Software Modeling and Verification
Lehrstuhl für Informatik 2
RWTH Aachen University

Prof. Dr. Ir. J.-P. Katoen
C. Kern

Foundations of the UML

Winter Term 07/08

– Lecture 3: Compositional MSGs –

(14.11.2007)

summarized by *Aleksandar Bojinović* and *Leonid Pishchulin*

Message Sequence Graphs

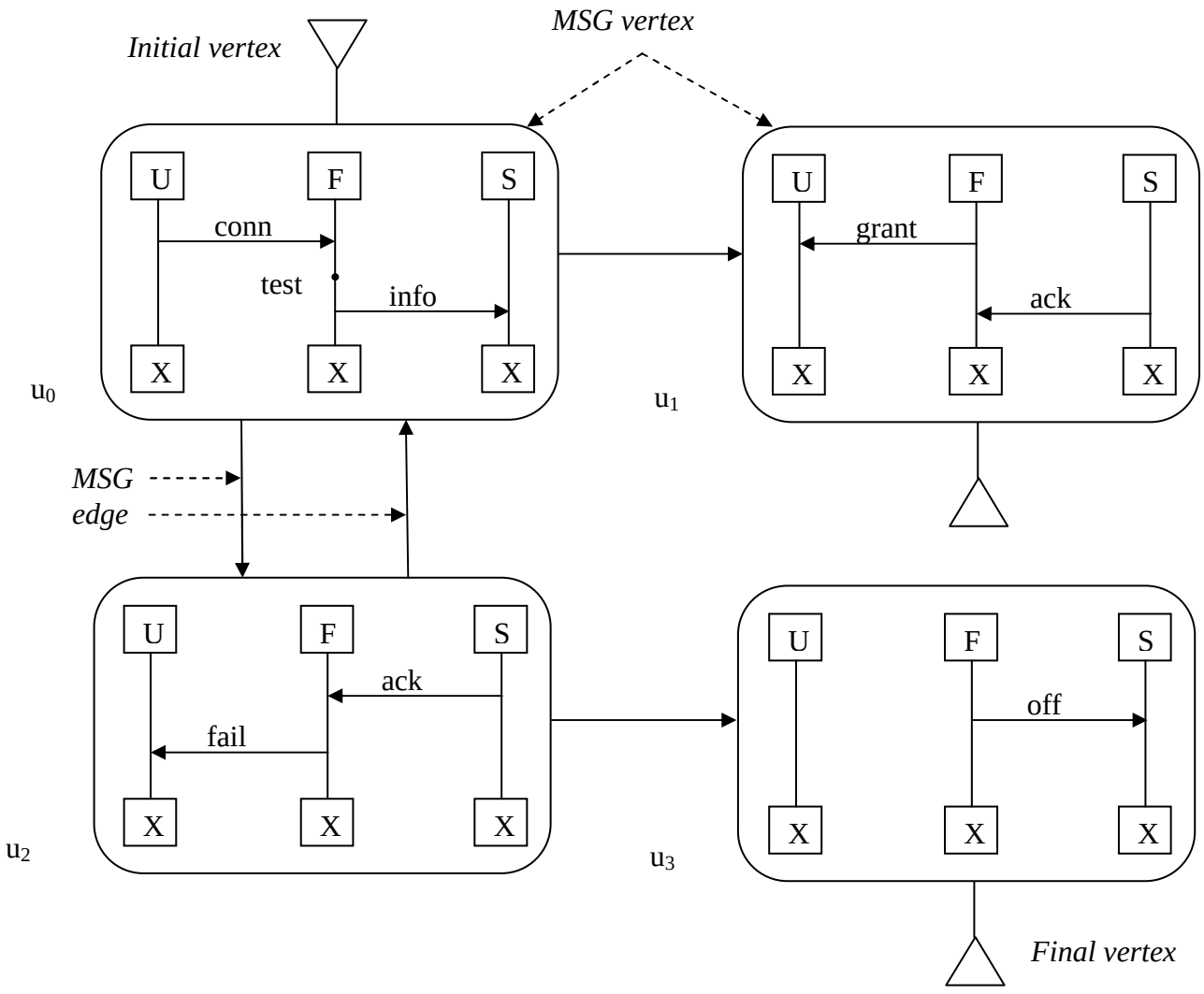


Figure 3.1 – *Message Sequence Graph*

Definition

Let $\mathbb{M}$ be the set of MSCs *(up to isomorphism i.e. event identities).*

A ***Message Sequence Graph (MSG)*** G is a tuple $\mathbf{G = (V, \rightarrow, v_0, F, \lambda)}$ with:

- $(V, \rightarrow)$ is a digraph with finite set V of vertices and $\rightarrow \subseteq V \times V$
- $v_0 \in V$ is *starting* (or *initial*) *vertex*
- $F \in V$ is a set of *final vertices*
- $\lambda: V \rightarrow \mathbb{M}$ associates to each vertex $v \in V$, an MSC $\lambda(v)$

Note:

1. An MSG is an NFA without input alphabet where states are MSCs
2. Every MSC is an MSG
3. Generalization towards a *set* of initial vertices is straightforward

Concatenation of MSCs

Let $M_i = (P_i, E_i, C_i, l_i, m_i, <_i)$, $i \in \{1,2\}$, be an MSC with $E1 \cap E2 = \varnothing$

The ***concatenation*** of M_1 and M_2 is the MSC

$M_1 \bullet M_2 = (P, E, C, l, m, <)$ with:

$$P = P_1 \cup P_2,\ E = E_1 \cup E_2,\ C = C_1 \cup C_2 \quad (\text{with } E? = E_{1?} \cup E_{2?} \text{ etc.})$$

$$l(e) = \begin{cases} l_1(e) & \text{if } e \in E_1 \\ l_2(e) & \text{if } e \in E_2 \end{cases} \qquad m(e) \begin{cases} m_1(e) & \text{if } e \in E_1 \\ m_2(e) & \text{if } e \in E_2 \end{cases}$$

