \((\overline{s}[p], ?(p, q, a), m, \overline{s}'[p])\) $\in \Delta_p$
  - \(\eta((q, p)) = w \cdot (a, m) \neq \epsilon\) and \(\eta' = \eta[(q, p) := w]\)
  - \(\overline{s}[r] = \overline{s}'[r]\) for all $r \in \mathcal{P} \setminus \{p\}$
receipt of a message: $((\bar{s}, \eta), ?(p, q, a), m, (\bar{s}', \eta')) \in \rightarrow_{A}$ if

- $(\bar{s}[p], ?(p, q, a), m, \bar{s}'[p]) \in \Delta_p$
- $\eta((q, p)) = w \cdot (a, m) \neq \varepsilon$ and $\eta' = \eta[(q, p) := w]$
- $\bar{s}[r] = \bar{s}'[r]$ for all $r \in \mathcal{P} \setminus \{p\}$

\[
\begin{align*}
\left((s_1, \ldots, s_p, \ldots, s_k), \eta\right) & \quad \xrightarrow{?(p, q, a), m} \quad \left((s_1, \ldots, s_p', \ldots, s_k), \eta'\right) \\
& \quad \xrightarrow{q \rightarrow p} \quad \text{current configuration}
\end{align*}
\]

\[
\begin{array}{c}
m(m) \quad a, m \\
\quad \text{w} \\
\quad \neq \varepsilon \\
\eta'((q, p)) = w \\
\text{for all other channels } c \in \mathcal{P} \setminus \{p\} \\
\eta'(c) = \eta(c)
\end{array}
\]
Example
Example 2

process p

\[
\begin{array}{c}
\rightarrow\quad \mathbb{1} \quad !a \\
\quad \downarrow \quad \text{empty} \\
\quad \mathbb{2} \quad ?b \\
\rightarrow \quad \mathbb{3} \\
\end{array}
\]

\[
\begin{array}{c}
\rightarrow \quad \mathbb{2} \quad !a \\
\end{array}
\]

process q

\[
\begin{array}{c}
\rightarrow \quad \mathbb{A} \quad ?a \\
\downarrow \quad \text{empty} \\
\rightarrow \quad \mathbb{B} \quad ?b \\
\end{array}
\]

\[
\begin{array}{c}
\rightarrow \quad \mathbb{C} \\
\end{array}
\]

\[= \mathbb{3} \]

\[\eta(p, q) = \varepsilon\]
\[\eta(q, p) = \varepsilon\]

global initial state = \((1, A)\)

global final state = \((2, B)\)

starting configuration: \((1, A), (e, \varepsilon)\) \[= \mathcal{T}_0\]

\[\Downarrow \text{empty} \]

\[(1, C), (e, b)\) \[= \mathcal{T}_1\]

\[\Downarrow \text{empty} \]

\[(2, C), (a \cdot b)\) \[= \mathcal{T}_2\]

\[\Downarrow \quad \varepsilon \quad \eta((p, q)) = a \cdot a\]

\[\eta((q, p)) = b\]

\[\cdots \quad (2, C), (a \cdot a, \varepsilon) \Leftarrow (3, C), (a \cdot a, b)\) \[= \mathcal{T}_3\]

\[= \mathcal{T}_4\]
Linearizations of a CFM

Let $A = (((S_p, \Delta_p))_{p \in P}, D, s_{init}, F)$ be a CFM over $P$ and $C$.

**Definition (accepting runs)**

A run $\rho$ of CFM $A$ on word $w = \sigma_1 \ldots \sigma_n \in Act^*$ is an alternating sequence $\rho = \gamma_0 m_1 \gamma_1 \ldots \gamma_{n-1} m_n \gamma_n$ such that

1. $\gamma_0 = (s_{init}, \eta_\varepsilon)$ with $\eta_\varepsilon$ mapping any channel to $\varepsilon$ (empty content)
2. $\gamma_i = A \gamma_i$ for any $i \in \{1, \ldots, n\}$
Linearizations of a CFM

Let $\mathcal{A} = (((S_p, \Delta_p))_{p \in \mathcal{P}}, \mathbb{D}, s_{\text{init}}, F)$ be a CFM over $\mathcal{P}$ and $\mathcal{C}$.

