Concurrency Theory

Winter 2025/26

Lecture 17: Petri Net Semantics of CCS

Thomas Noll, Peter Thiemann
Programming Languages Group
University of Freiburg

https://proglang.github.io/teaching/25ws/ct.html

Thomas Noll, Peter Thiemann

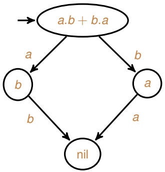
Winter 2025/26

Outline of Lecture 17

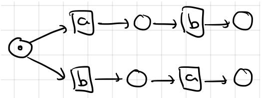
Motivation

Goal: Define true concurrency semantics for (a subset of) CCS

- Distinguish between $+$ and $\parallel$
  - $a.b.\mathrm{nil} + b.a.\mathrm{nil}$:

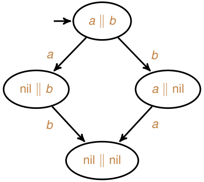


LTS

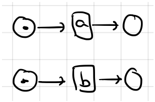


Net

  - $a.\mathrm{nil} \parallel b.\mathrm{nil}$:



LTS



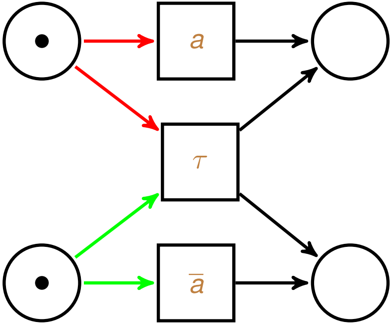
Net

- Enable analysis of CCS processes by Petri net algorithms

Non-Determinism and Unboundedness

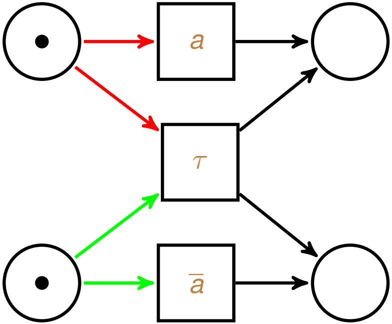
Observations:

- Also without interleaving, parallel composition $\parallel$ can still induce non-determinism (due to conflicts), e.g., $a.\mathrm{nil} \parallel \overline{a}.\mathrm{nil}$:

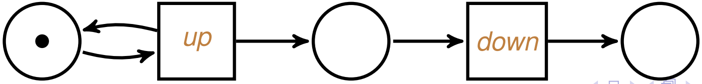


Observations:

- Also without interleaving, parallel composition $\parallel$ can still induce non-determinism (due to conflicts), e.g., $a.\mathrm{nil} \parallel \overline{a}.\mathrm{nil}$:



- Recursive process calls can entail unboundedness, e.g., $C = up.(C \parallel down.\mathrm{nil})$ (counter; cf. Example 2.6):



The Approach

Goal: Map (a restricted class of) CCS process definitions to finite Petri nets.

The Approach

Goal: Map (a restricted class of) CCS process definitions to finite Petri nets.

Requirements:

(1) Cover as much of CCS as possible (problem: CCS is Turing complete and finite Petri nets are not).

(2) To support inductive verification proofs, nets should be constructed inductively by means of composition operators (as in CCS).

The Approach

Goal: Map (a restricted class of) CCS process definitions to finite Petri nets.

Requirements:

(1) Cover as much of CCS as possible (problem: CCS is Turing complete and finite Petri nets are not).

(2) To support inductive verification proofs, nets should be constructed inductively by means of composition operators (as in CCS).

Method:

(1) Consider only guarded processes and omit restriction and relabelling operators.

