Foundations of the UML

Lecture 6: Communicating Finite-State Machines

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

http://moves.rwth-aachen.de/i2/370

16. November 2009

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 FIFO channels

$\Rightarrow$ Communicating finite-state machines

Intuition

The need for synchronisation messages

The architecture of a message-passing system

Definition

We fix the following parameters:

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

The architecture of a message-passing system

Definition

We fix the following parameters:

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

Definition (communication actions, channels)

- $Act^!_p := \{!(p,q,a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$ (for $p \in \mathcal{P}$)
  "p sends message a to q"

The architecture of a message-passing system

Definition

We fix the following parameters:

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

Definition (communication actions, channels)

- $Act_p^! := \{!(p, q, a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$ (for $p \in \mathcal{P}$)
  "p sends message a to q"
- $Act_p^? := \{?(p, q, a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$
  "p receives message a from q"

The architecture of a message-passing system

Definition

We fix the following parameters:

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

Definition (communication actions, channels)

- $Act_p^! := \{!(p, q, a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$ (for $p \in \mathcal{P}$)
  "p sends message a to q"
- $Act_p^? := \{?(p, q, a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$
  "p receives message a from q"
- $Act_p := Act_p^! \cup Act_p^?$

The architecture of a message-passing system

Definition

We fix the following parameters:

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

Definition (communication actions, channels)

- $Act_p^! := \{!(p,q,a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$ (for $p \in \mathcal{P}$)
  "p sends message a to q"
- $Act_p^? := \{?(p,q,a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$
  "p receives message a from q"
- $Act_p := Act_p^! \cup Act_p^?$
- $Act := \bigcup_{p \in \mathcal{P}} Act_p$

The architecture of a message-passing system

Definition

We fix the following parameters:

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

Definition (communication actions, channels)

- $Act_p^! := \{!(p,q,a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$ (for $p \in \mathcal{P}$)
  "p sends message a to q"
- $Act_p^? := \{?(p,q,a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$
  "p receives message a from q"
- $Act_p := Act_p^! \cup Act_p^?$
- $Act := \bigcup_{p \in \mathcal{P}} Act_p$
- $Ch := \{(p,q) \mid p,q \in \mathcal{P},\ p \neq q\}$ "channels"

The architecture of a message-passing system

Definition

We fix the following parameters:

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

Definition (communication actions, channels)

- $Act^!_p := \{!(p, q, a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$ (for $p \in \mathcal{P}$)
  "p sends message a to q"
- $Act^?_p := \{?(p, q, a) \mid q \in \mathcal{P} \setminus \{p\},\ a \in \mathcal{C}\}$
  "p receives message a from q"
- $Act_p := Act^!_p \cup Act^?_p$
- $Act := \bigcup_{p \in \mathcal{P}} Act_p$
- $Ch := \{(p, q) \mid p, q \in \mathcal{P},\ p \neq q\}$ "channels"
- $Comm := \{(!(p, q, a), ?(q, p, a)) \mid (p, q) \in Ch,\ a \in \mathcal{C}\}$

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)

