Theoretical Foundations of the UML
Lecture 14: Realising Local-Choice MSGs

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

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

June 9, 2020
Outline

1. Introduction

2. Local Choice MSGs

3. A Realisation Algorithm for MSGs
Overview

1. Introduction

2. Local Choice MSGs

3. A Realisation Algorithm for MSGs
Today’s topic

Today’s lecture

An algorithm to realise local-choice MSGs using CFMs with synchronisation messages.

Results:

1. An algorithm that generates a CFM from local-choice MSGs.
1 Introduction

2 Local Choice MSGs

3 A Realisation Algorithm for MSGs
Non-local choice

Inconsistency if process $p$ behaves according to vertex $v_1$ and process $q$ behaves according to vertex $v_2$

$\implies$ realisation by a CFM may yield a deadlock

Problem:

Subsequent behavior in $G$ is determined by distinct processes. When several processes independently decide to initiate behavior, they might start executing different successor MSCs (= vertices). This is called a non-local choice.
**Definition (Local choice)**

Let $MSG\ G = (V, \rightarrow, v_0, F, \lambda)$. $MSG\ G$ is **local choice** if for every branching vertex $v \in V$ it holds:

$$\exists \text{process } p. \ (\forall \pi \in \text{Paths}(v). \ |\min(\pi')| = 1 \land \min(\pi') \subseteq E_p)$$

where for $\pi = vv_1v_2 \ldots v_n$ we have $\pi' = v_1v_2 \ldots v_n$. 

---

**MSG\ G:**

- $v$
- $w$
- $\pi_1 = vv$
- $\pi_1' = w$
- $\pi_2 = vw$
- $\pi_2' = w$
Local choice property

Definition (Local choice)

Let \( MSG\ G = (V, \rightarrow, v_0, F, \lambda) \). \( MSG\ G \) is local choice if for every branching vertex \( v \in V \) it holds:

\[ \exists \text{ process } p. \ (\forall \pi \in \text{Paths}(v). \ |\min(\pi')| = 1 \land \min(\pi') \subseteq E_p) \]

where for \( \pi = vv_1v_2 \ldots v_n \) we have \( \pi' = v_1v_2 \ldots v_n \).

Intuition:
There is a single process that initiates behavior along every path from the branching vertex \( v \). This process decides how to proceed. In a realisation by a CFM, it can inform the other processes how to proceed.

Local choice or not?
Deciding whether \( MSG\ G \) is local choice or not is in \( P \).
Overview

1. Introduction

2. Local Choice MSGs

3. A Realisation Algorithm for MSGs
Local choice MSGs

An example local-choice MSG on black board.
Realising local choice (C)MSGs

Theorem [Genest et al., 2005]

Any local-choice MSG $G$ is safely realisable by a CFM with synchronisation data (which is of size linear in $G$).

Proof

As MSG $G$ is local choice, at every branch $v$ of $G$ there is a unique process, $p(v)$, say, such that on every path from $v$ the unique minimal event occur at $p(v)$. Then:

1. Process $p(v)$ determines the successor vertex of $v$.
2. Process $p(v)$ informs all other processes about its decision by adding synchronisation data to the exchanged messages.
3. Synchronisation data is the successor vertex (in $G$) from $v$ chosen by $p(v)$.
Structure of the CFM of local choice MSG $G$

Let MSG $G = (V, \rightarrow, v_0, F, \lambda)$ be local choice.

Define the CFM $A_G = (((S_p, \Delta_p))_{p \in P}, D, s_{init}, F')$ with:

- Local automaton $A_p = (S_p, \Delta_p)$ as defined on next slides

\[ \text{pairs } (v, E) \]

\[ \text{downward-closed wrt } <_p \]
Structure of the CFM of local choice MSG $G$

Let MSG $G = (V, \rightarrow, v_0, F, \lambda)$ be local choice.

Define the CFM $A_G = (((S_p, \Delta_p))_{p \in \mathcal{P}}, D, s_{init}, F')$ with:

1. Local automaton $A_p = (S_p, \Delta_p)$ as defined on next slides

2. $D = V$
   
   synchronisation data = vertices in the MSG

3. $s_{init} = \{(v_0, \emptyset)\}^n$ where $n = |\mathcal{P}|$
   
   each local automaton $A_p$ starts in initial state $(v_0, \emptyset)$, i.e.,
   in initial vertex $v_0$ while no events of $p$ have been performed
Structure of the CFM of local choice MSG $G$

Let MSG $G = (V, \rightarrow, v_0, F, \lambda)$ be local choice.

Define the CFM $A_G = (((S_p, \Delta_p))_{p \in \mathcal{P}}, D, s_{init}, F')$ with:

1. Local automaton $A_p = (S_p, \Delta_p)$ as defined on next slides

2. $D = V$

   synchronisation data = vertices in the MSG

3. $s_{init} = \{(v_0, \emptyset)\}^n$ where $n = |\mathcal{P}|$

   each local automaton $A_p$ starts in initial state $(v_0, \emptyset)$, i.e., in initial vertex $v_0$ while no events of $p$ have been performed

4. $s \in F'$ iff for all $p \in \mathcal{P}$, local state $s[p] = (v, E)$ with $E \subseteq E_p$ and:

   1. $v \in F$ and $E$ contains a maximal event wrt. $<_p$ in MSC $\lambda(v)$, or
   2. $v \notin F$ and $\pi = v \ldots w$ is a path in $G$ with $w \in F$ and $E$ contains a maximal event wrt. $<_p$ in MSC $M(\pi)$.
State space of local automaton $A_p$