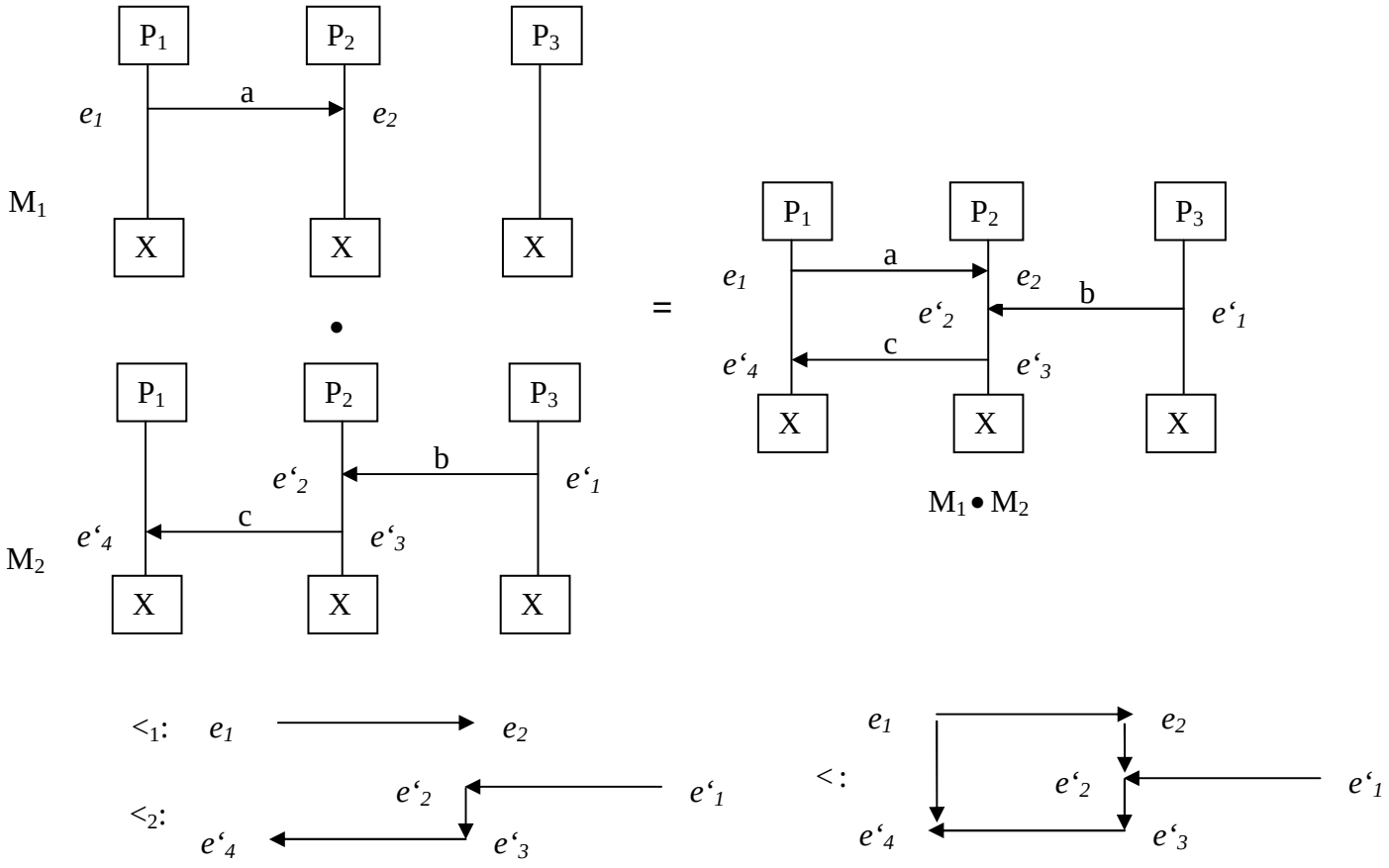
$$< = <_1 \cup <2 \cup \{(e1, e') \mid e \in E_1 \cap E_p,\ e' \in E_2 \cap E_p\}$$

same process

Note:

- Events are ordered process-wise, events at p in M_1 precede events at p in M_2
- Thus: some processes may proceed to M_2 before others!
- $\neq$: first complete M_1, then execute M_2

Example



Note:
e_1 and e'_1 are not ordered in $M_1 \bullet M_2$

e.g. $e_1\, e_2\, e'_1\, e'_2 \ldots \in \mathrm{Lin}(M_1 \bullet M_2)$

$e'_1\, e_1\, e_2\, e'_2 \ldots \in \mathrm{Lin}(M_1 \bullet M_2)$

Figure 3.2 – *Example from the previous class*

Language of an MSG

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

A ***path*** Π of G is a finite sequence

$$\Pi = u_0\, u_1 \ldots u_n \qquad \text{with} \quad u_i \in V,\ 0 \le i \le n$$

$$u_i \rightarrow u_{i+1},\ 0 \le i \le n$$

The ***MSC of a path*** $\Pi = u_0\, u_1 \ldots u_n$ is:

$$M(\Pi) = \lambda(u_0) \bullet \lambda(u_1) \bullet \ldots \bullet \lambda(u_n) = \prod_{i=0}^{n} \lambda(u_i)$$

MSC concatenation

Path $\Pi = u_0\, u_1 \ldots u_n$ is ***accepting*** if: $u_0 = v_0$ and $u_n \in F$

The ***(MSC) language*** of MSC G is defined by:

$$L(G) = \{M(\Pi) | \ \Pi \text{ is accepting path of } G\}$$

The ***word language*** of MSG G is Lin(L(G)) where

$$\text{Lin}(\{M_1, \ldots, M_k\}) = \bigcup_{i=1}^{k} Lin(M_i)$$

Expressiveness

- The decision problem "MSG G has a race" is undecidable
- MSGs may represent infinite-state systems
- State space of an MSG may not be CF
- State space of an MSG is context-sensitive
- The decision problem: for MSGs G1 and G2, do we have $L(G1) \cap L(G2) = \varnothing$ is undecidable.
- ***Corollary :*** the decision problem for MSG G and DFA A, do we have $\text{Lin}(L(G)) \cap L(A) = \varnothing$ is *undecidable.*

Local Choice Property

Example 1:

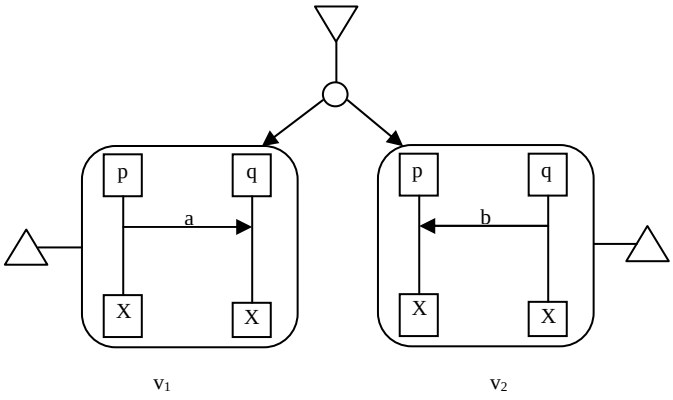


Figure 3.3 – *Local choice property example*

Inconsistency if p behaves according to V_1 and q behaves according to V_2

⇨ possible distributed realization may yield a *deadlock* (this may be made more precise in next two lectures)

Problem: subsequent behavior is determined by distinct processes

Example 2:

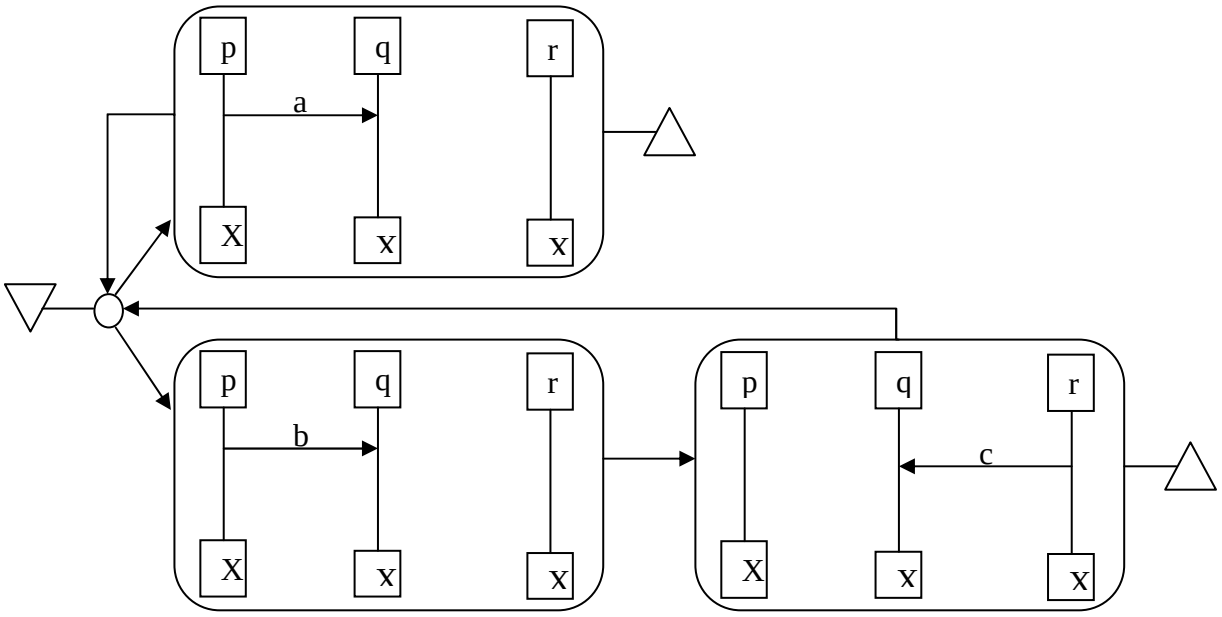


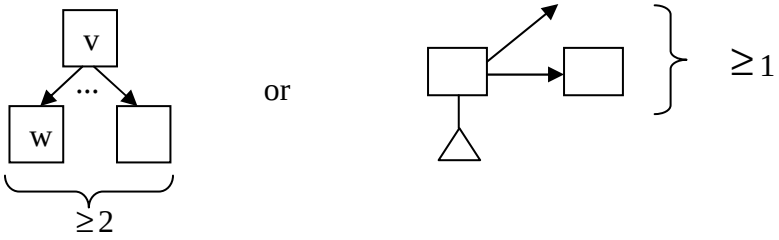
Figure 3.4 - *Local choice property example*

Local Choice Property

- *e* is a minimal event wrt. < if ¬ ($\exists e', e' < e$)
- p is active in MSC M if $E_p \neq \emptyset$, p is active in path $v_1\ v_2 \ldots v_n$ in MSG G if

$$\exists\ v_i, \text{ p is active in } \lambda(v_i) \qquad 0 \leq i \leq n$$

- ***Definition:*** MSG G = (V, →, v_0, F, λ) is ***local choice*** if:
  1. $\exists$ active p, $\forall\ \Pi \in$ Paths(v_0), Π has a single minimal event *e* with $e \in E_p$
  2. $\forall$ branching vertex v∈ V



$\exists$ active p, $\forall$ $\Pi \in$ Paths(w), and v $\rightarrow$ w, Π has a single minimal event *e* with *e* $\in E_p$

Intuition: Along every path from an initial or branching vertex there is a single process deciding how to proceed which can inform the other processes how to proceed.

Example

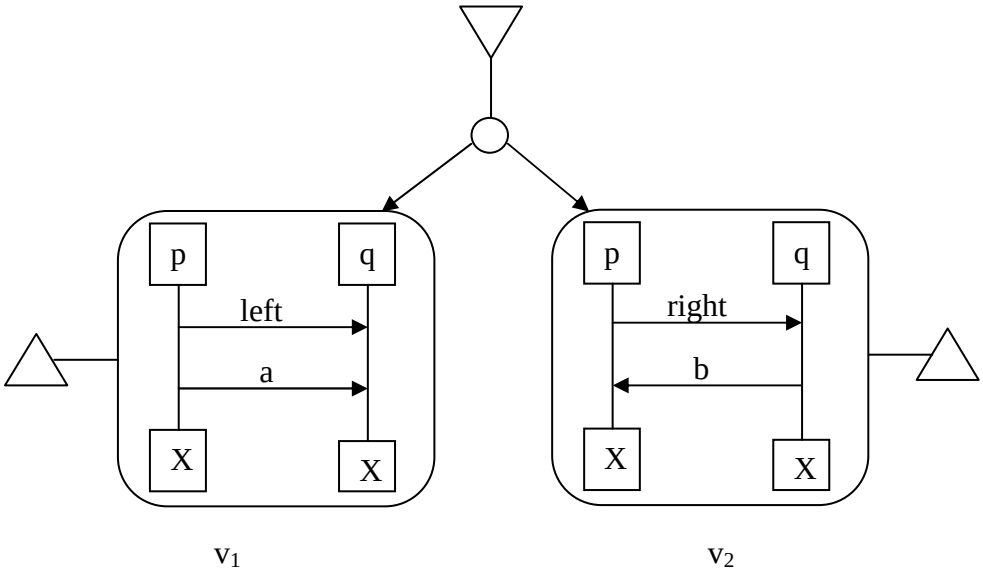
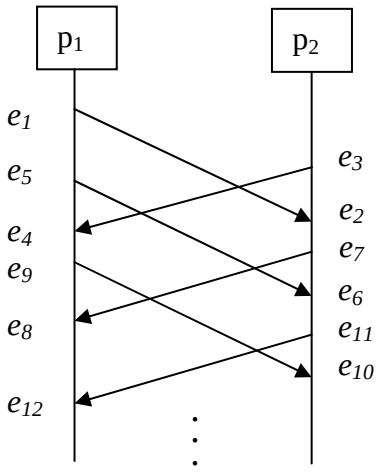