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

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

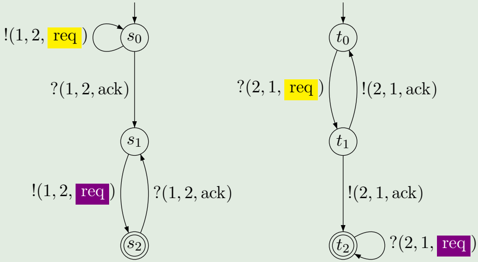
where

- $\mathbb{D}$ 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

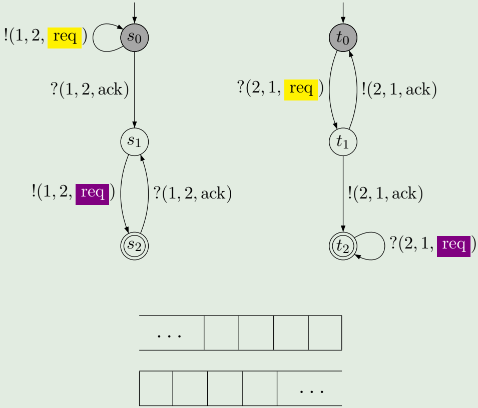


CFM $\mathcal{A}$ over $\{1, 2\}$ and $\{\text{req}, \text{ack}\}$

- $\mathbb{D} = \{\square, \square, \square\}$
- $S_1 = \{s_0, s_1, s_2\}$
- $S_2 = \{t_0, t_1, t_2\}$
- Δ_1: $s_0 \xrightarrow{!(1,2,\text{req})}_1 s_0 \ \ldots$
  Δ_2: $t_0 \xrightarrow{?(2,1,\text{req})}_2 t_1 \ \ldots$
- $s_{init} = (s_0, t_0)$
- $F = \{(s_2, t_2)\}$

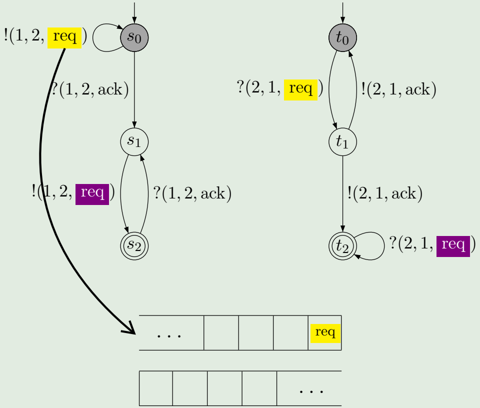
Communicating finite-state machines

Example



Communicating finite-state machines

Example

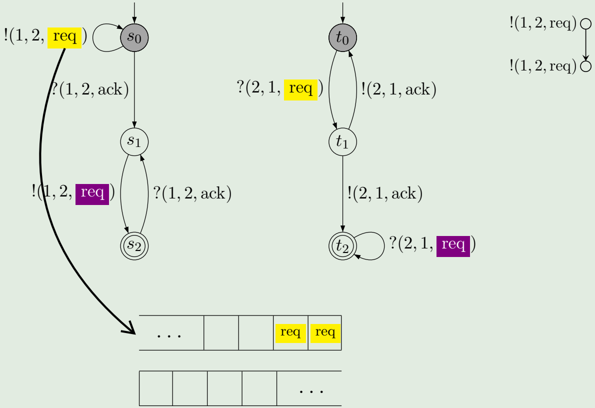


!(1, 2, req) ○

!(1, 2, req)

Communicating finite-state machines

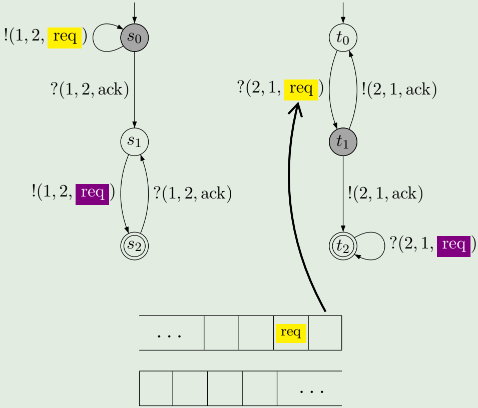
Example



!(1, 2, req) !(1, 2, req)