**Definition (accepting runs)**

A run $\rho$ of CFM $\mathcal{A}$ on word $w = \sigma_1 \ldots \sigma_n \in \text{Act}^*$ is an alternating sequence $\rho = \gamma_0 m_1 \gamma_1 \ldots \gamma_{n-1} m_n \gamma_n$ such that:

1. $\gamma_0 = (s_{\text{init}}, \eta_{\varepsilon})$ with $\eta_{\varepsilon}$ mapping any channel to $\varepsilon$
2. $\gamma_{i-1} \xrightarrow{\sigma_i, m_i} A \gamma_i$ for any $i \in \{1, \ldots, n\}$

The run $\rho$ is accepting if $\gamma_n \in F \times \{\eta_{\varepsilon}\}$. 

$fn = \text{global final state}$

all channels are empty.
Let $A = (((S_p, \Delta_p))_{p \in P}, \mathbb{D}, s_{\text{init}}, F')$ be a CFM over $P$ and $C$.

**Definition (accepting runs)**

A run $\rho$ of CFM $A$ on word $w = \sigma_1 \ldots \sigma_n \in Act^*$ is an alternating sequence $\rho = \gamma_0 m_1 \gamma_1 \ldots \gamma_{n-1} m_n \gamma_n$ such that

1. $\gamma_0 = (s_{\text{init}}, \eta_\varepsilon)$ with $\eta_\varepsilon$ mapping any channel to $\varepsilon$
2. $\gamma_{i-1} \xrightarrow{\sigma_i, m_i} A \gamma_i$ for any $i \in \{1, \ldots, n\}$

The run $\rho$ is accepting if $\gamma_n \in F \times \{\eta_\varepsilon\}$.

**Definition (linearization of a CFM)**

The (word) language of CFM $A$ is defined by:

$Lin(A) := \{w \in Act^* \mid \text{there is an accepting run of } A \text{ on } w\}$
Linearizations of an example CFM

Example

CFM \( A \) over \( \{1, 2\} \) and \( \{\text{req, ack}\} \)
Linearizations of an example CFM

Example

\[ \text{Lin}(A) = \{ w \in Act^* \mid \text{there is } n \geq 1 \text{ such that:} \]

\[ w \upharpoonright 1 = (!1, 2, \text{req}))^n (?(1, 2, \text{ack}) !(1, 2, \text{req}))^n \]

\[ w \upharpoonright 2 = (?(2, 1, \text{req}) !(2, 1, \text{ack}))^n (?(2, 1, \text{req}))^n \]

for any \( u \in \text{Pref}(w) \) and \( (p, q) \in \text{Ch} \):

\[ \sum_{a \in C} |u|!(p,q,a) - \sum_{a \in C} |u|?(q,p,a) \geq 0 \]

CFM \( A \) over \{1, 2\} and \{\text{req, ack}\}

\( \text{WM}_1 = \) the sequence of actions in \( w \) that occur at process 1

\( \text{WM}_2 = \ldots \) for process 2.
Linearizations of an example CFM

Example

![Diagram of a Finite Control Machine (FCM)]

CFM $\mathcal{A}$ over
\{1, 2\} and \{req, ack\}

- !(1, 2, req) and !(2, 1, ack) are always independent.
- !(1, 2, req) and ?(1, 2, ack) are always dependent.
- !(1, 2, req) and ?(2, 1, req) are sometimes independent.

$\Rightarrow$ non-regular (word) languages $\Rightarrow$ more expressive than finite-state automata!
Linearizations and MSCs of an example CFM

Example

\[
\text{CFM } \mathcal{A} \text{ over } \{1, 2\} \text{ and } \{\text{req, ack}\}
\]

\[
\text{Lin}(\mathcal{A}) = \left\{ w \in \text{Act}^* \mid \text{there is } n \geq 1 \text{ such that:} \right. \\
w | 1 = (\!(1, 2, \text{req})\!)^n (\?(1, 2, \text{ack}) \! (1, 2, \text{req}))^n \\
w | 2 = (\?(2, 1, \text{req}) \! (2, 1, \text{ack}))^n (\!(2, 1, \text{req})\!)^n \\
\text{for any } u \in \text{Pref}(w) \text{ and } (p, q) \in \text{Ch:} \\
\sum_{a \in \mathcal{C}} |u|_{!\!(p,q,a)} - \sum_{a \in \mathcal{C}} |u|_{?(q,p,a)} \geq 0 \right\}
\]
Linearizations and MSCs of an example CFM

Example

\[ \mathcal{L}(A) = \left\{ M \in \mathcal{M} \mid \text{there is } n \geq 1 \text{ such that:} \right\} \]

\[ M \upharpoonright 1 = (\textcolor{red}{!(1,2,\text{req})})^n \textcolor{blue}{(?(1,2,\text{ack}) \text{!}(1,2,\text{req}))^n} \]

\[ M \upharpoonright 2 = (\textcolor{blue}{?(2,1,\text{req}) \text{!}(2,1,\text{ack})})^n \textcolor{red}{(?(2,1,\text{req}))^n} \]

CFM \( A \) over \( \{1,2\} \) and \( \{\text{req, ack}\} \)
Overview

1 Introduction

2 Communicating Finite-State Machines

3 Semantics of Communicating Finite-State Machines

4 Emptiness Problem for CFMs

---

CFMs are more expressive than finite-state automata

does a CFM accept at least one word? undecidable
Elementary questions are undecidable for CFMs

Emptiness of CFMs is undecidable

The following problem is undecidable (even if \( C \) is a singleton):

**Input:** CFM \( A \) over processes \( P \) and message contents \( C \)

**Question:** Is \( \mathcal{L}(A) \) empty?

Note: \( \mathcal{L}(A) \) refers to the set of MSCs accepted by CFM \( A \), and \( \text{Lin}(A) \) refers to the set of linearizations accepted by CFM \( A \).

Example: \( C = \{a\} \)
Elementary questions are undecidable for CFMs

**Emptiness of CFMs is undecidable**

The following problem is undecidable (even if $\mathcal{C}$ is a singleton):

**Input:** CFM $\mathcal{A}$ over processes $\mathcal{P}$ and message contents $\mathcal{C}$

**Question:** Is $\mathcal{L}(\mathcal{A})$ empty?

**Proof (sketch)**

Reduction from the halting problem for Turing machine $TM = (Q, \Sigma, \Delta, \square, q_0, q_f)$ to emptiness for a CFM with two processes.

Build CFM $\mathcal{A} = ((\mathcal{A}_1, \mathcal{A}_2), \mathbb{D}, s_{init}, F)$ over $\{1, 2\}$ and some singleton set $\mathcal{C}$ such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $TM$ can reach $q_f$, i.e., $TM$ accepts.

- Process 1 sends current configurations to process 2
- Process 2 chooses successor configurations and sends them to 1
- $\mathbb{D} = \left( (\Sigma \cup \{\square\}) \times (Q \cup \{\_\}) \right) \cup \{\#\}$
A CFM simulating a Turing machine

Proof (contd.)
Proof (contd.)

- **Left or standstill transition:** Process 2 may just wait for a symbol containing a state of \( TM \) and to alter it correspondingly. In the example, the left-moving transition \((q_2, a, a', L, q_3)\) is applied so that process 2
  - sends \( b \) unchanged back to process 1
  - detects (receives) \( a ← q_2 \)
  - sends \( a' \) to process 1 entering a state indicating that the symbol to be sent next has to be equipped with \( q_3 \)
  - receives \( \# \) so that the symbol \( □ ← q_3 \) has to be inserted before returning \( \# \)
Left or standstill transition: Process 2 may just wait for a symbol containing a state of \( TM \) and to alter it correspondingly. In the example, the left-moving transition \((q_2, a, a', L, q_3)\) is applied so that process 2

- sends \( b \) unchanged back to process 1
- detects (receives) \( a \leftarrow q_2 \)
- sends \( a' \) to process 1 entering a state indicating that the symbol to be sent next has to be equipped with \( q_3 \)
- receives \( \# \) so that the symbol \( \sqcup \leftarrow q_3 \) has to be inserted before returning \( \# \)

Right transition: Process 2 has to guess what the position right before the head is. For example, provided process 2 decided in favor of \((q_2, a, a', R, q_3)\) while reading \( b \), it would have to

- send \( b \leftarrow q_3 \) instead of just \( b \), entering some state \( t(a \leftarrow q_2) \)
- receive \( a \leftarrow q_2 \) (no other symbol can be received in state \( t(a \leftarrow q_2) \))
- send \( a' \) back to process 1
Communicating Finite-state Machines

realisation of system operational model of an implementation

(c) MSG = "requirements"

all scenarios a system should exhibit
Proof (contd.)

- Introduce local final states $s_f$ and $t_f$, one for process 1 and one for process 2, respectively (i.e., $F = \{(s_f, t_f)\}$ and $A$ is locally accepting).

- At any time, process 1 may switch into $s_f$, in which arbitrary and arbitrarily many messages can be received to empty channel $(2, 1)$.

- Process 2 is allowed to move into $t_f$ and to empty the channel $(1, 2)$ as soon as it receives a letter $c \leftarrow q_f$ for some $c$.

- As process 2 modifies a configuration of $TM$ locally, finitely many states are sufficient in $A$. □