Figure 3.5 – *Local choice property*

Checking whether an MSG is local choice can be done in polynomial time.

How can non-local choices be resolved? Refine your MSG and add control messages (cf. above example).

Consider the following example, proposed by Yannakakis:



This MSC ***cannot*** be decomposed as $M_1 \bullet M_2 \bullet \ \ldots \ \bullet M_n$ (n>1)

This can be seen as follows:

- e_1 and $e_2 = m(e_1)$ must reside in same M_i
- $e_3 < e_2$ and $e_1 < e_4$ thus $e_3, e_4 \notin M_j$, $j < i$ or $j > i$ $\Rightarrow$ $e_3, e_4 \in M_i$
- by similar reasoning: $e_5, e_6 \in M_i$ etc.

Problem: compulsory matching between send and receive events in the *same* MSG vertex (i.e. send and receive m(e))

Compositional MSGs

Solution: drop restriction that e and m(e) belong to the same MSC (= allow for incomplete message transfer)

Definition: M = (P, E, C, *l*, *m*, <) is a ***compositional MSC***, where P, E, C and *l* are as before, and

- $m: E_! \rightarrow E_?$ is a partial, injective function (*not a bijection!*) such that (as before):

$$m(e) = e' \ \wedge \ l(e) = p!q(a) \ \rightarrow \ l(e') = q?p(a)$$

- $< = \bigcup_{p \in P} <_p \ \cup \{(e, m(e)) \mid e \in dom(m)\}$ (*dom(m) – domain of m, "m(e) is defined"*)

Note: an MSC is a CMSC where *m* is total and bijective.

CMSC Example

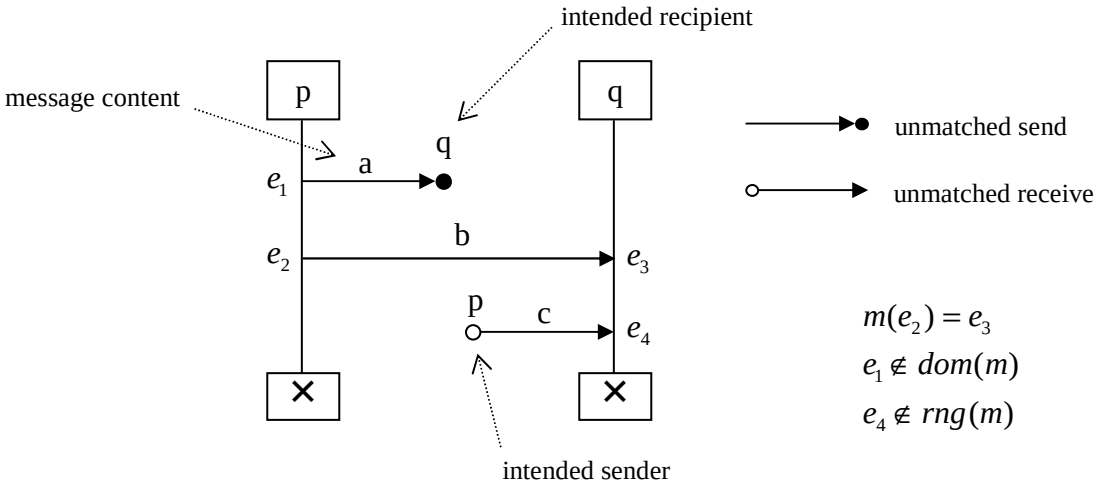


Figure 3.6 – *CMSC example*

Definition: A compositional MSG (CMSG) is a MSG $G = (V, \rightarrow, v_0, F, \lambda)$ with $\lambda : V \rightarrow \mathbb{CM}$, where $\mathbb{CM}$ is the set of CMSCs and $V, \rightarrow, v_0$ and F are as before.

Thus, in CMSG we allow vertices to be labeled with compositional message sequence charts (CMSCs).

Concatenation of CMSCs

Let $M_i = (P_i, E_i, C_i, l_i, m_i, <_i) \in \mathbb{CM}$ and $i \in \{1,2\}$, $E_1 \cap E_2 = \varnothing$.

Then $M_1 \cdot M_2 = (P_1 \cup P_2, E_1 \cup E_2, C_1 \cup C_2, l, m, <)$ with:

- $l(e) = l_1(e)$ if $e \in E_1$, $l(e) = l_2(e)$ otherwise
- $m : E_! \rightarrow E_?$ satisfies:
  1. m extends m_1 and m_2. In other words if e is in the domain of m_i $e \in dom(m_i)$, where $i \in \{1,2\}$ then $m(e) = m_i(e)$.
  2. m matches unmatched send events in M_1 with unmatched receive events in M_2 according to order on processes, matching them (events) from top to bottom.
     (In other words, if we consider the *k-th* unmatched send in M_1 then it will be

matched with the *k-th* unmatched receive in M_2, where of course the message content is the same and the sender and the receiver match.)

3. $M_1 \cdot M_2$ satisfies the FIFO condition, when restricted to match events.