Communicating finite-state machines

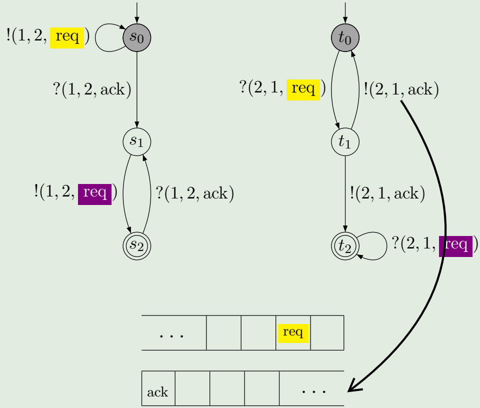
Example



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

Communicating finite-state machines

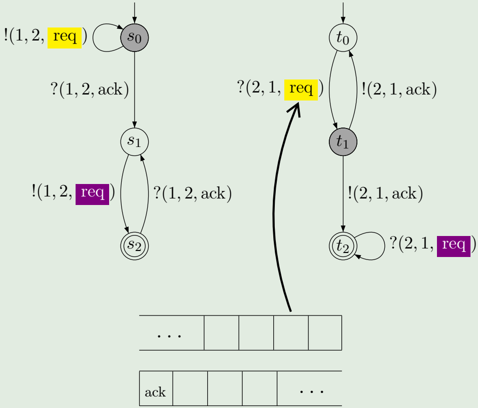
Example



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

Communicating finite-state machines

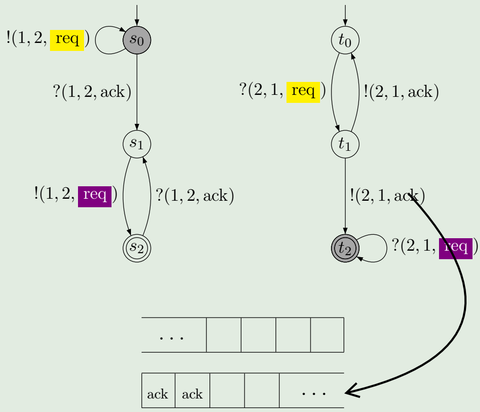
Example

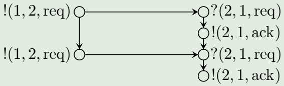


!(1, 2, req) !(1, 2, req) ?(2, 1, req) !(2, 1, ack) ?(2, 1, req)

Communicating finite-state machines

Example

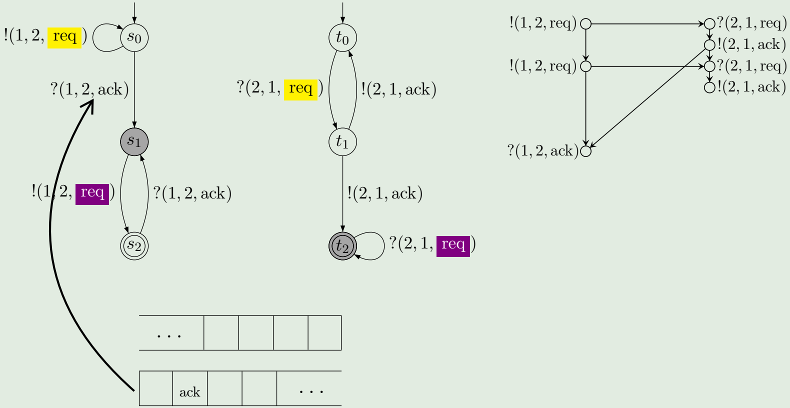




!(1, 2, req) !(1, 2, req) ?(2, 1, req) !(2, 1, ack) ?(2, 1, req) !(2, 1, ack)

Communicating finite-state machines

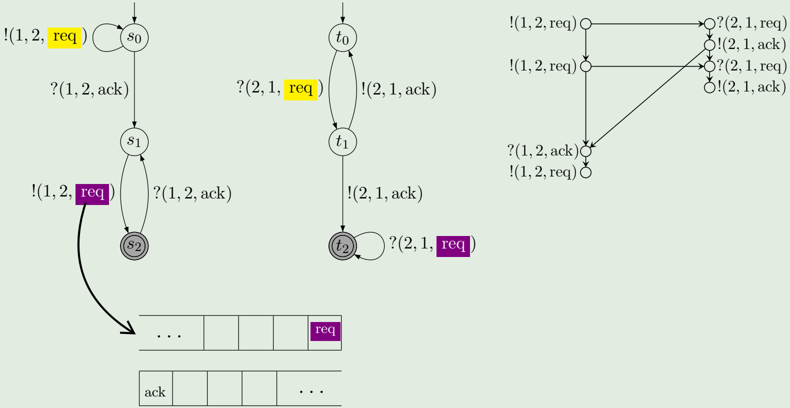
Example



!(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

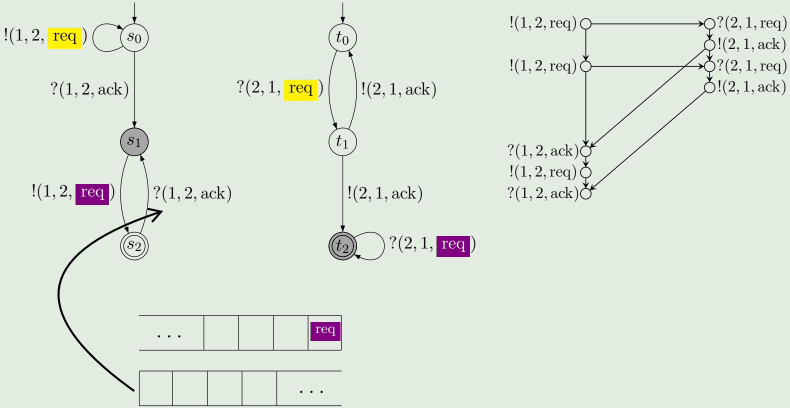
Example



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

Communicating finite-state machines

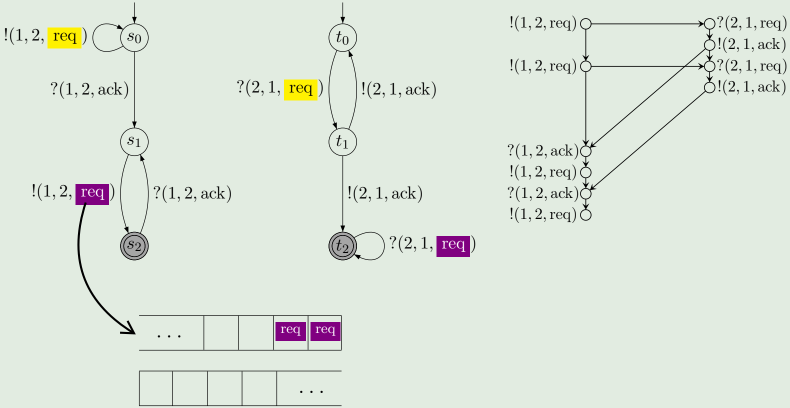
Example



!(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

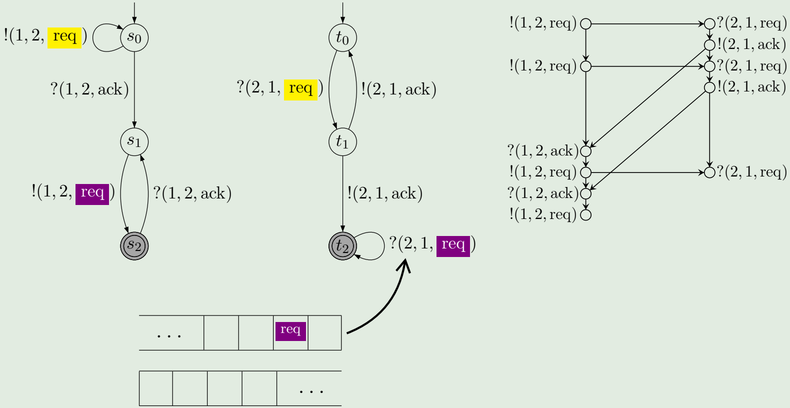
Example



!(1, 2, 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) !(1, 2, req)

Communicating finite-state machines

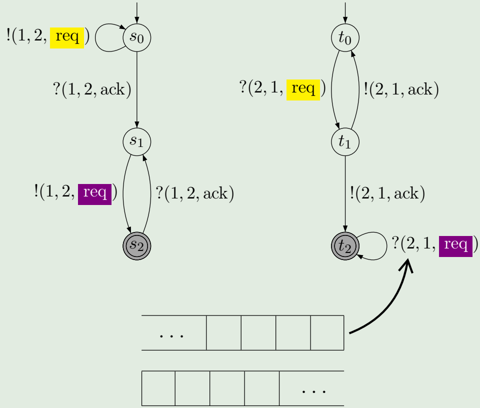
Example

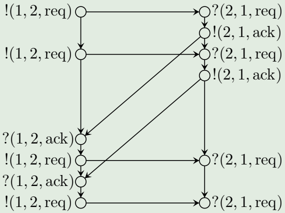


!(1, 2, req) !(1, 2, req) ?(2, 1, req) !(2, 1, ack) ?(2, 1, req) !(2, 1, ack) ?(1, 2, ack) !(1, 2, req) ?(1, 2, ack) !(1, 2, req) ?(2

