Theoretical Foundations of the UML

Lecture 8: Communicating Finite-State Machines

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

http://moves.rwth-aachen.de/teaching/ws-1415/uml/

14. November 2014

Outline

Overview

Specification to implementation

Specification to implementation

- Consider an MSGs as complete system specifications
  - they describe a full set of possible system scenarios

Specification to implementation

- 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?

Specification to implementation

- 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

Specification to implementation

- 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

$\Rightarrow$ This yields Communicating Finite-state Machines

Intuition

The need for synchronisation messages

Overview

Preliminaries

Definition

Let

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

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)

- $Act_p^! := \{!(p, q, a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$
  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)

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

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

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)

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

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)

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

Communicating finite-state machines

Definition

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

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

where

Communicating finite-state machines

Definition

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

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

where

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

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

Communicating finite-state machines

Definition

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

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

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)
  - $\Delta_p \subseteq S_p \times Act_p \times \mathbb{D} \times S_p$ 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

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

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)
  - $\Delta_p \subseteq S_p \times Act_p \times \mathbb{D} \times S_p$ is a set of local transitions
- $s_{init} \in S_{\mathcal{A}}$ 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

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

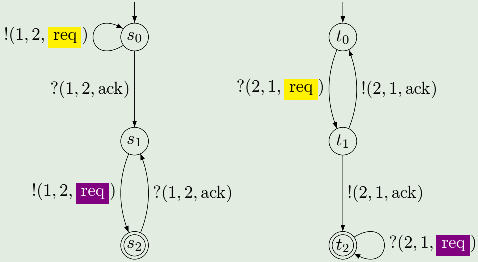
where

- $\mathbb{D}$ 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)
  - $\Delta_p \subseteq S_p \times Act_p \times \mathbb{D} \times S_p$ is a set of local transitions
- $s_{init} \in S_{\mathcal{A}}$ is the global initial state
  - where $S_{\mathcal{A}} := \prod_{p \in \mathcal{P}} S_p$ is the set of global states of $\mathcal{A}$
- $F \subseteq S_{\mathcal{A}}$ 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



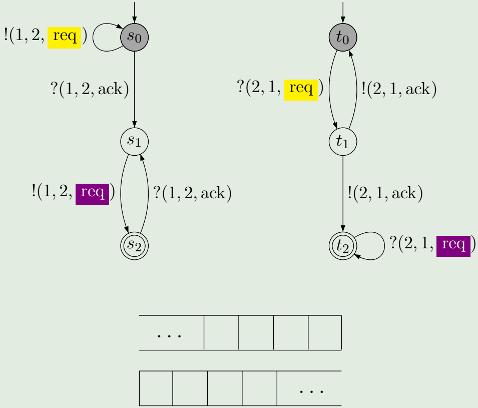
CFM $\mathcal{A}$ over $\mathcal{P} = \{1,2\}$ and $\mathcal{C} = \{\text{req}, \text{ack}\}$

- $\mathbb{D} = \{\boxed{}, \boxed{}, \boxed{}\}$
- $S_1 = \{s_0, s_1, s_2\}$
- $S_2 = \{t_0, t_1, t_2\}$
- Δ_1: $s_0 \xrightarrow{!(1,2,\text{req})}_1 s_0 \ \ldots$

  Δ_2: $t_0 \xrightarrow{?(2,1,\text{req})}_2 t_1 \ \ldots$
- $s_{init} = (s_0, t_0)$
- $F = \{(s_2, t_2)\}$

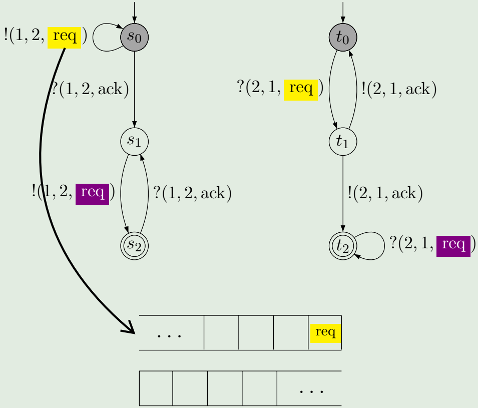
Communicating finite-state machines

Example



Communicating finite-state machines

Example

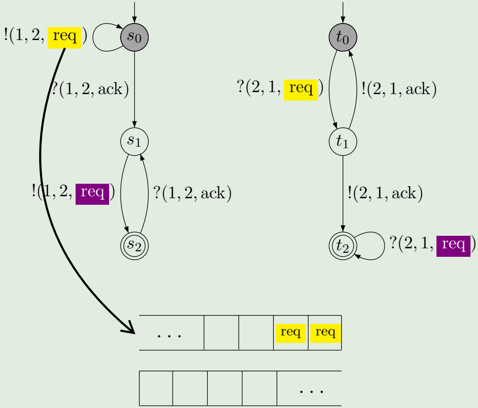


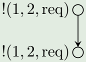
$!(1,2,\mathrm{req})$

$!(1,2,\mathrm{req})$

Communicating finite-state machines

Example

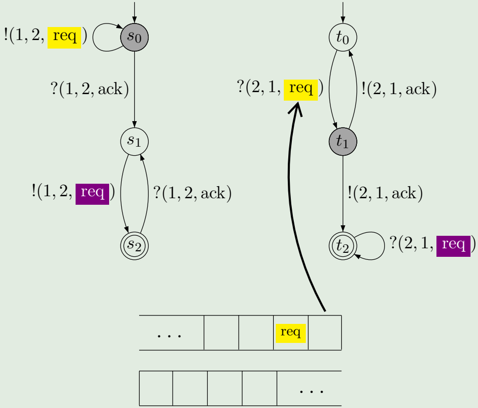




$!(1, 2, \text{req})$ $!(1, 2, \text{req})$

Communicating finite-state machines

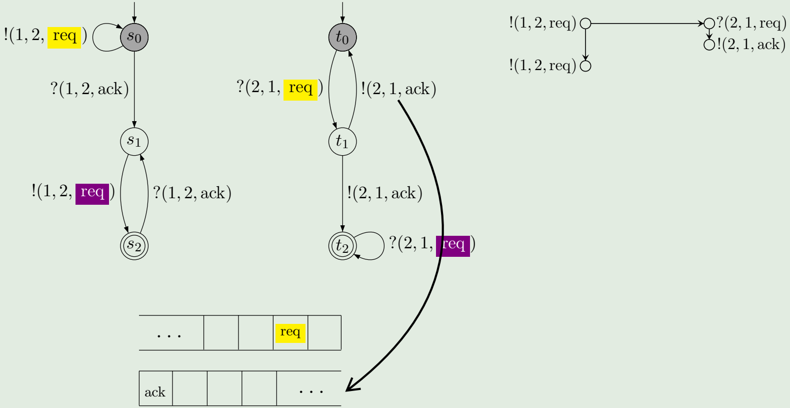
Example



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

Communicating finite-state machines

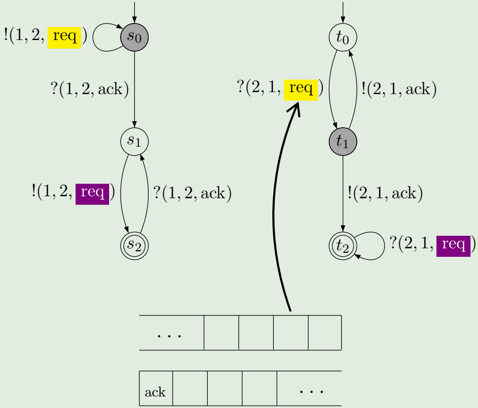
Example



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

Communicating finite-state machines

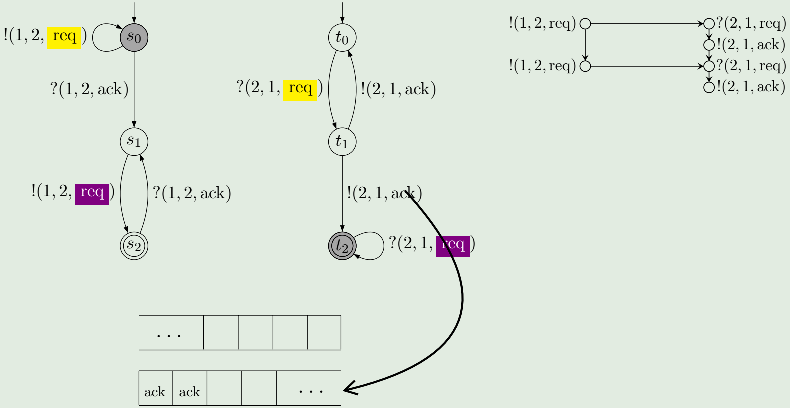
Example



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

Communicating finite-state machines

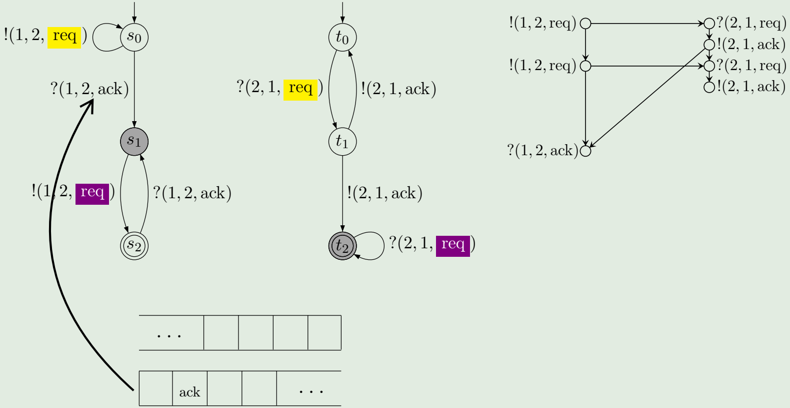
Example



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

Communicating finite-state machines

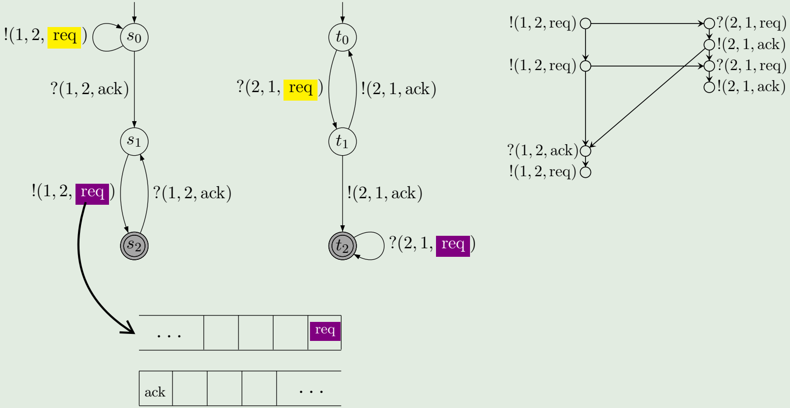
Example



!(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

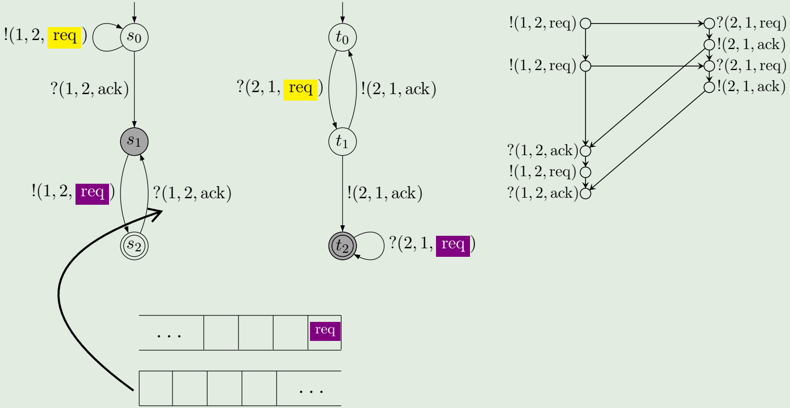
Example



!(1, 2, req) !(1, 2, req) ?(2, 1, req) !(2, 1, ack) ?(2, 1, req) !(2, 1, ack) ?(1, 2, ack) !(1, 2, req)