- $<= = \left(\bigcup_{p \in P} <_{p1} \cup <_{p2} \right) \cup \{(e, e') \mid e \in E_1 \cap E_p, e' \in E_2 \cap E_p\} \cup \{(e, m(e)) \mid e \in dom(m)\}$

Examples

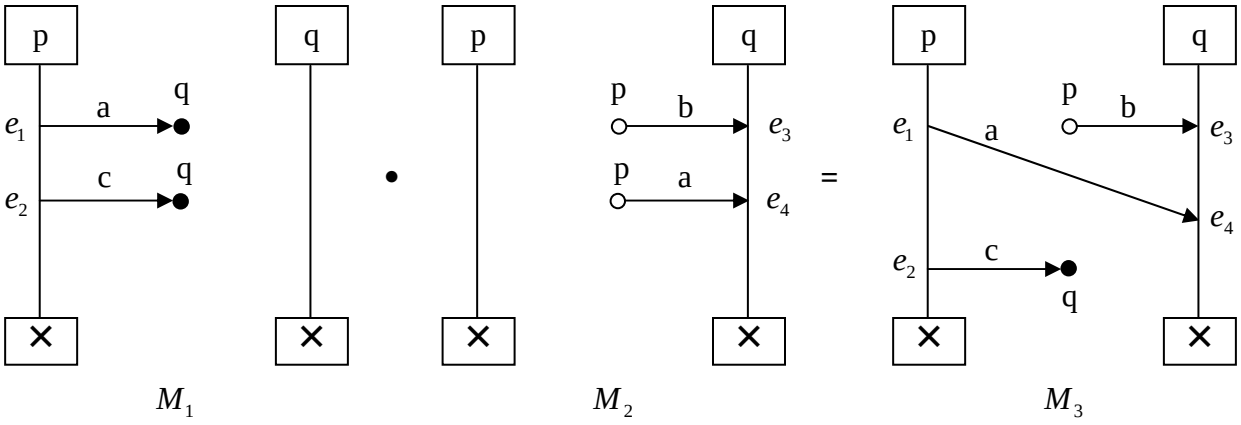


Figure 3.7 – *Example of CMSC concatenation*

In the example shown in the figure above (
Figure 3.7) the process *p* in the CMSC M_1 is first sending message *a* and later message *c* to the process *q*. In CMSC M_2 message *a* can be received by process *q*, as it is matched with the same message sent before by process *p*. However, messages *b* and *c* do not match. Message *b* cannot be matched as process *p* has not sent this message before, and message *c* cannot be matched as process *q* does not expect to receive this message from process *p*.

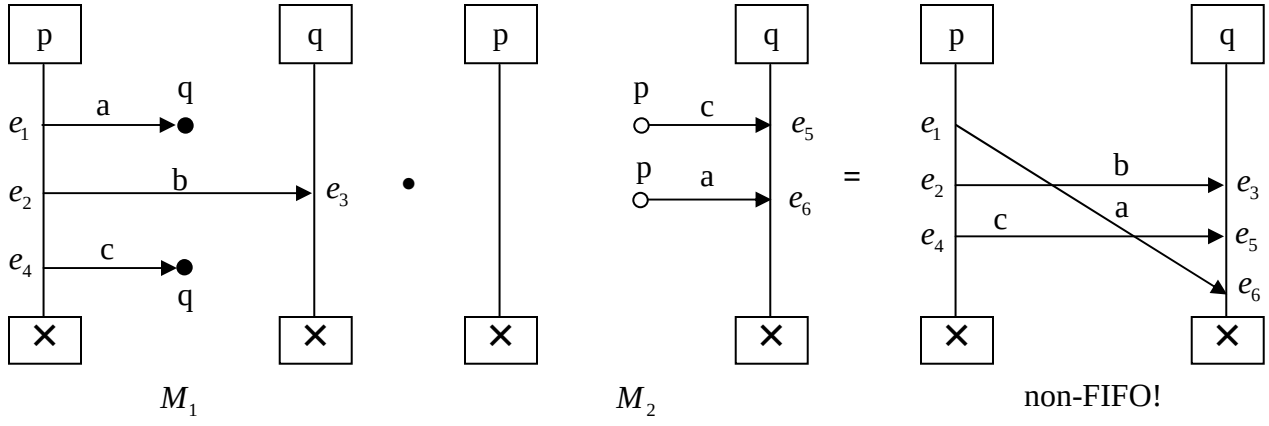


Figure 3.8 – *Example of CMSC concatenation*

The resulting CMSC shown in

Figure 3.8 violates the FIFO property. First process *p* from CMSC M_1 sends message *a* to the process *q* in e_1. This message can be received by *q* in CMSC M_2 as early as e_6. Process *p* sends message *c* to the process *q* in e_4. This message can be received by *q* in e_5. Sending of the message *b* is straightforward. Now, we also have the following order of events in M_1: $e_1 \to e_2, e_2 \to e_3, e_2 \to e_4$, in M_2: $e_5 \to e_6$ and from concatenation of two CMSCs: $e_3 \to e_5$. All of this results in the CMSC shown in the figure above.

