Teoria współbieżności

Piotr Hofman

Theoretical aspects of concurrency

How to specify systems

- Process Algebra,
- Petri nets and their extensions.

Process algebra

A gentle introduction.

What do we need to model a system with many independent components?

1. We need to have a possibility to model individual components (processes).
   (LTS)
2. We need to have a possibility to execute a two processes in parallel.
   Tokens.
3. We need to have some notion of communication between processes/synchronisation.
   1. channels,
   2. rendezvous,
   3. sheared resources,
   4. broadcasting.
4. A possibility of creation of new processes.
5. A private communication. (static / dynamic)

CCS

The calculus of communicating systems.

Concurrent process expressions, syntax

Let Σ be a set of names. Suppose that our systems communicates using signals form a following set $\Sigma \times \{\_^1, \_^{-1}\}$ where for $\alpha \in \Sigma$ $\alpha^1 = \alpha$ means send α and α^{-1} means receive α. (Convection: $(\alpha^{-1})^{-1} = \alpha$).

Definition CCS

The set $\mathbb{P}$ of concurrent process expressions is defined by a following syntax

$$P := \left| \begin{array}{l} 0, A, \\ \sum_{i \in I} \alpha_i . P_i \\ P_1 | P_2 \\ (\nu \alpha) P \end{array} \right.$$

Where:

- $\alpha_i \in \Sigma \times \{\_^1, \_^{-1}\} \cup \{\tau\}$.
- every process identifier A has its own unique expression that defines it $A := P_A$, where P_A is a process expression.
- ν (new) and $\alpha \in \Sigma$

Formal semantic

Non-deterministic choice: $\alpha.P + \sum_{i \in I} \alpha_i.P_i \xrightarrow{\alpha} P$

REL: $\frac{P_A \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}$ if $A := P_A$,

$$R - par : \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'} \qquad L - par_t : \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

RES: $\frac{P \xrightarrow{\alpha} P'}{(\nu\beta)P \xrightarrow{\alpha} (\nu\beta)P'}$ if $\alpha \notin \{\beta, \beta^{-1}\}$

Internal: $\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\alpha^{-1}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$

Example

$$A = \alpha.A' \qquad A' = \beta^{-1}.A$$

$$B = \alpha^{-1}.B' \qquad B' = \delta^{-1}.B$$

Example

$$A = \alpha.A' \qquad A' = \beta^{-1}.A$$
$$B = \alpha^{-1}.B' \qquad B' = \delta^{-1}.B$$

1. What are the possible actions of $(A|B)$?

Example

$$A = \alpha.A' \qquad A' = \beta^{-1}.A$$
$$B = \alpha^{-1}.B' \qquad B' = \delta^{-1}.B$$

1. What are the possible actions of $(A|B)$?
2. What are the possible actions of $(\nu\{\alpha\})(A|B)$?

Example

$$A = \alpha.A' \qquad A' = \beta^{-1}.A$$
$$B = \alpha^{-1}.B' \qquad B' = \delta^{-1}.B$$

1. What are the possible actions of $(A|B)$?
2. What are the possible actions of $(\nu\{\alpha\})(A|B)$?
3. Draw LTS induced by $A|B$.

Example 2

$$C = 0 + (\alpha.(C|B)) + (\beta.(C|A))$$
$$A = \alpha.0$$
$$B = \beta.0$$

1. What are the possible traces of C ?
2. Draw an initial fragment of the LTS induced by C.

Example 3 - a counter

How we can model a counter using process algebra?

$$\begin{aligned} Count_0 &\stackrel{\text{def}}{=} inc.Count_1 + zero.Count_0 \\ Count_i &\stackrel{\text{def}}{=} inc.Count_{i+1} + dec.Count_{i-1} \end{aligned} \tag{1}$$

Example 3 - a counter

How we can model a counter using process algebra?

$$\begin{aligned} Count_0 &\stackrel{\text{def}}{=} inc.Count_1 + zero.Count_0 \\ Count_i &\stackrel{\text{def}}{=} inc.Count_{i+1} + dec.Count_{i-1} \end{aligned} \tag{1}$$

$$\begin{aligned} Z &:= zero.Z + inc.((\nu a)(C_1|a.Z)) \\ C_1 &:= dec.a^{-1}.0 + inc.((\nu b)(C_2|b.C_1)) \\ C_2 &:= dec.b^{-1}.0 + inc.((\nu a)(C_1|a.C_2)) \end{aligned} \tag{2}$$