(2) Specify translation $[\![.]\!] : \mathit{CCS} \to \mathit{Petri}$ in a compositional way, e.g.,

$$[\![Q_1 + Q_2]\!] := \underbrace{[\![Q_1]\!] \oplus [\![Q_2]\!]}_{\text{operation on Petri nets}}$$

Definition 17.1 (Syntax of Guarded CCS; cf. Definition 2.1)

- Let $A, \overline{A} := \{\overline{a} \mid a \in A\}$ and $Act := A \cup \overline{A} \cup \{\tau\}$ be the sets of (action) names, co-names, and actions, and let Pid be a set of process identifiers.
- The set $Prc^\dagger$ of guarded process expressions is defined by the following syntax:

$$Q ::= \sum_{i=1}^{n} \alpha_i.Q_i \quad \mid \quad Q_1 \parallel Q_2 \quad \mid \quad C$$

  where $n \in \mathbb{N}$, $\alpha_i \in Act$ and $C \in Pid$.
- Also, every process call C must be guarded, i.e., occur in an expression of the form $\alpha.Q$.
- A guarded process definition is an equation system of the form

$$(C_i = Q_i \mid 1 \leq i \leq k)$$

  where $k \geq 1$, $C_i \in Pid$ (pairwise distinct), and $Q_i \in Prc^\dagger$ (with identifiers from $\{C_1, \ldots, C_k\}$).

Definition 17.1 (Syntax of Guarded CCS; cf. Definition 2.1)

- Let A, $\overline{A} := \{\overline{a} \mid a \in A\}$ and $Act := A \cup \overline{A} \cup \{\tau\}$ be the sets of (action) names, co-names, and actions, and let Pid be a set of process identifiers.
- The set $Prc^\dagger$ of guarded process expressions is defined by the following syntax:

$$Q ::= \sum_{i=1}^{n} \alpha_i.Q_i \quad \mid \quad Q_1 \parallel Q_2 \quad \mid \quad C$$

  where $n \in \mathbb{N}$, $\alpha_i \in Act$ and $C \in Pid$.
- Also, every process call C must be guarded, i.e., occur in an expression of the form $\alpha.Q$.
- A guarded process definition is an equation system of the form

$$(C_i = Q_i \mid 1 \leq i \leq k)$$

  where $k \geq 1$, $C_i \in Pid$ (pairwise distinct), and $Q_i \in Prc^\dagger$ (with identifiers from $\{C_1, \ldots, C_k\}$).

Notes:

- Restriction and relabelling are not used any longer.
- The guardedness condition excludes, e.g., definitions of the form $C = C$.
- Since $Prc^\dagger \subseteq Prc$ (Definition 2.1), Definition 2.4 of the semantics still applies.

Petri Nets Revisited

In order to connect transitions to actions and to support the handling of process identifiers, we introduce labels for transitions and places.

Definition 17.2 (Labelled Petri net; cf. Definition 14.2)

A labelled Petri net N is a quintuple (P, T, F, l, m) where:

- P is a finite set of places,
- T is a finite set of transitions with $P \cap T = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ are the arcs,
- $l : T \to Act$ is the transition labelling, and
- $m : P \dashrightarrow Pid$ is the (partial) place labelling.

Adding an initial marking $M_0 : P \to \mathbb{N}$ yields a labelled elementary system net (P, T, F, l, m, M_0).

Definition 17.3 (Marking graph; cf. Definition 14.18)

Let $N = (P, T, F, l, m, M_0)$ be a labelled elementary system net and $M : P \to \mathbb{N}$.

- Marking M enables a transition $t \in T$ if $M(p) \geq 1$ for each place $p \in {}^\bullet t$.
- Its firing leads to marking M', denoted by the step relation $M \xrightarrow{l(t)} M'$ and defined for each place $p \in P$ by

$$M'(p) := M(p) - F(p, t) + F(t, p)$$

  where we represent F by its characteristic function.
- The marking graph of N has as nodes the reachable markings of N and as edges the corresponding steps of N.[a]

[a] Due to transition labels, marking graphs are generally no longer deterministic LTSs.

Outline of Lecture 17

Guarded Choice I

(**Reminder:** $Q ::= \sum_{i=1}^{n} \alpha_i.Q_i \mid Q_1 \parallel Q_2 \mid C \in Prc^\dagger$)

Approach: Implement non-determinism by conflicting transitions (one for each choice) and branch to outset of respective subnet.[1]