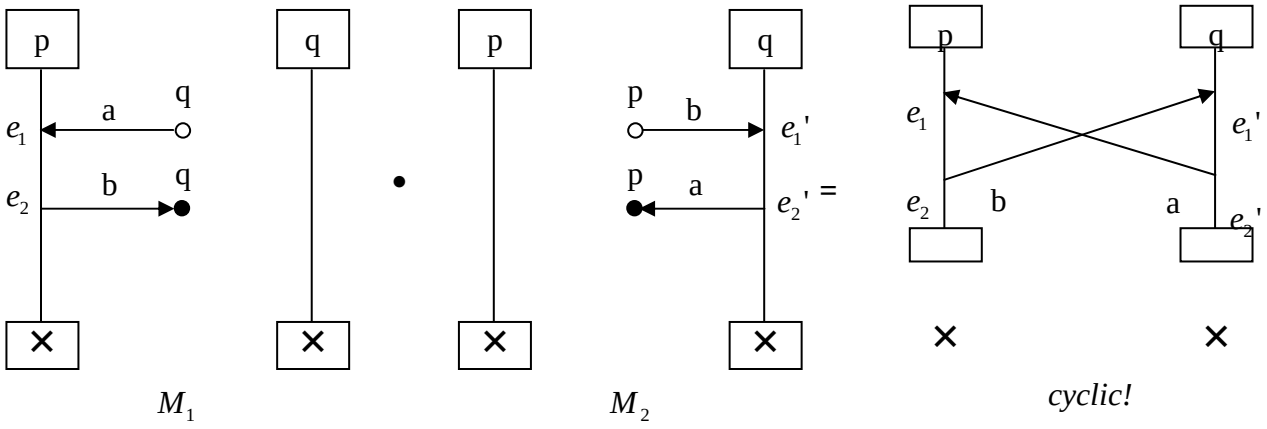
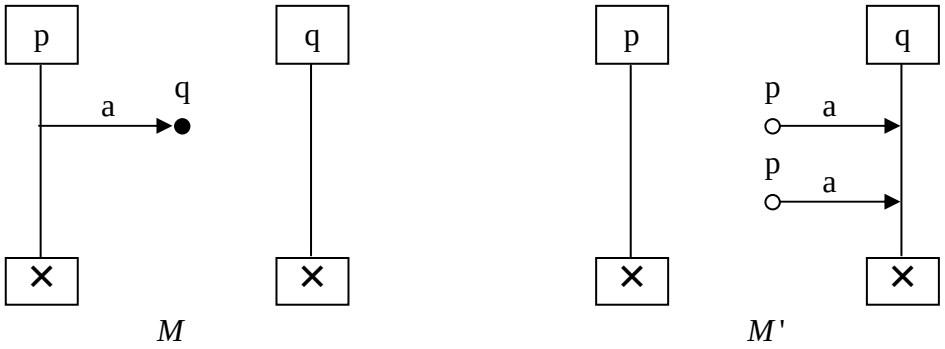


Figure 3.9 – *Example of CMSC concatenation*

Going from top to bottom, we match first e_1 with e_2' and then e_2 with e_1'. The resulting CMSC is shown in the Figure 3.9 and it is cyclic.

Associativity

There are two CMSCs M and M' as shown in the figure below.



If we try to calculate the concatenation of the expression $(M \cdot M) \cdot M'$ then we will get the CMSC as shown in the Figure 3.10.

$(M \cdot M) \cdot M'$:

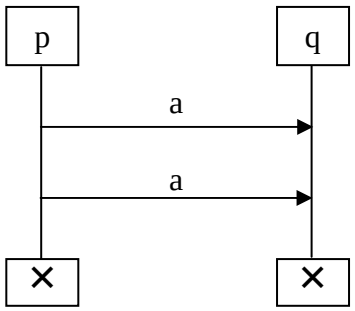


Figure 3.10 – *Resulting CMSC*

In an other case, if we try to evaluate the expression $M \cdot (M \cdot M')$, we will get the CMSC as shown in the Figure 3.11. That CMSC violates FIFO property and therefore is undefined.

$M \cdot (M \cdot M')$:

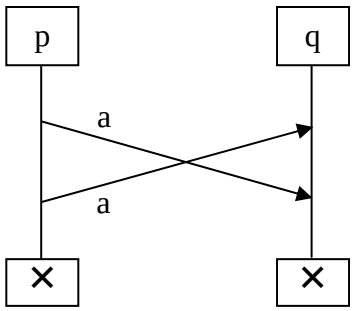


Figure 3.11 – *Resulting CMSC*

Conclusion: Concatenation of CMSCs is **not** associative.

Language of a CMSG

Let $G = (V, \rightarrow, v_0, F, \lambda)$ be a CMSG.

- A *path* π of *G* is a finite sequence $\pi = u_0 u_1 \ldots u_n$ with $u_i \in V$ and $u_i \rightarrow u_{i+1}$.
- The *CMSC of a path* $\pi = u_0 u_1 \ldots u_n$ is:

  $$M(\pi) = (\ldots(\lambda(u_1) \cdot \lambda(u_2)) \cdot \lambda(u_3) \ldots) \cdot \lambda(u_n) = \prod_{i=0}^{n} \lambda(u_i)$$

  Note: Symbol '·' denotes CMSC concatenation, which is left-associative.
- Path $\pi = u_0 u_1 \ldots u_n$ is *accepting* if $u_0 = v_0$ and $u_n \in F$.
- The *(MSC) language* of CMSG *G* is defined by:
  $L(G) = \{M(\pi) \in M \mid \pi \text{ is an accepting path of } G\}$
  Note: We consider all the accepting paths of MSCs only.
- *G* is *safe* if for every accepting path π of *G*, $M(\pi)$ is an MSC. In other words, there are no accepting paths for unmatched sends and receives.

Consider again

As seen before, we could not decompose the example Yannakakis proposed in a concatenation of MSCs. However, the proposed example can be modeled in terms of CMSG as shown in the Figure 3.12.

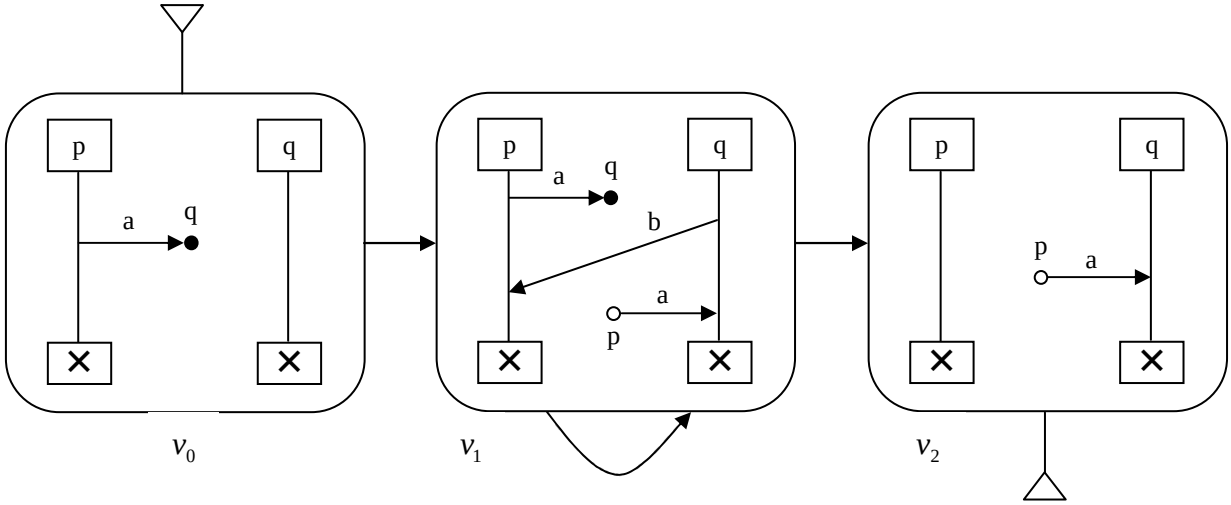


Figure 3.12 – *CMSG for the Yannakakis' MSC*

Some properties of CMSGs

- CMSGs are strictly more expressive than MSGs.
- There are subclasses of CMSGs that are equally expressive to MSGs, the so-called *locally safe* CMSGs.
- If there is a path π of CMSG *G* such that $M(\pi) \in \mathbb{M}$ then we call it a *safe* path.
- The decision problem "does G have at least one safe, accepting path?" is *undecidable*.
- The decision problem "is CMSG safe?" is *decidable* in polynomial time.