Proof of the equivalence

1. Equivalence - weak bisimulation
2. How to prove? Define a winning region for Duplicator and prove that it is closed for one round of the weak bisimulation game.
3. Let us start from $count_i \approx (Z \xrightarrow{inc^i})$
   - $count_0 \approx Z$,
   - $count_1 \approx (\nu a)(a.Z|C_1)$,
   - $count_2 \approx (\nu a)(a.Z|((\nu b)(b.C_1|C_2))) \approx count_2$
   - $count_3 \approx (\nu a)(a.Z|((\nu b)(b.C_1|((\nu a)(a.C_2|C_1))))) \approx count_3$
   - ...
   - Terms.
4. Problem - what is happening if *dec* is used?

When processes are equivalent?

So what are the properties of the equivalence that we are looking for?

When processes are equivalent?

So what are the properties of the equivalence that we are looking for?

Syntactic equivalence $\equiv$

1. $P \equiv Q$ should imply $P \approx Q$.

When processes are equivalent?

So what are the properties of the equivalence that we are looking for?

Syntactic equivalence $\equiv$

1. $P \equiv Q$ should imply $P \approx Q$.
2. $\equiv$ is a relation on the syntactical level.

When processes are equivalent?

Let us denote process equivalence by $\equiv$.

1. $P|Q \equiv Q|P$

When processes are equivalent?

Let us denote process equivalence by $\equiv$.

1. $P|Q \equiv Q|P$
2. $P|0 \equiv P$

When processes are equivalent?

Let us denote process equivalence by $\equiv$.

1. $P|Q \equiv Q|P$
2. $P|0 \equiv P$
3. $P + P \equiv P$

When processes are equivalent?

Let us denote process equivalence by $\equiv$.

1. $P|Q \equiv Q|P$
2. $P|0 \equiv P$
3. $P + P \equiv P$
4. $(\nu a)a.P \equiv 0$.

When processes are equivalent?

Let us denote process equivalence by $\equiv$.

1. $P|Q \equiv Q|P$
2. $P|0 \equiv P$
3. $P + P \equiv P$
4. $(\nu a)a.P \equiv 0$.
5. if $A := P$ then $A \equiv P$

When processes are equivalent?

Let us denote process equivalence by $\equiv$.

1. $P|Q \equiv Q|P$
2. $P|0 \equiv P$
3. $P + P \equiv P$
4. $(\nu a)a.P \equiv 0$.
5. if $A := P$ then $A \equiv P$

Problem

Let $P = A|a.0$ then $A = P$ has many solutions.

- $A = b.0|a.0|a.0|a.0|\ldots$
- $A = c.0|a.0|a.0|a.0|\ldots$

This two processes should not be equivalent.

Unique solutions of recurrent definitions.

Definition (a guarded process identifier)

An occurrence of a process identifier A in a process q is strongly guarded in q if such occurrence of A occurs within a prefixed sub-process $\alpha.q'$ of q.

Unfold procedure:

- take each not strongly guarded process identifier and substitute it with the defining expression.
- repeat this process until you have some not strongly guarded process identifiers.

Constant A, defined as $A = p$, is guarded if the unfolding process stops.

Lemma

If all process identifiers are guarded then the equation has unique solution up to bisimulation.

Unique solutions of recurrent definitions.

Definition (CCS - guarded)

We restrict our-self to the fragment of CCS where every process identifier is guarded.

Observe that all examples considered until now were guarded.

When processes are equivalent?

Let us denote process equivalence by $\equiv$.

1. $P|Q \equiv Q|P$
2. $P|0 \equiv P$
3. $P + P \equiv P$
4. $P + 0 \equiv P$.

What is the fundamental property of a good notion of equivalence? (compositionality)

$$\text{if } P \equiv Q \text{ then for example } R|P \equiv R|Q \text{ and } P + R \equiv Q + R.$$

Let us try to generalise this.

Congruence

Definition

Context is a process with a hole represented by []. Formally it is given by a grammar:

$$\mathbf{C} := [] \ , \ \alpha.C \ , \ C + \sum_{i \in I} P_i \ , \ (\nu\alpha)C \ , \ C|P \ , \ P|C$$

where I is a finite family and P_i are processes.

Congruence

Definition

Context is a process with a hole represented by []. Formally it is given by a grammar:

$$\mathbf{C} := [] \ , \ \alpha.C \ , \ C + \sum_{i \in I} P_i \ , \ (\nu\alpha)C \ , \ C|P \ , \ P|C$$

where I is a finite family and P_i are processes.

Definition

A congruence relation $\cong$ (or simply congruence) is an equivalence relation on an algebraic structure that is compatible with the structure i.e. If $P \cong Q$ then for any context C holds $C[P] \cong C[Q]$.

Congruence

Definition

Context is a process with a hole represented by []. Formally it is given by a grammar:

$$\mathbf{C} := [] \ , \ \alpha.C \ , \ C + \sum_{i \in I} P_i \ , \ (\nu\alpha)C \ , \ C|P \ , \ P|C$$

where I is a finite family and P_i are processes.

Definition

A congruence relation $\cong$ (or simply congruence) is an equivalence relation on an algebraic structure that is compatible with the structure i.e. If $P \cong Q$ then for any context C holds $C[P] \cong C[Q]$.

Which relations are congruencies?

Congruence

Definition

Context is a process with a hole represented by []. Formally it is given by a grammar:

$$\mathbf{C} := [] \ , \ \alpha.C \ , \ C + \sum_{i \in I} P_i \ , \ (\nu\alpha)C \ , \ C|P \ , \ P|C$$

where I is a finite family and P_i are processes.

Definition

A congruence relation $\cong$ (or simply congruence) is an equivalence relation on an algebraic structure that is compatible with the structure i.e. If $P \cong Q$ then for any context C holds $C[P] \cong C[Q]$.

Which relations are congruencies?

- Trace equivalence.
- Bisimilarity.
- Weak bisimilarity.

Trace equivalence

We analyse rule by rule. Suppose that $P =_{Tr} Q$.

Trace equivalence

We analyse rule by rule. Suppose that $P =_{Tr} Q$.

- $P + R =_{Tr} Q + R$ and $R + P =_{Tr} R + Q$.

Trace equivalence

We analyse rule by rule. Suppose that $P =_{Tr} Q$.

- $P + R =_{Tr} Q + R$ and $R + P =_{Tr} R + Q$.
- $P|R =_{Tr} Q|R$ and $R|P =_{Tr} R|Q$.

Trace equivalence

We analyse rule by rule. Suppose that $P =_{Tr} Q$.

- $P + R =_{Tr} Q + R$ and $R + P =_{Tr} R + Q$.
- $P|R =_{Tr} Q|R$ and $R|P =_{Tr} R|Q$.
- $a.P =_{Tr} a.Q$

Trace equivalence

We analyse rule by rule. Suppose that $P =_{Tr} Q$.

- $P + R =_{Tr} Q + R$ and $R + P =_{Tr} R + Q$.
- $P|R =_{Tr} Q|R$ and $R|P =_{Tr} R|Q$.
- $a.P =_{Tr} a.Q$
- $\nu aP =_{Tr} \nu aQ$.

Completed trace equivalence

Definition

For a given system a set of completed traces is a set of words via which we may reach a state which is a deadlock in the induced *LTS*.

Completed trace equivalence

Definition

For a given system a set of completed traces is a set of words via which we may reach a state which is a deadlock in the induced *LTS*.

It was introduced, because sometimes we are interested in deadlocks.

Completed trace equivalence

Definition

For a given system a set of completed traces is a set of words via which we may reach a state which is a deadlock in the induced *LTS*.

It was introduced, because sometimes we are interested in deadlocks.
Question: Does the completed traces equivalence is a congruence?

Completed trace equivalence

Definition

For a given system a set of completed traces is a set of words via which we may reach a state which is a deadlock in the induced *LTS*.

It was introduced, because sometimes we are interested in deadlocks.
Question: Does the completed traces equivalence is a congruence?
Answer: No. we apply νc to two equivalent processes

1. $a.(b.0 + c.0)$
2. $a.b.0 + a.c.0$

Bisimilarity

Theorem

The bisimilarity relation is a congruence.

Bisimilarity

Theorem

The bisimilarity relation is a congruence.

Proof.

- Let $\sim_C$ is a set of pairs $(C[P/X], C[Q/X])$ where $C[X]$ is some context and $P \sim Q$ is a pair of bisimilar processes.
- We have to prove that $\sim_C$ is a bisimulation.

$\square$

Bisimilarity

Theorem

The bisimilarity relation is a congruence.

Proof.

- Let $\sim_C$ is a set of pairs $(C[P/X], C[Q/X])$ where $C[X]$ is some context and $P \sim Q$ is a pair of bisimilar processes.
- We have to prove that $\sim_C$ is a bisimulation.
- Take a pair $P \sim Q$ and any context $C[X]$, we have to verify that for every α if $C[P/X] \xrightarrow{\alpha} P'$ then there is Q' such that $C[Q/X] \xrightarrow{\alpha} Q'$ and $P' \sim Q'$.

$\square$

Bisimilarity

Theorem

The bisimilarity relation is a congruence.

Proof.

- Let $\sim_C$ is a set of pairs $(C[P/X], C[Q/X])$ where $C[X]$ is some context and $P \sim Q$ is a pair of bisimilar processes.
- We have to prove that $\sim_C$ is a bisimulation.
- Take a pair $P \sim Q$ and any context $C[X]$, we have to verify that for every α if $C[P/X] \xrightarrow{\alpha} P'$ then there is Q' such that $C[Q/X] \xrightarrow{\alpha} Q'$ and $P' \sim Q'$.
- We prove by induction on the size of the context.

$\square$

Weak Bisimilarity

Fact

Weak bisimilarity is not a congruence.

$P \approx \tau.P$ but $a.0 + P$ not necessarily weakly bisimilar with $a.0 + \tau.P$.

Weak Bisimilarity

Fact

Weak bisimilarity is not a congruence.

$P \approx \tau.P$ but $a.0 + P$ not necessarily weakly bisimilar with $a.0 + \tau.P$.

Lemma

Weak bisimilarity is a congruence with respect to the following operations. Let $P \approx Q$ and $R \approx S$ then

1. $P|R \approx Q|S$
2. $\nu aP \approx \nu aQ$
3. $\alpha.P + \sum_i R_i \approx \alpha.Q + \sum_i R_i$ where $\alpha \neq \tau$

Weak Bisimilarity

Fact

Weak bisimilarity is not a congruence.

$P \approx \tau.P$ but $a.0 + P$ not necessarily weakly bisimilar with $a.0 + \tau.P$.

Lemma

Weak bisimilarity is a congruence with respect to the following operations. Let $P \approx Q$ and $R \approx S$ then

1. $P|R \approx Q|S$
2. $\nu aP \approx \nu aQ$
3. $\alpha.P + \sum_i R_i \approx \alpha.Q + \sum_i R_i$ where $\alpha \neq \tau$

Weak bisimilarity is good as it abstract form internal τ moves.

Weak Bisimilarity

Fact

Weak bisimilarity is not a congruence.

$P \approx \tau.P$ but $a.0 + P$ not necessarily weakly bisimilar with $a.0 + \tau.P$.

Lemma

Weak bisimilarity is a congruence with respect to the following operations. Let $P \approx Q$ and $R \approx S$ then

1. $P|R \approx Q|S$
2. $\nu aP \approx \nu aQ$
3. $\alpha.P + \sum_i R_i \approx \alpha.Q + \sum_i R_i$ where $\alpha \neq \tau$

Weak bisimilarity is good as it abstract form internal τ moves.
New concept, let us try to identify a biggest congruence included in weak bismilarity.

Biggest congruence included in the weak bisimilarity relation.

Lemma

Equivalence closure of a union of a family of congruence relations is a congruence.

Lemma

There is a biggest congruence included in the weak bisimilarity relation.

Rooted Weak Bisimilarity

By $\overset{a+}{\Longrightarrow}$ is a sequence of transitions like $\overset{a+}{\Longrightarrow}$ but a nonempty one.