- $S_p = V \times E_p$ such that for any $s = (v, E) \in S_p$:

\[ \forall e, e' \in \lambda(v). ( e <_p e' \text{ and } e' \in E \text{ implies } e \in E ) \]

That is, $E$ is downward-closed with respect to $<_p$ in MSC $\lambda(v)$.
State space of local automaton $A_p$

- $S_p = V \times E_p$ such that for any $s = (v, E) \in S_p$:
  \[
  \forall e, e' \in \lambda(v). \ (e <_p e' \text{ and } e' \in E \implies e \in E)
  \]
  that is, $E$ is downward-closed with respect to $<_p$ in MSC $\lambda(v)$

- Intuition: a state $(v, E)$ means that process $p$ is currently in vertex $v$ of MSG $G$ and has already performed the events $E$ of $\lambda(v)$

- Initial state of $A_p$ is $(v_0, \emptyset)$
Transition relation of local automaton $A_p$

- Executing events **within a vertex** of the MSG $G$:

  $$e \in E_p \cap \lambda(v) \text{ and } e \notin E$$

  $$(v, E) \xrightarrow{l(e), v} (v, E \cup \{e\})$$

Note: since $E \cup \{e\}$ is downward-closed wrt. $<_p$, $e$ is enabled.
Transition relation of local automaton $A_p$

- Executing events **within a vertex** of the MSG $G$:

  \[
  e \in E_p \cap \lambda(v) \text{ and } e \not\in E
  \]

  \[
  \frac{}{(v, E) \xrightarrow{l(e),v} p (v, E \cup \{ e \})}
  \]

  Note: since $E \cup \{ e \}$ is downward-closed wrt. $<_p$, $e$ is enabled

- **Taking an edge** (possibly a self-loop) of the MSG $G$:

  \[
  E = E_p \cap \lambda(v) \text{ and } e \in E_p \cap \lambda(w) \text{ and }
  \]

  \[
  v u_0 \ldots u_n w \in V^* \text{ with } p \text{ not active in } u_0 \ldots u_n
  \]

  \[
  \frac{}{(v, E) \xrightarrow{l(e),w} p (w, \{ e \})}
  \]

  Note: vertex $w$ is the first successor vertex of $v$ on which $p$ is active
Taking an edge (possibly a self-loop) of the MSG $G$:

\[
E = E_p \cap \lambda(v) \text{ and } e \in E_p \cap \lambda(w) \text{ and } \\
uu_0 \ldots uu_n w \in V^* \text{ with } p \text{ not active in } uu_0 \ldots uu_n \\
(v, E) \xrightarrow{l(e), w} p (w, \{e\})
\]

Note: vertex $w$ is the first successor vertex of $v$ on which $p$ is active

all events that $p$ executed in vertex $v$

1. MSG

2. MSG

synchronisation data
On the black board.
Local automata \( A_1, A_2, A_3 \)

\[ \text{\( A_1: \)} \quad V_0, \varnothing \xrightarrow{!(1,2,a)} V_0, \{e_1\} \]

\[ \text{\( V_0, \{e_1\} \xrightarrow{?(1,2,c)} V_3, \{e_8\} \)} \]

\[ \frac{e \in E_p \cap \lambda(v) \text{ and } e \notin E}{(v, E) \xrightarrow{l(e),w} \gamma_p (v, E \cup \{e\})} \]

**Taking an edge** (possibly a self-loop) of the MSG \( G \):

\[ E = E_p \cap \lambda(v) \text{ and } e \in E_p \cap \lambda(w) \text{ and } \]

\[ vu_0 \ldots u_n w \in V^* \text{ with } p \text{ not active in } u_0 \ldots u_n \]

\[ (v, E) \xrightarrow{l(e),w} \gamma_p (w, \{e\}) \]

Note: vertex \( w \) is the first successor vertex of \( v \) on which \( p \) is active

\[ V_0 \xrightarrow{e_8} V_3 \]

\[ V_0 \xrightarrow{\text{process 1 is inactive}} V_2 \xrightarrow{V_3} \]
local automaton $A_2$

$\rightarrow v_0, \emptyset \xrightarrow{? (2,1,a)} v_0, \{e_2\} \xrightarrow{! (2,3,a)} v_1, \{e_3\}$

$\xrightarrow{! (2,3,b)} v_2, \{e_5\} \xrightarrow{! (2,1,c)} v_3, \{e_7\}$

$\frac{e \in E_p \cap \lambda(v) \text{ and } e \notin E}{(v, E) \xrightarrow{l(e), v}_p (v, E \cup \{e\})}$

local automaton $A_3$

$\rightarrow v_0, \emptyset \xrightarrow{? (3,2,a)} v_1, \{e_4\}$

$\xrightarrow{? (3,2,b)} v_2, \{e_6\}$
Second example

A \xrightarrow{reg} e_1 \rightarrow e_2

B \xrightarrow{ack} e_3

regular expression = A \cdot (A + B)^*

MSG

Applying the realiziation construction yields \( A_1, A_2 \)

local automaton \( A_1 \):

\( V_0, \emptyset \) \rightarrow \( V_0, \{ e_1 \} \) \rightarrow \( V_0, \{ e_1 \} \) \rightarrow \( V_0, \{ e_1 \} \) \rightarrow \( V_0, \{ e_1 \} \) \rightarrow \( V_0, \{ e_1 \} \) \rightarrow \( V_0, \{ e_1 \} \)
local automaton $A_2$: