TECHNISCHE UNIVERSITÄT DRESDEN

Concurrency Theory

Lecture 6: Checking Bisimilarity in Polynomial Time

Dr. Stephan Mennicke

Institute for Theoretical Computer Science
Knowledge-Based Systems Group

May 12, 2025

International Center
for Computational Logic

Summary

Bisimilarity

1. is a process equivalence ($\simeq$ is reflexive, symmetric, and transitive),
2. is deadlock-sensitive ($\simeq \subsetneq \equiv_{\mathsf{ctr}}$),
3. preserves traces ($\simeq \subsetneq \equiv_{\mathsf{tr}}$), and
4. preserves **interaction**, meaning it is a **process congruence** (e.g., for CCS).

The *de Simone*-format brings a first metatheoretical result about bisimilarity, admitting congruences for free. Other formats exist, relaxing the coniditions of the presented format or extending/adapting it towards other equivalence notions.

Today

- Checking bisimilarity for CCS is undecidable
- Restricting CCS
  1. admits finite axiomatizations
  2. admits polynomial time procedures

CCS_{fin} and CCS^{++}_{fin}

Towards Undecidability

Minsky machines are computing devices working on so-called *counters* (aka. *registers*) that hold natural numbers. Finitely many lines of code alter the states of the counters. Minsky machines with at least two counters are Turing-complete. The following program adds the values of c_1 and c_2.

```
01: dec c2 : 02 : 03
02: inc c1 : 01
03: HALT
```

Rule of Thumb

Equivalence of models of Turing-complete formalisms is **not semi-decidable**.

- of course, bisimilarity may be different from *language equivalence*
- but undecidability persists if only deterministic systems are involved

Towards Undecidability

Definition 29 A *Minsky machine* is a pair $\mathcal{M} = (\mathbb{P}, R)$ with $\mathbb{P}$ a finite sequence $\ell_1 \ell_2 ... \ell_m$ of *program lines* and a finite set R of *counters* (i.e., $R = \{c_1, c_2, ..., c_n\}$) such that ℓ_m : HALT and each line ℓ_i $(1 \leq i < m)$ has one of the following shapes

$$i : \mathsf{inc}\ c : j \qquad\qquad i : \mathsf{dec}\ c : j : k$$

for counter $c \in R$ and $1 \leq j, k \leq m$.

A configuration of $\mathcal{M}$ is a pair $\gamma = (i, \beta)$ where $1 \leq i \leq m$ and $\beta : R \to \mathbb{N}$. For $\gamma_1 = (i, \beta_1)$ and $\gamma_2 = (j, \beta_2)$, define $\gamma_1 \triangleright \gamma_2$ by cases:

i : inc c : j $\beta_2 = \beta_1[c \mapsto \beta_1(c) + 1]$

i : dec c : k_1 : k_2 if $\beta_1(c) > 0$, $j = k_1$ and $\beta_2 = \beta_1[\beta_1(c) - 1]$; otherwise, if $\beta_1(c) = 0$, $\beta_2 = \beta_1$ and $j = k_2$.

Theorem 30 The *halting problem* for 2-counter Minsky machines is undecidable.

Exercise: Counting with CCS

Naive Approach

Consider a counter c represented by process constants $\{C_n \mid n \in \mathbb{N}\} \subseteq \mathcal{K}$. Represent operation of *incrementing* c by synchronizing on name i. Conversely, *decrementing* c by name d. Synchronization on name z may be used to check the counter value for *zero*. Define in $\mathcal{T}_{\mathcal{K}} \subseteq \mathcal{K} \times \mathsf{Act} \times \mathsf{Pr}$

$$C_0 \xrightarrow{i} C_1$$

$$C_0 \xrightarrow{\overline{z}} C_0$$

$$C_n \xrightarrow{i} C_{n+1} (n > 0)$$

$$C_n \xrightarrow{d} C_{n-1} (n > 0)$$

Here C_n represents the counter value n of c. Consequently, only C_0 may *emit* on name z.

Exercise: Counting with CCS

Names and process constants are specific to counters. Hence, if several counters $\left(c_j\right)_{j \in J}$ are in use, separate action names like i_j, d_j, z_j and process constants $C_0^j, C_1^j, C_2^j, \ldots$ for counter c_j must be reserved. There is no apparent reason why C_0 emits on name z (i.e., $C_0 \xrightarrow{\overline{z}} C_0$).