Definition

The rooted weak bisimulation is a relation R on the set of configurations such that if $(s, s') \in R$ then

1. For any label $a \in \Sigma \cup \Sigma^{-1} \cup \{\tau\}$ and every step $s \xrightarrow{a} t$ there is an answer $s' \overset{a+}{\Longrightarrow} t'$ such that $t \approx t'$.
2. For any label $a \in \Sigma \cup \Sigma^{-1} \cup \{\tau\}$ and every step $s' \xrightarrow{a} t'$ there is an answer $s \overset{a+}{\Longrightarrow} t$ such that $t \approx t'$

The difference, comparing with the weak bisimulation relation, is only in the first move!

Rooted Weak Bisimilarity

By $\overset{a+}{\Longrightarrow}$ is a sequence of transitions like $\overset{a+}{\Longrightarrow}$ but a nonempty one.

Definition

The rooted weak bisimulation is a relation R on the set of configurations such that if $(s, s') \in R$ then

1. For any label $a \in \Sigma \cup \Sigma^{-1} \cup \{\tau\}$ and every step $s \xrightarrow{a} t$ there is an answer $s' \overset{a+}{\Longrightarrow} t'$ such that $t \approx t'$.
2. For any label $a \in \Sigma \cup \Sigma^{-1} \cup \{\tau\}$ and every step $s' \xrightarrow{a} t'$ there is an answer $s \overset{a+}{\Longrightarrow} t$ such that $t \approx t'$

The difference, comparing with the weak bisimulation relation, is only in the first move!

Definition

Rooted Weak bisimilarity $\approx_R$ is the biggest rooted weak bisimulation.

Rooted Weak Bisimilarity

By $\overset{a+}{\Longrightarrow}$ is a sequence of transitions like $\overset{a+}{\Longrightarrow}$ but a nonempty one.

Definition

The rooted weak bisimulation is a relation R on the set of configurations such that if $(s, s') \in R$ then

1. For any label $a \in \Sigma \cup \Sigma^{-1} \cup \{\tau\}$ and every step $s \xrightarrow{a} t$ there is an answer $s' \overset{a+}{\Longrightarrow} t'$ such that $t \approx t'$.
2. For any label $a \in \Sigma \cup \Sigma^{-1} \cup \{\tau\}$ and every step $s' \xrightarrow{a} t'$ there is an answer $s \overset{a+}{\Longrightarrow} t$ such that $t \approx t'$

The difference, comparing with the weak bisimulation relation, is only in the first move!

Definition

Rooted Weak bisimilarity $\approx_R$ is the biggest rooted weak bisimulation.

Question: design a system where Rooted Bisimilarity is in between strong and weak bisimilarity.

Rooted Weak Bisimilarity

Definition (Free names)

The free names of a process p, denoted $fn(p)$, are defined as the set $F(p, \emptyset)$, where $F(p, I)$, with I a set of process identifiers, is defined as follows:

- $F(0, I) = \emptyset$,
- $F(a.p, I) = F(a^{-1}.p, I) = F(p, I) \cup \{a\}$
- $F(\tau.p, I) = F(p, I)$
- $F(p + q, I) = F(p|q, I) = F(p, I) \cup F(q, I)$
- $F((\nu a)p, I) = F(p, I) \setminus \{a\}$
- $$F(C, I) = \begin{cases} F(q, I \cup \{C\}) & \text{if } C = q \text{ and } C \notin I \\ \emptyset & \text{if } C \in I. \end{cases} \tag{3}$$

According to this definition, $fn(p)$ is effectively computable for any process p.

Rooted Weak Bisimilarity

Lemma

Rooted Weak bisimilarity is a congruence for CCS.

Rooted Weak Bisimilarity

Lemma

Rooted Weak bisimilarity is a congruence for CCS.

Lemma

Rooted Weak Bisimilarity is a biggest congruence contained in Weak Bisimilarity for CCS.

Rooted Weak Bisimilarity

Lemma

Rooted Weak bisimilarity is a congruence for CCS.

Lemma

Rooted Weak Bisimilarity is a biggest congruence contained in Weak Bisimilarity for CCS.

Rooted Weak Bisimilarity

Lemma

Rooted Weak bisimilarity is a congruence for CCS.

Lemma

Rooted Weak Bisimilarity is a biggest congruence contained in Weak Bisimilarity for CCS.

We can prove correctness of syntactic equivalences via

1. Strong Bisimilarity,
2. Rooted Weak Bisimilarity.

Syntactic equivalences

1. Monoid laws:
   - $P + Q \cong Q + P$
   - $P + (Q + R) \cong (P + Q) + R$
   - $P + 0 \cong P$
   - $P + P \cong P$
2. τ laws:
   - $\alpha.\tau.P \cong \alpha.P$
   - $P + \tau.P \cong \tau.P$
   - $\alpha.(P + \tau.Q) + \alpha.Q \cong \alpha.(P + \tau.Q)$
3. The expansion law:
   - Let $P \cong (\nu b_1) \ldots (\nu b_k)\ (P_1 | \ldots | P_n)$ then

$$P \cong \sum_{\alpha} \alpha.(\nu b_1) \ldots (\nu b_k)\ (P_1 | \ldots | P_i' | \ldots | P_n) : P_i \xrightarrow{\alpha} P_i', \alpha \notin \{b_1, \ldots b_k^{-1}\} +$$

$$\sum_{\alpha} \tau.(\nu b_1) \ldots (\nu b_k)\ (P_1 | \ldots | P_i' | \ldots | P_j' | \ldots | P_n) : P_i \xrightarrow{\alpha} P_i',\ P_j \xrightarrow{\alpha^{-1}} P_j'$$

4. Composition laws:
   - $P|Q \cong Q|P$
   - $P|(Q|R) \cong (P|Q)|R$
   - $P|0 \cong P$

Syntactic equivalences

5. Restriction laws
   - $(\nu a)\ (\nu a)\ P \cong (\nu a)P$
   - $(\nu a)\ (\nu b)\ P \cong (\nu b)(\nu a)P$
   - $\nu a\ P \cong P$ if $\{a, a^{-1}\} \cap fn(P) = \emptyset$
   - $\nu a(P|Q) \cong \nu a\ P|\nu a\ Q$ if $fn(P) \cap fn(Q) \cap \{a, a^{-1}\} = \emptyset$

Example 3 - a counter. The second attempt.

$$\begin{aligned} Count_0 &\stackrel{\text{def}}{=} inc.Count_1 + zero.Count_0 \\ Count_i &\stackrel{\text{def}}{=} inc.Count_{i+1} + dec.Count_{i-1} \end{aligned} \tag{4}$$

$$\begin{aligned} Z &:= zero.Z + inc.((\nu a)(C_1|a.Z)) \\ C_1 &:= dec.a^{-1}.0 + inc.((\nu b)(C_2|b.C_1)) \\ C_2 &:= dec.b^{-1}.0 + inc.((\nu a)(C_1|a.C_2)) \end{aligned} \tag{5}$$

CCS reachability.

Lemma

The reachability problem for CCS is undecidable.

Proof.

1. Fact: The reachability problem for two counter machine is undecidable.
2. Using the construction form Example 3 one can simulate the two counter machine using CCS.

□

The two counter machine.

Definition

A two counter machine is a tuple (Q, q_0, F, T, L) where Q is a finite set of states q_0 is the initial state, F is the set of final states, $T \subseteq Q \times Q$ is the transition relation and L is a function from T into the set $\{r_1 = 0, r_2 = 0, r_1 ++, r_1 --, r_2 ++, r_2 --\}$.
Semantics: configurations are elements of the set $Q \times \mathbb{N} \times \mathbb{N}$. From (p, c_1, c_2) there is a step to (p', c_1', c_2') if $(p, p') \in T$ and

$$(c_i', c_{1-i}') = \begin{cases} (0, c_{1-i}) & \text{if } L((p,p')) = r_i = 0 \text{ and } c_i = 0 \\ (c_i + 1, c_{1-i}) & \text{if } L((p,p')) = r_i ++ \\ (c_i - 1, c_{1-i}) & \text{if } L((p,p')) = r_i -- \end{cases}$$

Two counter machine reachability.

Lemma

The reachability problem for two counter machine is undecidable.

Proof - Homework, may appear on the exam!