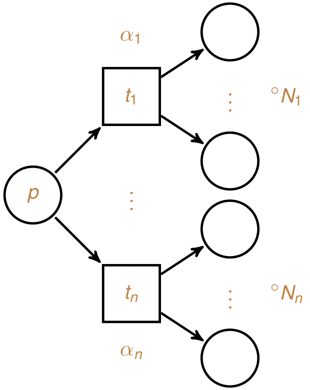
Translating guarded choice

Let $Q = \sum_{i=1}^{n} \alpha_i.Q_i \in Prc^\dagger$ and $[\![Q_i]\!] = N_i = (P_i, T_i, F_i, l_i, m_i)$ for $1 \leq i \leq n$. Then

$$[\![Q]\!] := (P \,\dot\cup\, P', T \,\dot\cup\, T', F \,\dot\cup\, F', l \,\dot\cup\, l', m)$$

where

$$\begin{aligned}
P &:= \bigcup_{i=1}^{n} P_i & P' &:= \{p\} \\
T &:= \bigcup_{i=1}^{n} T_i & T' &:= \{t_1, \ldots, t_n\} \\
F &:= \bigcup_{i=1}^{n} F_i & F' &:= \{(p, t_i) \mid 1 \leq i \leq n\} \,\dot\cup\, \bigcup_{i=1}^{n} \{t_i\} \times {}^\circ N_i \\
l &:= \bigcup_{i=1}^{n} l_i & l' &:= [t_i \mapsto \alpha_i \mid 1 \leq i \leq n] \\
m &:= \bigcup_{i=1}^{n} m_i & &
\end{aligned}$$



[1] Reminder: $^\circ N = \{p \in P \mid {}^\bullet p = \emptyset\}$.

Guarded Choice II

Example 17.4

(1) $Q = \mathsf{nil} \quad (= \sum_{\emptyset} \alpha_i.Q_i)$:

Guarded Choice II

Example 17.4

(1) $Q = \text{nil} \quad (= \sum_{\emptyset} \alpha_i.Q_i)$:

(2) $Q = a.\text{nil}$:

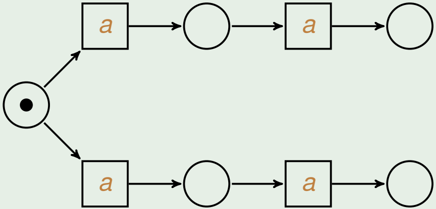
Guarded Choice II

Example 17.4

(1) $Q = \text{nil} \quad (= \sum_{\emptyset} \alpha_i.Q_i)$:

(2) $Q = a.\text{nil}$:

(3) $Q = a.b.\text{nil} + b.a.\text{nil}$:



Parallel Composition I

Approach:

- Model concurrency by disjoint union of subnets, enlarged by τ-transitions for all possible synchronisation operations.
- The latter are enabled by transitions in both subnets with complementary action labels.

Translating parallel composition

Let $Q = Q_1 \parallel Q_2 \in Prc^\dagger$ and $[\![Q_i]\!] = N_i = (P_i, T_i, F_i, l_i, m_i)$ for $i \in \{1, 2\}$ (all P_i and T_i disjoint). Then

$$[\![Q]\!] := (P_1 \dot\cup P_2, T_1 \dot\cup T_2 \dot\cup T_\tau, F_1 \dot\cup F_2 \dot\cup F_\tau, l_1 \dot\cup l_2 \dot\cup l_\tau, m_1 \dot\cup m_2)$$

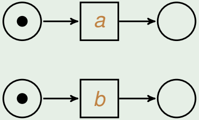
where

$$\begin{aligned} T_\tau &:= \{(t_1, t_2) \mid t_1 \in T_1, l_1(t_1) \in A \cup \overline{A}, \\ &\qquad\qquad t_2 \in T_2, l_2(t_2) = \overline{l_1(t_1)}\} && \text{(new } \tau\text{-transitions)} \\ F_\tau &:= \{(p_1, (t_1, t_2)), (p_2, (t_1, t_2)), ((t_1, t_2), p_1'), ((t_1, t_2), p_2') \mid && \text{(corresponding arcs)} \\ &\qquad (t_1, t_2) \in T_\tau, p_1 \in {}^\bullet t_1, p_2 \in {}^\bullet t_2, p_1' \in t_1^\bullet, p_2' \in t_2^\bullet\} \\ l_\tau &:= [(t_1, t_2) \mapsto \tau \mid (t_1, t_2) \in T_\tau] && \text{(}\tau\text{-labels for transitions)} \end{aligned}$$