A Finite Approach

In contrast to the previous approach, we may also use a single constant to perform incrementation and respective decrementation by

$$C \xrightarrow{i} C \mid d.\mathbf{0}$$

Excluding transition $C_0 \xrightarrow{\overline{z}} C_0$ from the naive approach, $C_0 \simeq C$.

Exercise: Let us try to show $C_0 \simeq C$

In order to see that the counter implementation actually works, we have to show that the finite counter representation actually works as expected. Recall,

$$C_0 \xrightarrow{i} C_1$$

$$C_n \xrightarrow{i} C_{n+1}$$

$$C_n \xrightarrow{d} C_{n-1}$$

$$C \xrightarrow{i} C \mid d.\mathbf{0}$$

How to prove $C_0 \simeq C$?

$$\mathcal{R} = \{(C_n, C \mid \Pi_{i=0}^{n} d.\mathbf{0}) \mid n \in \mathbb{N}\}$$

Is $\mathcal{R}$ a bisimulation? **no**

$\mathcal{R}$ is a bisimulation up-to $\simeq$.

Exercise: Bisimulations up-to $\simeq$

Definition 31 A process relation $\mathcal{R}$ is a *bisimulation up-to* $\simeq$ if, whenever $p \mathcal{R} q$, for all $\mu \in \mathsf{Act}$, we have

1. $p \xrightarrow{\mu} p'$ implies a q' such that $q \xrightarrow{\mu} q'$ and $p' \simeq \mathcal{R} \simeq q'$;
2. $q \xrightarrow{\mu} q'$ implies a p' such that $p \xrightarrow{\mu} p'$ and $p' \simeq \mathcal{R} \simeq q'$.

$p' \simeq \mathcal{R} \simeq q'$ iff there are p'', q'' such that $p' \simeq p''$, $p'' \mathcal{R} q''$, and $q'' \simeq q'$.

Lemma 32 If $\mathcal{R}$ is a bisimulation up-to $\simeq$, then $\simeq \mathcal{R} \simeq$ is a bisimulation.

A Finite Approach with Zero Testing

With $C \xrightarrow{i} C \mid d.\mathbf{0}$ in place, there must not be any transition $C \xrightarrow{\overline{z}}$ since C is a component of every parallel composition succeeding C. Thus, we need to separate a constant that has $\overline{z}$ enabled while other processes must not be capable of emitting on z.

For the final encoding, we consider three constants C_0, C_1, C_2 for a single counter c. C_0 reflects the state of c where its value is 0. After incrementing, the counter holds an odd value, represented by process constant C_1. After incrementing an odd value (i.e., C_1), we reach an even value, represented by process constant C_2. Likewise, incrementing an even value (i.e., C_1), we reach another odd value (C_1).

A Finite Approach with Zero Testing

Thus, for certain contexts $\mathcal{C}, \mathcal{D}$, we get

$$C_0 \xrightarrow{\overline{z}} C_0$$
$$C_0 \xrightarrow{i} \mathcal{C}[C_1]$$
$$C_1 \xrightarrow{i} \mathcal{D}[C_2]$$
$$C_2 \xrightarrow{i} \mathcal{C}[C_1]$$

Contexts $\mathcal{C}, \mathcal{D}$ are specified next:

- because C_1 represents arbitrary odd values of c, we must not specify $C_1 \xrightarrow{d} C_0$ directly;
- instead, we make use of **alternating restrictions** on names ℓ_1, ℓ_2 hiding certain action names (like $\overline{z}$) as long as the counter has not been decremented to a respective value (e.g., 0);

A Finite Approach with Zero Testing

$$C_0 \xrightarrow{i} \boldsymbol{\nu}\ell_1\,(C_1 \mid \ell_1.C_0)$$
$$C_1 \xrightarrow{i} \boldsymbol{\nu}\ell_2\,(C_2 \mid \ell_2.C_1)$$
$$C_2 \xrightarrow{i} \boldsymbol{\nu}\ell_1\,(C_1 \mid \ell_1.C_2)$$

The *lhs* in every process context takes care of the *next* counter value, being either *odd* (C_1) or *even* (C_2). The *rhs* waits for the decrement operation to have taken place to *unguard* the counter's original value. Consequently,

$$C_1 \xrightarrow{d} \overline{\ell_1}.\mathbf{0}$$
$$C_2 \xrightarrow{d} \overline{\ell_2}.\mathbf{0}$$