1. The halting problem for Turing machines is undecidable.
2. The reachability problem for an automaton with two pushdowns is undecidable.
3. One can simulate operations on the pushdown using counters (binary encoding, push - multiply by 2, pop - divide by 2).
4. Many counters can be reduced to two counters. We encode values of many counters on a single counter as $p_1^{c_1} p_2^{c_2} p_3^{c_3} \ldots$ where p_i are different prime numbers and c_i are counter values, the second counter is auxiliary.

$\square$

Analysis of CCS is not possible.

The question if α can be a next observable action is undecidable.

In theory analysis of CCS is not possible.

Analysis of CCS is not possible.

The question if α can be a next observable action is undecidable.

In theory analysis of CCS is not possible. But you can be lucky, maybe there is a proof that can be found and your system can be verified. You can try brute force for example.

Analysis of CCS is not possible.

The question if α can be a next observable action is undecidable.

In theory analysis of CCS is not possible. But you can be lucky, maybe there is a proof that can be found and your system can be verified. You can try brute force for example.

So what are benefits of CCS?

1. It is a simple formalism. Thus it is easier to design algorithms analysing it (even a brute force).
2. Compositionality, helps to reflect the original structure of our system. (For example LTS are not compositional).

Analysis of CCS is not possible.

The question if α can be a next observable action is undecidable.

In theory analysis of CCS is not possible. But you can be lucky, maybe there is a proof that can be found and your system can be verified. You can try brute force for example.

So what are benefits of CCS?

1. It is a simple formalism. Thus it is easier to design algorithms analysing it (even a brute force).
2. Compositionality, helps to reflect the original structure of our system. (For example LTS are not compositional).
3. We can try to describe some principles of concurrency.

Concurrency

For a word $w \in (\Sigma \cup \Sigma^{-1} \cup \{\tau\})^*$, where $w = \alpha_1 \ldots \alpha_k$ by

- $\xrightarrow{w}$ we mean a sequence of moves $\xrightarrow{\alpha_1}\xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_k}$,
- $\overset{w}{\Rightarrow}$ we mean a sequence of moves $\overset{\alpha_1}{\Rightarrow}\overset{\alpha_2}{\Rightarrow} \ldots \overset{\alpha_k}{\Rightarrow}$,

Definition (determinism)

We say that a process is deterministic if for every sequence of observable actions w we have that if $P \overset{w}{\Rightarrow} P_1$ and $P \overset{w}{\Rightarrow} Q_1$ then $P_1 \approx Q_1$.

This definition is a natural extension of determinism that we know.

Which operations preserve the determinacy?

If the processes P and P_i are deterministic then deterministic are also:

1. 0, $\alpha.P$, $(\nu\alpha)\ P$ are always,

Which operations preserve the determinacy?

If the processes P and P_i are deterministic then deterministic are also:

1. 0, $\alpha.P$, $(\nu\alpha)\ P$ are always,
2. $\sum_{i \in I} \alpha_i.P_i$ assuming that if $\alpha_i = \alpha_j$ then $P_i \approx P_j$, ,

Which operations preserve the determinacy?

If the processes P and P_i are deterministic then deterministic are also:

1. 0, $\alpha.P$, $(\nu\alpha)\ P$ are always,
2. $\sum_{i \in I} \alpha_i.P_i$ assuming that if $\alpha_i = \alpha_j$ then $P_i \approx P_j$, $(\alpha_i \neq \tau)$,

Which operations preserve the determinacy?

If the processes P and P_i are deterministic then deterministic are also:

1. 0, $\alpha.P$, $(\nu\alpha)\ P$ are always,
2. $\sum_{i \in I} \alpha_i.P_i$ assuming that if $\alpha_i = \alpha_j$ then $P_i \approx P_j$, $(\alpha_i \neq \tau)$,
3. $P_1 | P_2$ if $fn(P_1) \cap fn(P_2) = \emptyset$,

Which operations preserve the determinacy?

If the processes P and P_i are deterministic then deterministic are also:

1. 0, $\alpha.P$, $(\nu\alpha)\ P$ are always,
2. $\sum_{i \in I} \alpha_i.P_i$ assuming that if $\alpha_i = \alpha_j$ then $P_i \approx P_j$, $(\alpha_i \neq \tau)$,
3. $P_1 | P_2$ if $fn(P_1) \cap fn(P_2) = \emptyset$,
4. relabelling if it is one to one.

Problems with the determinacy?