Existence of safe paths

The decision problem:

Input: CMSG *G*

Output: Yes, if *G* has a safe, accepting path
No, otherwise

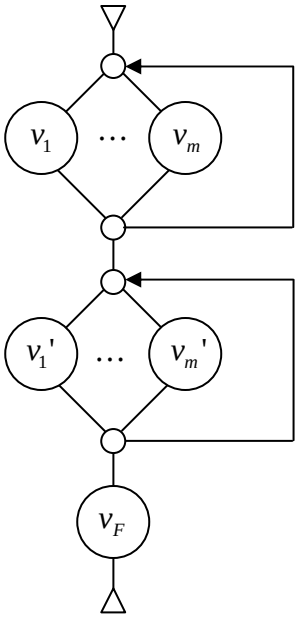
is undecidable.

Proof: Reduction from the Post's Correspondence Problem (PCP).

- Let PCP *(u, w)* over alphabet Σ with $U = u_1, u_2, \ldots, u_m$ and $W = w_1, w_2, \ldots, w_m$, where $u_i, w_i \in \Sigma^*$, i.e. instance $\langle (u_1, w_1), \ldots, (u_m, w_m) \rangle$.
- Construct a CMSG $G_{u,w}$ such that PCP *(u, w)* has a solution if and only if CMSG *G* has an accepting, safe path.
- PCP is undecidable and the above statement implies: the above decision problem is undecidable.

$G_{u,w}$ Let:

- $P = \{P_1, P_2, P_3, P_4\}$
- $C = \Sigma \cup \{finish\} \cup \{1,2,\ldots,m\}$
- $G = (v_1 \mid v_2 \mid \ldots \mid v_m)^+ \cdot (v_1' \mid v_2' \mid \ldots \mid v_m')^+ \cdot v_F$
- $F = \{v_F\}$
- $V_0 = \{v_1, v_2, \ldots, v_m\}$
- $\lambda(v_i) =$ CMSC over u_i
- $\lambda(v_i') =$ CMSC over w_i

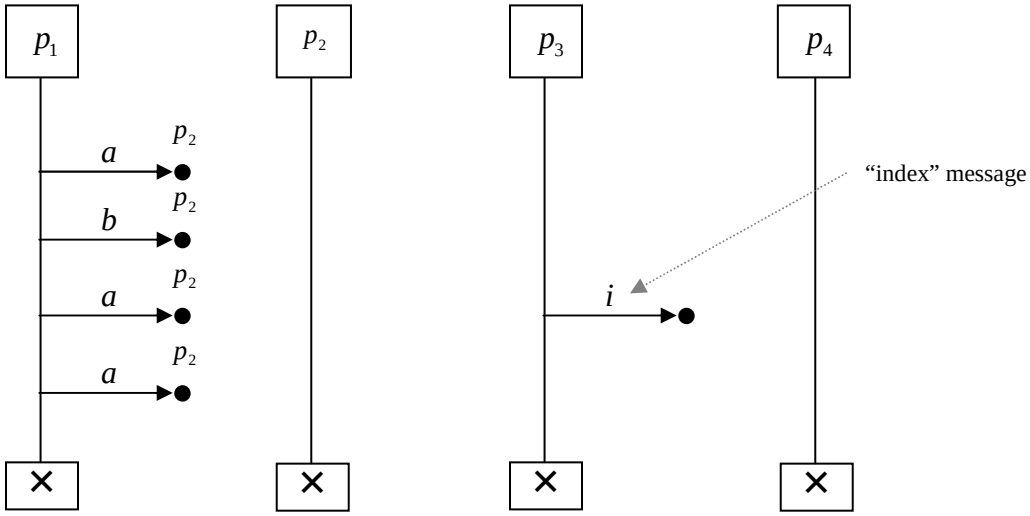


Example:

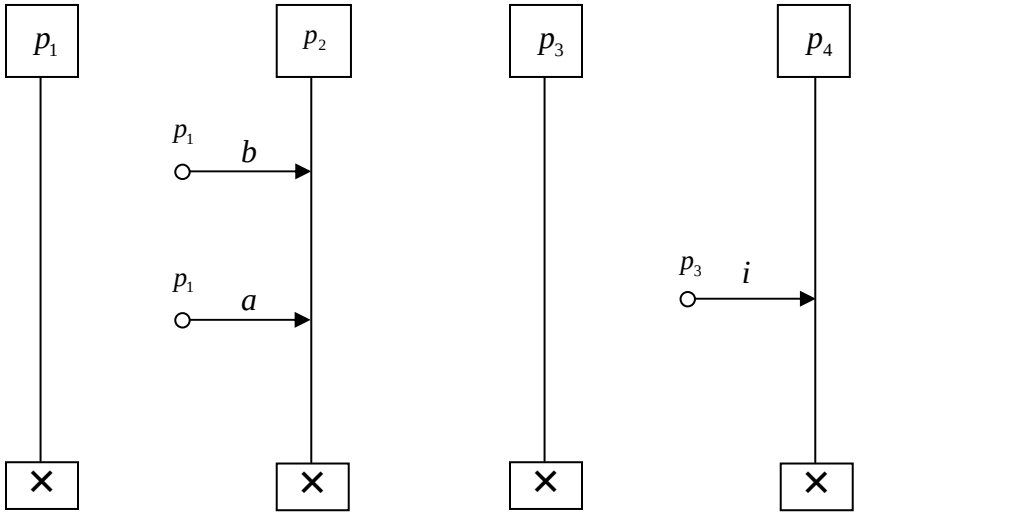
$\Sigma = \{a, b\}$

$u_i = abaa$

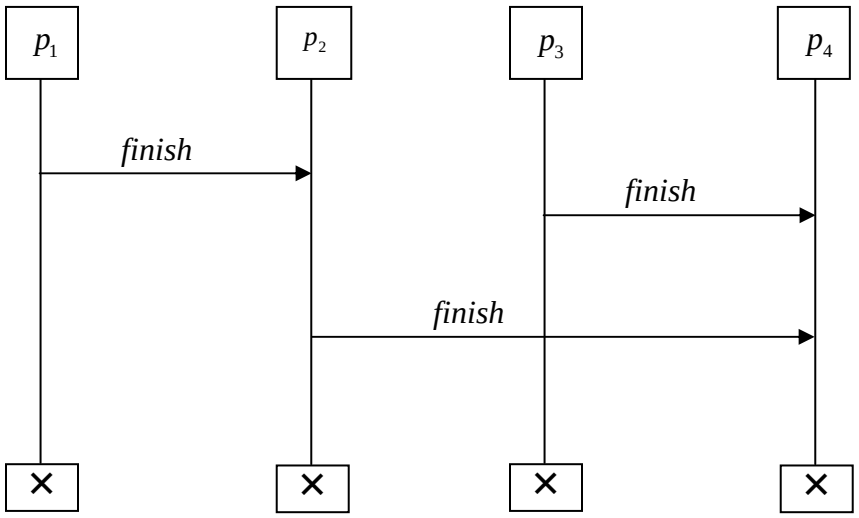
$\lambda(v_i) =$



$\lambda(v_i') =$



$\lambda(v_F) =$



Claim: $PCP(u,w)$ has a solution if and only if $G_{u,w}$ has a safe, accepting path.

Proof:

"$\Rightarrow$"

Let $i_1, i_2, \ldots, i_n$ be a solution of *(u,w)*. Then consider path π in $G_{u,w}$:

$\pi = v_{i_1} \ldots v_{i_n} \cdot v_{i_1}' \ldots v_{i_n}' \cdot v_F$ is accepting and $\prod_{j=i_1}^{i_n} \lambda(v_{i_j}) \cdot \prod_{j=i_1}^{i_n} \lambda(v_{i_j}') \cdot \lambda(v_F)$ is an MSC.

"$\Leftarrow$"

Let $\pi = v_{i_1} \ldots v_{i_n} \cdot w_{j_1} \ldots w_{j_k} \cdot v_F$ be a safe, accepting path of $G_{u,w}$.

Observe:

1. $p_2 ? p_1(finish)$ occurs in $v_F \Rightarrow$ all unmatched sends in p_1 are matched by unmatched receives in p_2.
   $\Rightarrow$ Number of sends $p_1 ! p_2(\ldots)$ is equal to number of receives $p_2 ? p_1(\ldots)$
   $\Rightarrow n = k$
2. $p_4 ? p_3(finish)$ in $v_F \Rightarrow$ all unmatched "index" message sends are matched

π is safe $\Rightarrow M(\pi) \in \mathbb{M}$, but then:

$$i_1 = j_1, i_2 = j_2, \ldots, i_n = j_n$$

$\Rightarrow i_1, i_2, \ldots, i_n$ is a solution of *PCP(u,w)*.

Safeness of CMSGs

The decision problem "is CMSG *G* safe?" is decidable in polynomial time.

- Let's consider a pair of processes (p_i, p_j)
- Construct a push-down automaton $K_{i,j} = (Q, \Gamma, \Sigma, \Delta)$ with:
  - *Q* a finite set of control states
  - Γ = *stack alphabet* = $\{1, \perp\}$, with 1 indicating *counter* and $\perp$ indicating *stack bottom*
  - $\Sigma = \left\{ \begin{matrix} unmatched\ !\ (a) \\ unmatched\ ?\ (a) \\ unmatched\ (a) \end{matrix} \right\}$, for $a \in C$
  - $\Delta \subseteq (Q \times \Sigma \times \Gamma) \times (Q \times \{push, pop, skip\})$
    (q, a, γ, q', pop) means:
    On reading *a* and top stack is γ in state *q*, change to *q'* and *pop* γ.
  - Accept if p_i and p_j are completed and stack is non-empty or if unmatched $p_i ! p_j(a)$ is matched by $p_j ? p_i(b)$ with $a \neq b$.

Functioning of PDA $K_{i,j}$:

Phase 1: Replace vertex v in *G* by a linearization of $\lambda(v)$ such that all unmatched receive events precede all unmatched send events of same type. (Same type means that the sender and receiver match and that message content matches as well).
$K_{i,j}$ follows events in $\lambda(v)$ and continues nondeterministically to $\lambda(w)$ for some direct successor *w* of *v* in *G*.

Phase 2: Push *1* on stack if $p_i ! p_j(a)$ is unmatched, pop *1* if $p_j ? p_i(a)$ is not matched. On occurrence of $p_i ! p_j(a)$ move to control state $q_a \in Q$ and remember what message is not matched, ignore all events except unmatched receipts (*pop!*); if the stack is empty, compare message content of last receive: if $a \neq b$ → accept, otherwise ignore rest of events and continue further.

Push-down automaton (PDA) $K_{i,j}$ accepts if and only if CMSG *G* is not safe with respect to pair (p_i, p_j).

- Checking safeness amounts to checking reachability in all PDAs $K_{i,j}$.
- Reachability of a configuration in a PDA is decidable in polynomial time; hence checking safeness can be done in polynomial time.

Note: In above construction we have assumed (for simplicity) that each safe path is also well-formed (see the following heading). Without this assumption, the PDA $K_{i,j}$ needs to be adapted (*how?*).

Well-formed paths

Let CMSC M be well-formed if:

- $E_? = rng(m)$
- For all $e, e' \in E$:
  $l(e) = p!q(a) \wedge e \notin dom(m) \wedge l(e') = l(e) \wedge e' \in dom(m)$ implies $e' <_p e$

Path $v_1 \ldots v_k$ in CMSG G is well-formed if for all $n \leq k$, $\prod_{c=1}^{n} \lambda(v_i)$ is a well-formed CMSC.

■