Communicating finite-state machines

Example

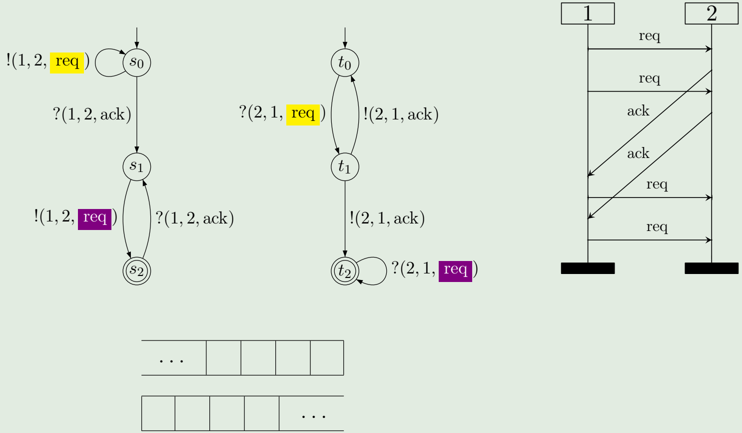




!(1, 2, req) !(1, 2, req) ?(2, 1, req) !(2, 1, ack) ?(2, 1, req) !(2, 1, ack) ?(1, 2, ack) !(1, 2, req) ?(1, 2, ack) !(1, 2, req) ?(2

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) !(1, 2, 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 of $\mathcal{A}$: $Conf_{\mathcal{A}} := S_{\mathcal{A}} \times \{\eta \mid \eta : Ch \rightarrow (\mathcal{C} \times \mathbb{D})^*\}$

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

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 $r \in \mathcal{P} \setminus \{p\}$
- 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

A run of $\mathcal{A}$ on $\sigma_1 \ldots \sigma_n \in Act^*$ is a sequence $\rho = \gamma_0 \, m_1 \, \gamma_1 \ldots \gamma_{n-1} \, m_n \, \gamma_n$ such that

- $\gamma_0 = (s_{init}, \eta_\varepsilon)$ with η_ε mapping any channel to ε
- $\gamma_{i-1} \xRightarrow{\sigma_i, m_i}_{\mathcal{A}} \gamma_i$ 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

A run of $\mathcal{A}$ on $\sigma_1 \ldots \sigma_n \in Act^*$ is a sequence $\rho = \gamma_0 \, m_1 \, \gamma_1 \ldots \gamma_{n-1} \, m_n \, \gamma_n$ such that

- $\gamma_0 = (s_{init}, \eta_\varepsilon)$ with η_ε mapping any channel to ε
- $\gamma_{i-1} \xRightarrow{\sigma_i, m_i}_{\mathcal{A}} \gamma_i$ for any $i \in \{1, \ldots, n\}$

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

A run of $\mathcal{A}$ on $\sigma_1 \ldots \sigma_n \in Act^*$ is a sequence $\rho = \gamma_0 \, m_1 \, \gamma_1 \ldots \gamma_{n-1} \, m_n \, \gamma_n$ such that

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

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

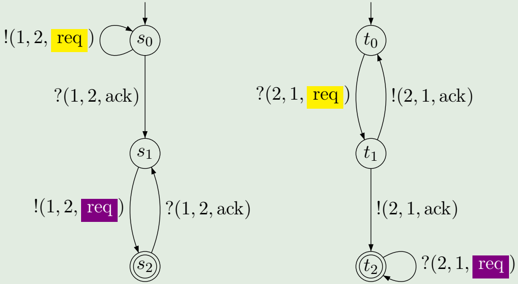
Definition

The set of linearizations of CFM $\mathcal{A}$:

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

Linearizations of an example CFM

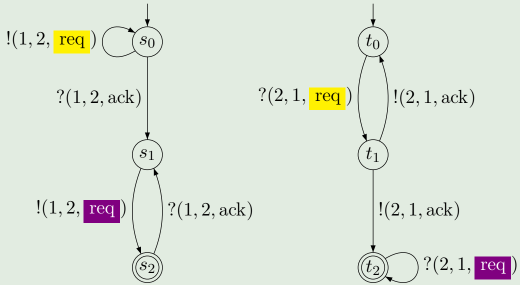
Example



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

Linearizations of an example CFM

Example



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

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

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

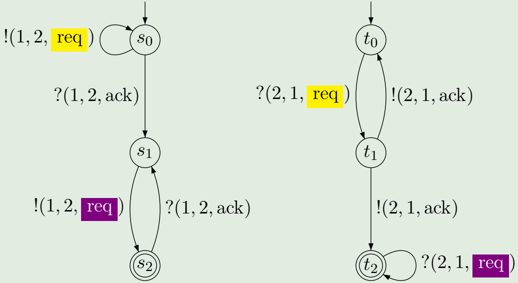
$$w \upharpoonright 2 = (?(2,1,\text{req}) \ !(2,1,\text{ack}))^n \ (?(2,1,\text{req}))^n$$

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



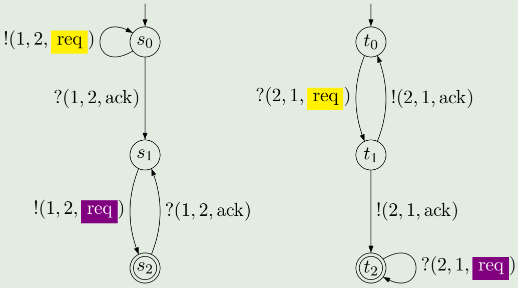
CFM $\mathcal{A}$ over $\{1, 2\}$ and $\{\text{req}, \text{ack}\}$

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

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

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

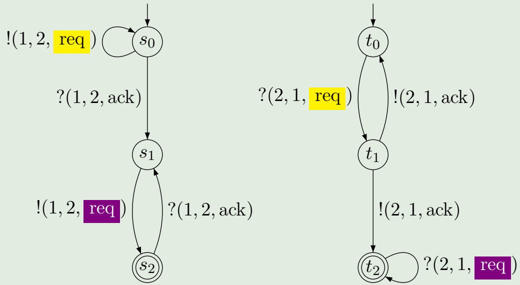
$$w \upharpoonright 2 = (?(2, 1, \text{req}) \; !(2, 1, \text{ack}))^n \; (?(2, 1, \text{req}))^n$$

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 and MSCs of an example CFM

Example



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

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

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

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

Elementary questions are undecidable for CFMs

Proposition ([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 $L(\mathcal{A})$ empty?

Elementary questions are undecidable for CFMs

Proposition ([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 $L(\mathcal{A})$ empty?

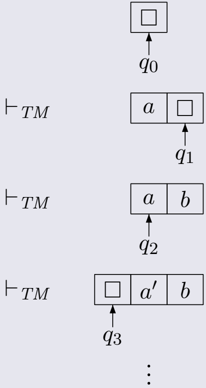
Proof (sketch)

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

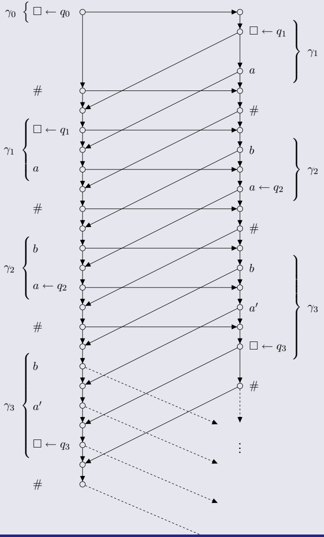
Build CFM $\mathcal{A} = ((\mathcal{A}_1, \mathcal{A}_2), \mathbb{D}, s_{init}, F)$ over $\{1, 2\}$ and some singleton set such that $L(\mathcal{A}) \neq \varnothing$ iff TM can reach q_f.

- Process 1 sends current configurations to process 2
- Process 2 chooses successor configurations and sends them to 1
- $\mathbb{D} = \Big((\Sigma \cup \{\Box\}) \times (Q \cup \{\_\}) \Big) \cup \{\#\}$

Proof (contd.)



q_0
$\vdash_{TM}$
a
q_1
$\vdash_{TM}$
a b
q_2
$\vdash_{TM}$
a' b
q_3



γ_0
$\square \leftarrow q_0$
$\square \leftarrow q_1$
γ_1
a
#
γ_2
b
$a \leftarrow q_2$
γ_3
a'
$\square \leftarrow q_3$

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.

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 $\mathcal{A}$. □