Problem 1

The rule $P_1|P_2$ if $fn(P_1) \cap fn(P_2) = \emptyset$, is very restrictive. The parallel composition is allowed if processes can not interact, at all.

We can not weakened the restriction.
Take processes

$$P = \alpha.\gamma.0 + \beta^{-1}.\alpha.0|\beta.0 \ .$$

Processes can interact, thus $P \stackrel{\alpha}{\Rightarrow} \gamma.0|\beta.0$ and $\stackrel{\alpha}{\Rightarrow} 0|0$.

Problem 2

Determinacy does not speak about concurrent behaviours.

Concurrency

Definition (Confluence)

We say that a process P is confluent all its successors are and for every P' and P'' such that $P \xrightarrow{\alpha} P'$ and $P \overset{\beta}{\Rightarrow} P''$ hold :

- if $\alpha \neq \beta$ then there are $Q' \approx Q''$ such that $P' \overset{\beta}{\Rightarrow} Q'$ and $P'' \overset{\alpha}{\Rightarrow} Q''$.
- if $\alpha = \beta$ then there are $Q' \approx Q''$ such that $P' \overset{\tau}{\Rightarrow} Q'$ and $P'' \overset{\tau}{\Rightarrow} Q''$.

Intuition

Programs that are confluent have to be linear. It is impossible to branch without joining later.

Confluence, simpler characterisation

Definition

The *excess* of r over s, written r/s is defined recursively upon r as follows:

$$\varepsilon/s = \varepsilon$$

$$(\alpha.r)/s = \begin{cases} \alpha.(r/s) & \text{if } \alpha \notin s \\ r/(s/\alpha) & \text{if } \alpha \in s \end{cases}$$

lemma

A process P is confluent if for all $r, s \in \Sigma^*$ and $P \overset{r}{\Rightarrow} P'$ and $P \overset{s}{\Rightarrow} P''$, we have that $P' \overset{s/r}{\Longrightarrow} Q'$, and $P'' \overset{r/s}{\Longrightarrow} Q''$, and $Q' \approx Q''$.

Confluence, properties

Lemma

If P is confluent then P is determinate.

Lemma

If P_1 and P_2 are confluent then so also the following:

1. $0, \alpha.P_1$, *and* $(\nu\alpha)\ P_1$ *for* $\alpha \in \Sigma$,
2. $P[f]$, *provided that* f *is injective,*
3. $\alpha.\beta.P_1 + \beta.\alpha.P_1$.

Definition (Free names 2)

The free names2 of a process p, denoted $fn2(p)$, are defined as the set $F(p, \emptyset)$, where $F(p, I)$, with I a set of process identifiers, is defined as follows:

- $F(0, I) = \emptyset$,
- $F(a.p, I) = F(p, I) \cup \{a\}$
- $F(a^{-1}.p, I) = F(p, I) \cup \{a^{-1}\}$
- $F(\tau.p, I) = F(p, I)$
- $F(p + q, I) = F(p|q, I) = F(p, I) \cup F(q, I)$
- $F((\nu a)p, I) = F(p, I) \setminus \{a\}$
- $$F(C, I) = \begin{cases} F(q, I \cup \{C\}) & \text{if } C = q \text{ and } C \notin I \\ \emptyset & \text{if } C \in I. \end{cases} \tag{6}$$

According to this definition, $fn2(p)$ is effectively computable for any process p.

Confluence, properties

Definition

Restricted Composition For $L \subset \Sigma$ we define a restricted composition by $P_1|_L P_2 \stackrel{\text{def}}{=} \nu L\ (P_1|P_2)$
and we call it a confluent composition if additionally $fn2(P_1)^{-1} \cap fn2(P_2) \subseteq L \cup L^{-1}$, and $fn2(P_1) \cap fn2(P_2) = \emptyset$.

Lemma

Let P_1 and P_2 are confluent. Then, if $P_1|_L P_2$ is a confluent composition then it is also confluent.

Promela and Spin

- Please read the following tutorial:
  `https://spinroot.com/spin/Man/Manual.html`
- and try to do the following excerxcises:
  `http://www.cse.chalmers.se/edu/year/2016/course/TDA293/Lab1.html`
- I will ask you about Promela at the exam.