Communicating finite-state machines

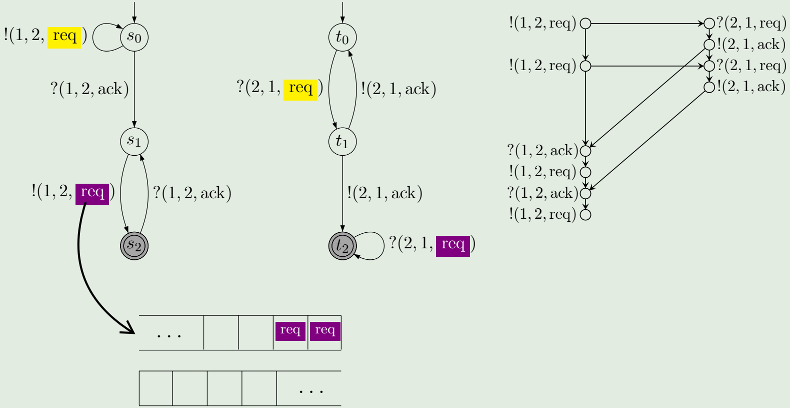
Example



!(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

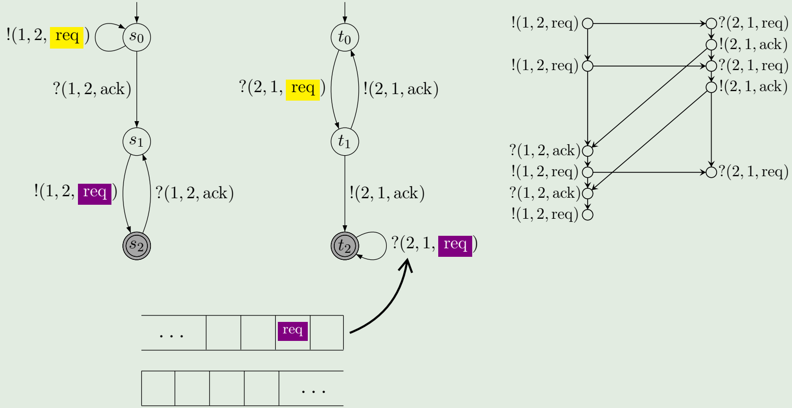
Example



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

Communicating finite-state machines

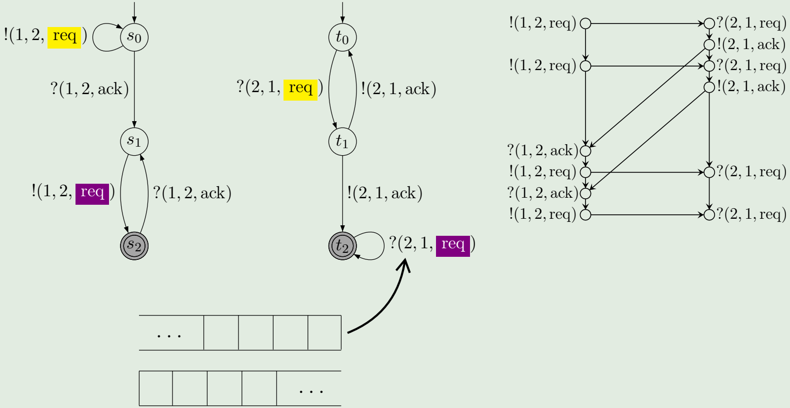
Example



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

Communicating finite-state machines

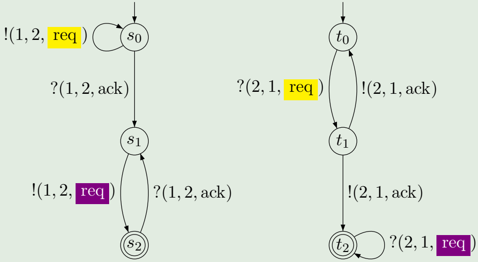
Example

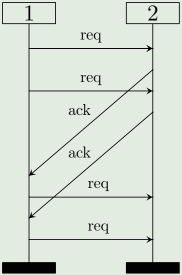


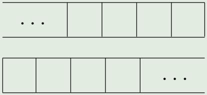
$!(1,2,\mathrm{req})\ !(1,2,\mathrm{req})\ ?(2,1,\mathrm{req})\ !(2,1,\mathrm{ack})\ ?(2,1,\mathrm{req})\ !(2,1,\mathrm{ack})\ ?(1,2,\mathrm{ack})\ !(1,2,\mathrm{req})\ ?(1,2,\mathrm{ack})\ !(1,2,\mathrm{req})\ ?(2$

Communicating finite-state machines

Example







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

Formal semantics of CFMs

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

Definition (configurations)

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

Formal semantics of CFMs

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

Definition (configurations)

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

Definition (global step)

$\Longrightarrow_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times Act \times \mathbb{D} \times Conf_{\mathcal{A}}$ is defined as follows:

Formal semantics of CFMs

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

Definition (configurations)

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

Definition (global step)

$\Longrightarrow_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times Act \times \mathbb{D} \times Conf_{\mathcal{A}}$ 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 $r \in \mathcal{P} \setminus \{p\}$

Formal semantics of CFMs

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

Definition (configurations)

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

Definition (global step)

$\Longrightarrow_{\mathcal{A}} \subseteq \mathit{Conf}_{\mathcal{A}} \times \mathit{Act} \times \mathbb{D} \times \mathit{Conf}_{\mathcal{A}}$ 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 $r \in \mathcal{P} \setminus \{p\}$