Parallel Composition II

Example 17.5

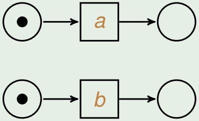
(1) $Q = a.\mathrm{nil} \parallel b.\mathrm{nil}$:



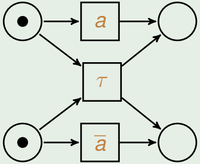
Parallel Composition II

Example 17.5

(1) $Q = a.\mathrm{nil} \parallel b.\mathrm{nil}$:



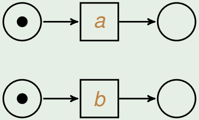
(2) $Q = a.\mathrm{nil} \parallel \overline{a}.\mathrm{nil}$:



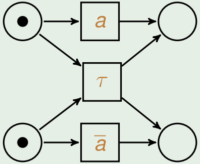
Parallel Composition II

Example 17.5

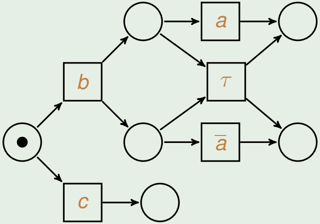
(1) $Q = a.\mathrm{nil} \parallel b.\mathrm{nil}$:



(2) $Q = a.\mathrm{nil} \parallel \overline{a}.\mathrm{nil}$:



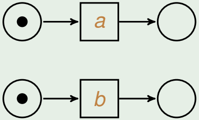
(3) $Q = b.(a.\mathrm{nil} \parallel \overline{a}.\mathrm{nil}) + c.\mathrm{nil}$:



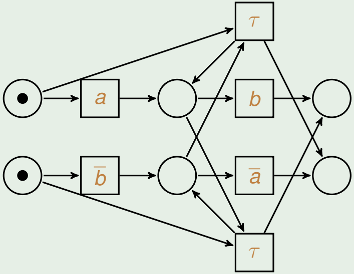
Parallel Composition II

Example 17.5

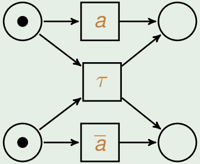
(1) $Q = a.\text{nil} \parallel b.\text{nil}$:



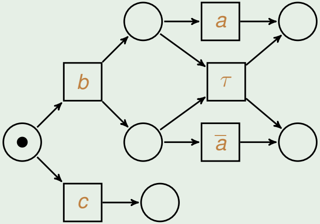
(4) $Q = a.b.\text{nil} \parallel \overline{b}.\overline{a}.\text{nil}$:



(2) $Q = a.\text{nil} \parallel \overline{a}.\text{nil}$:



(3) $Q = b.(a.\text{nil} \parallel \overline{a}.\text{nil}) + c.\text{nil}$:



Recursive Process Calls I

Approach: Introduce labelled places for process calls (using mapping m), replace each of them by arcs to all initial places of the corresponding process expression (convert tail recursion to loop).

Translating recursive process calls

- For a process call $C \in Prc^\dagger$ ($C \in Pid$), we let
$$[\![C]\!] := (\{p\}, \emptyset, \emptyset, \emptyset, [p \mapsto C]).$$
- For a guarded process definition $D = (C_i = Q_i \mid 1 \leq i \leq k)$ ($C_i \in Pid$, $Q_i \in Prc^\dagger$) with $[\![Q_i]\!] = N_i = (P_i, T_i, F_i, I_i, m_i)$ for $1 \leq i \leq k$, we let
$$[\![Q]\!] := (P \setminus P', T, F \setminus (T \times P') \,\dot{\cup}\, F', I, \emptyset)$$
where

$P := \bigcup_{i=1}^{n} P_i$	(all places)
$P' := \bigcup_{i=1}^{n} P'_i$	(process calls)
$P'_i := m^{-1}(\{C_i\}) \quad (= \{p \in P \mid m(p) = C_i\})$	(calls of C_i)
$T := \bigcup_{i=1}^{n} T_i$	(all transitions)
$F := \bigcup_{i=1}^{n} F_i$	(all flows)
$F' := \{(t,p) \in T \times P \mid \exists i \in \{1, \ldots k\} : t^\bullet \cap P'_i \neq \emptyset, p \in {}^\circ N_i\}$	(arcs for process calls)