Example

Linearizations of a CFM

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

Definition (accepting runs)

A run ρ of CFM $\mathcal{A}$ 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 η_ε mapping any channel to ε
2. $\gamma_{i-1} \xRightarrow{\sigma_i, m_i}_{\mathcal{A}} \gamma_i$ for any $i \in \{1, \ldots, n\}$

Linearizations of a CFM

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

Definition (accepting runs)

A run ρ of CFM $\mathcal{A}$ 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 η_ε mapping any channel to ε
2. $\gamma_{i-1} \xRightarrow{\sigma_i, m_i}_{\mathcal{A}} \gamma_i$ for any $i \in \{1, \ldots, n\}$

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

Linearizations of a CFM

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

Definition (accepting runs)

A run ρ of CFM $\mathcal{A}$ 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 η_ε mapping any channel to ε
2. $\gamma_{i-1} \xRightarrow{\sigma_i, m_i}_{\mathcal{A}} \gamma_i$ for any $i \in \{1, \ldots, n\}$

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

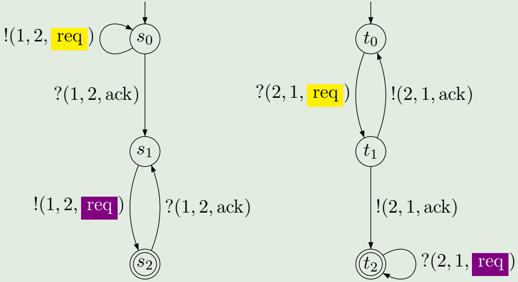
Definition (linearization of a CFM)

The (word) language of CFM $\mathcal{A}$ is defined by:

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

Linearizations of an example CFM

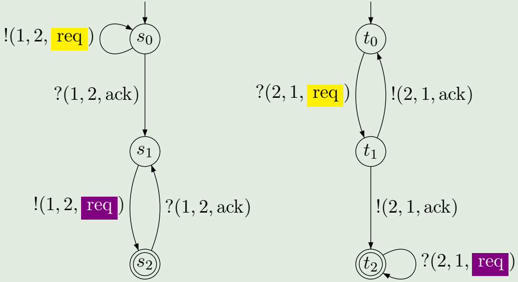
Example



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

Linearizations of an example CFM

Example



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

$$Lin(\mathcal{A}) = \Big\{ w \in Act^* \mid \text{there is } n \geqslant 1 \text{ such that:}$$

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

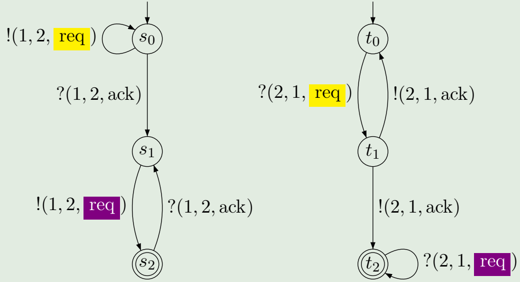
$$w\!\upharpoonright\!2 = (?(2,1,\text{req}) \; !(2,1,\text{ack}))^n \;\; (?(2,1,\text{req}))^n$$

for any $u \in Pref(w)$ and $(p,q) \in Ch$:

$$\sum_{a \in \mathcal{C}} |u|_{!(p,q,a)} - \sum_{a \in \mathcal{C}} |u|_{?(q,p,a)} \geqslant 0 \Big\}$$

Linearizations of an example CFM

Example



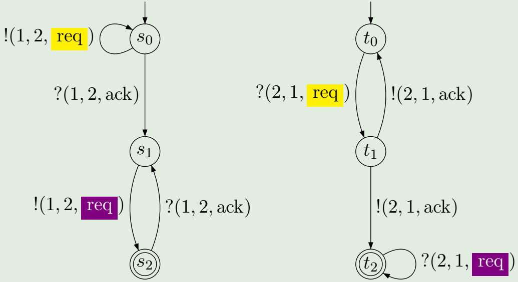
CFM $\mathcal{A}$ over $\{1, 2\}$ and $\{\text{req}, \text{ack}\}$

- $!(1, 2, \text{req})$ and $!(2, 1, \text{ack})$ are always independent.
- $!(1, 2, \text{req})$ and $?(1, 2, \text{ack})$ are always dependent.
- $!(1, 2, \text{req})$ and $?(2, 1, \text{req})$ are sometimes independent.

$\rightsquigarrow$ non-regular (word) languages

Linearizations and MSCs of an example CFM

Example



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

$$Lin(\mathcal{A}) = \Big\{ w \in Act^* \mid \text{there is } n \geqslant 1 \text{ such that:}$$

$$w \restriction 1 = (!(1,2,\text{req}))^n \; (?(1,2,\text{ack}) \; !(1,2,\text{req}))^n$$

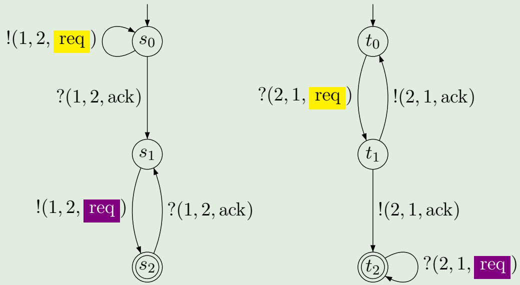
$$w \restriction 2 = (?(2,1,\text{req}) \; !(2,1,\text{ack}))^n \; (?(2,1,\text{req}))^n$$

for any $u \in Pref(w)$ and $(p,q) \in Ch$:

$$\sum_{a \in \mathcal{C}} |u|_{!(p,q,a)} - \sum_{a \in \mathcal{C}} |u|_{?(q,p,a)} \geqslant 0 \Big\}$$

Linearizations and MSCs of an example CFM

Example



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

$$\mathcal{L}(\mathcal{A}) = \Big\{ M \in \mathbb{M} \mid \text{there is } n \geq 1 \text{ such that:}$$

$$M \restriction 1 = (!(1,2,\text{req}))^k \; (?(1,2,\text{ack}) \; !(1,2,\text{req}))^n$$

$$M \restriction 2 = (?(2,1,\text{req}) \; !(2,1,\text{ack}))^n \; (?(2,1,\text{req}))^k \Big\}$$

Overview

Elementary questions are undecidable for CFMs

Emptiness of CFMs is undecidable [Brand & Zafiropulo 1983]

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?

Elementary questions are undecidable for CFMs

Emptiness of CFMs is undecidable [Brand & Zafiropulo 1983]

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?

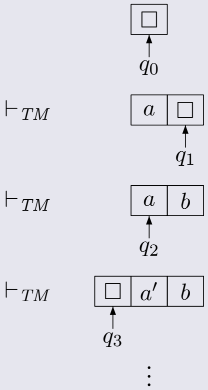
Proof (sketch)

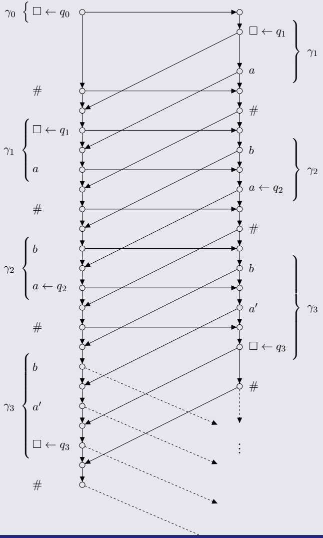
Reduction from the halting problem for Turing machine $TM = (Q, \Sigma, \Delta, \Box, q_0, q_f)$ to emptiness for a CFM with two processes. Build CFM $\mathcal{A} = ((\mathcal{A}_1, \mathcal{A}_2), \mathbb{D}, s_{init}, F)$ over $\{1, 2\}$ and some singleton set $\mathcal{C}$ such that $\mathcal{L}(\mathcal{A}) \neq \varnothing$ 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
- $\mathbb{D} = \Big((\Sigma \cup \{\Box\}) \times (Q \cup \{\_\}) \Big) \cup \{\#\}$

A CFM simulating a Turing machine

Proof (contd.)





A CFM simulating a Turing machine

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) $a \leftarrow q_2$
  - 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 $\Box \leftarrow q_3$ has to be inserted before returning $\#$

A CFM simulating a Turing machine

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) $a \leftarrow q_2$
  - 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 $\Box \leftarrow q_3$ 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

A CFM simulating a Turing machine

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 $\mathcal{A}$ is locally accepting).

A CFM simulating a Turing machine

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 $\mathcal{A}$ 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 $(2, 1)$.

A CFM simulating a Turing machine

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 $\mathcal{A}$ 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.

A CFM simulating a Turing machine

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 $\mathcal{A}$ 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 $\mathcal{A}$. □