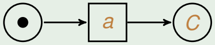
Recursive Process Calls II

Example 17.6

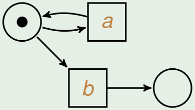
(1) Call $a.C$:

Example 17.6

(1) Call $a.C$:

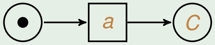


(2) Definition $C = a.C + b.\mathrm{nil}$:

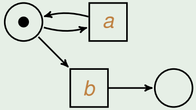


Example 17.6

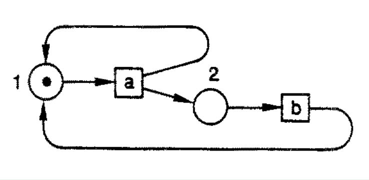
(1) Call $a.C$:



(2) Definition $C = a.C + b.\mathrm{nil}$:



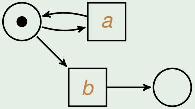
(3) Definition $C = a.(C \parallel b.C)$:



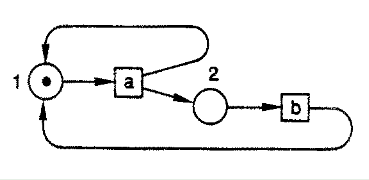
Example 17.6

(1) Call $a.C$:

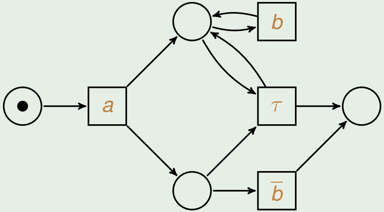
(2) Definition $C = a.C + b.\mathrm{nil}$:



(3) Definition $C = a.(C \parallel b.C)$:



(4) Definition $C = a.(D \parallel \overline{b}.\mathrm{nil}),\ D = b.D$:



Outline of Lecture 17

Theorem 17.7

Let $Q \in Prc^{\dagger}$ be a guarded process expression, and let

$$[\![Q]\!] = N = (P, T, F, l, m, M_0)$$

be its labelled elementary system net with initial marking $M_0 = {}^{\circ}N$.
Then $LTS(Q)$ and the marking graph of N are strongly bisimilar.

Theorem 17.7

Let $Q \in Prc^\dagger$ be a guarded process expression, and let

$$[\![Q]\!] = N = (P, T, F, l, m, M_0)$$

be its labelled elementary system net with initial marking $M_0 = {}^\circ N$.
Then $LTS(Q)$ and the marking graph of N are strongly bisimilar.

Proof.

see Ursula Goltz: *On representing CCS programs by finite Petri nets*, MFCS 1988 □

Correctness of Translation I

Theorem 17.7

Let $Q \in Prc^\dagger$ *be a guarded process expression, and let*

$$[\![Q]\!] = N = (P, T, F, l, m, M_0)$$

be its labelled elementary system net with initial marking $M_0 = {}^\circ N$*.*
Then $LTS(Q)$ *and the marking graph of* N *are strongly bisimilar.*

Proof.

see Ursula Goltz: *On representing CCS programs by finite Petri nets*, MFCS 1988 □

Conjecture

N is bounded iff $LTS(Q)$ is finite.

Correctness of Translation II

Example 17.8 (CCS process with finite LTS; cf. Example 17.6(4))

Process definition:

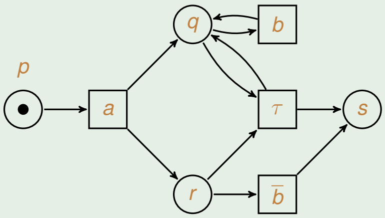
$$C = a.(D \parallel \overline{b}.\mathrm{nil}), \; D = b.D$$

Example 17.8 (CCS process with finite LTS; cf. Example 17.6(4))

Process definition:

$$C = a.(D \parallel \overline{b}.\mathrm{nil}), \; D = b.D$$

Net (one-bounded):

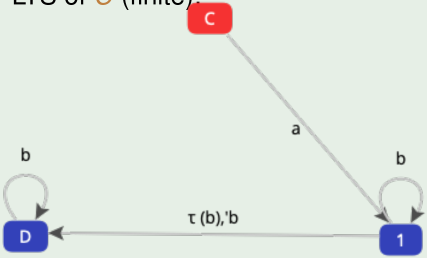


Example 17.8 (CCS process with finite LTS; cf. Example 17.6(4))

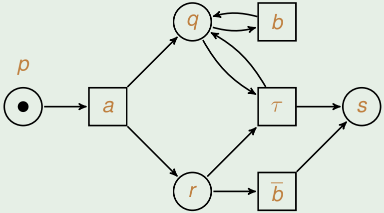
Process definition:

$$C = a.(D \parallel \overline{b}.\mathrm{nil}),\ \ D = b.D$$

LTS of C (finite):



Net (one-bounded):



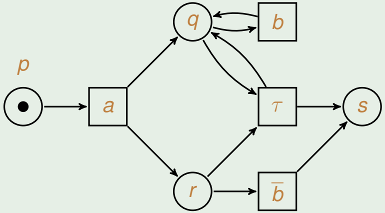
Example 17.8 (CCS process with finite LTS; cf. Example 17.6(4))

Process definition:

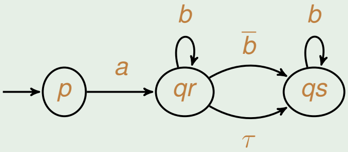
$$C = a.(D \parallel \overline{b}.\mathrm{nil}), \;\; D = b.D$$

LTS of C (finite):

Net (one-bounded):



Marking graph:



Correctness of Translation III

Example 17.9 (CCS process with infinite LTS; cf. Example 2.6)

Definition of counter process :

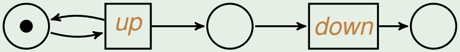
$$C = up.(C \parallel down.\mathrm{nil})$$

Example 17.9 (CCS process with infinite LTS; cf. Example 2.6)

Definition of counter process :

$$C = up.(C \parallel down.\mathrm{nil})$$

Net:



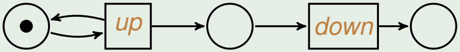
Correctness of Translation III

Example 17.9 (CCS process with infinite LTS; cf. Example 2.6)

Definition of counter process :

$$C = up.(C \parallel down.\mathrm{nil})$$

Net:



Reachable states:

$$C \xrightarrow{w} C \parallel (down.\mathrm{nil})^{u-d} \parallel \mathrm{nil}^{d}$$

where $w \in \{up, down\}^*$
with $|w|_{up} = u$ and $|w|_{down} = d$
(and $|v|_{down} \leq |v|_{up}$
for each prefix v of w).

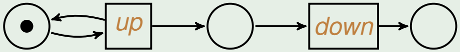
Correctness of Translation III

Example 17.9 (CCS process with infinite LTS; cf. Example 2.6)

Definition of counter process :

$$C = up.(C \parallel down.\mathrm{nil})$$

Net:



Reachable states:

$$C \xrightarrow{w} C \parallel (down.\mathrm{nil})^{u-d} \parallel \mathrm{nil}^{d}$$

where $w \in \{up, down\}^*$
with $|w|_{up} = u$ and $|w|_{down} = d$
(and $|v|_{down} \leq |v|_{up}$
for each prefix v of w).

Corresponding configurations:

Outline of Lecture 17

Summary

- Guarded CCS processes without restriction and relabelling can be mapped to finite Petri nets.

Summary

- Guarded CCS processes without restriction and relabelling can be mapped to finite Petri nets.
- Interleaving/synchronisation is handled via conflicting transitions, and recursion via looping.

Summary

- Guarded CCS processes without restriction and relabelling can be mapped to finite Petri nets.
- Interleaving/synchronisation is handled via conflicting transitions, and recursion via looping.
- The resulting marking graph is strongly bisimilar to the (LTS of) the CCS process.

Summary

- Guarded CCS processes without restriction and relabelling can be mapped to finite Petri nets.
- Interleaving/synchronisation is handled via conflicting transitions, and recursion via looping.
- The resulting marking graph is strongly bisimilar to the (LTS of) the CCS process.
- Conjecture: The net is bounded iff the LTS is finite.