Exercise: Implementing Minsky Programs with CCS

Minsky machines are computing devices working on so-called *counters* (aka. *registers*) that hold natural numbers. Finitely many lines of code alter the states of the counters. Minsky machines with at least two counters are Turing-complete. The following program adds the values of c_1 and c_2.

```
01: dec c2 : 02 : 03
02: inc c1 : 01
03: HALT
```

In CCS, we represent this program by

$$\boldsymbol{\nu}(i_1,i_2,d_1,d_2,z_1,z_2)\,(C^1 \mid C^2 \mid L_1)$$

with

$$L_1 \xrightarrow{\overline{d_2}} L_2 \qquad L_1 \xrightarrow{z_2} L_H \qquad L_2 \xrightarrow{\overline{i_1}} L_1 \qquad L_H \xrightarrow{\checkmark} L_H$$

where $\checkmark \in Names$.

Exercise: Implementing Minsky Programs with CCS

Theorem 33 CCS is Turing-complete. For every Minsky machine $\mathcal{M}$ there is a process $P(\mathcal{M})$ with a special constant L_H representing the halting line of $\mathcal{M}$ such that $\mathcal{M}$ terminates if and only if $P(\mathcal{M}) \xrightarrow{\tau}{}^{\star} \xrightarrow{\checkmark}$.

Everything *interesting* about CCS is undecidable.

The Bisimilarity Problem

Input Processes $p, q \in \mathsf{Pr}$.
Output Yes if and only if $p \simeq q$.

What if p and q came from very particular sets of processes?

What if p and q came from CCS, but not all of CCS?

$$p, q \in \mathsf{CCS}_{fin}$$

$$\mathsf{CCS}_{fin} = \mathsf{CCS}(\mathsf{Act}, \emptyset, \emptyset)$$

Recall: Bisimilarity in P for Finite LTSs

$\simeq_\omega := \bigcap_{i \geq 0} \simeq_i$

1. set $\simeq_0 = \mathcal{U}$
2. $p \simeq_{n+1} q$ for $n \geq 0$ if for all $a \in \mathsf{Act}$:
   a. for all p' with $p \xrightarrow{a} p'$, there is a q' with $q \xrightarrow{a} q'$ and $p' \simeq_n q'$;
   b. for all q' with $q \xrightarrow{a} q'$, there is a p' with $p \xrightarrow{a} p'$ and $p' \simeq_n q'$.

Theorem 21 $\simeq$ and $\simeq_\omega$ coincide on *image-finite* LTSs.

1. Finite LTSs are image-finite. **recall**
2. How hard is it to compute $\simeq$ on finite LTSs $(\mathsf{Pr}, \mathsf{Act}, \longrightarrow)$? i.e., $\simeq_\omega$
   - compute $\simeq_0 = \mathcal{U}$ $\mathcal{O}(|\mathsf{Pr}|^2)$
   - iteratively remove all pairs from $\simeq_i$ contradicting bisimulations $\not\rightsquigarrow \simeq_{i+1}$ $\mathcal{O}(|\mathsf{Pr}|^3)$
   - stop when nothing changes after at most $|\mathsf{Pr}|^2$ removals

Recall: Bisimilarity in P for Finite LTSs

Lemma 34 For all processes $P \in \mathsf{CCS}_{fin}$, $G(P)$ is finite and finitely branching.

Proof: By induction on the structure of P.

Base $P = \mathbf{0}$ has a single node (i.e., process) and no transitions.

Step Let $Q_1, Q_2 \in \mathsf{CCS}_{fin}$ with finite(ly branching) $G(Q_i)$ $(i = 1, 2)$.

$P = \mu.Q_1$ *easy*

$P = \nu a\, Q_1$ *easy*

$P = Q_1 + Q_2$ *easy*

$P = Q_1 \mid Q_2$ recall, every process can be turned into *head normal form*; for CCS_{fin} processes, repeated application eventually terminates with a process Q without any parallel composition operator.

∎

Recall: Bisimilarity in P for Finite LTSs

Theorem 35 $\simeq$ is in P for CCS_{fin} processes.

Axiomatizing $\simeq$ for CCS_{fin}

Decidability implies an algebraic characterization of bisimilarity in the shape of *axiomatizations*.

Axiomatizations are axioms that, incorporating equational reasoning, are sufficient to decide the equivalence.

1. use reflexivity, symmetry, and transitivity
2. use substitutivity by equivalent subterms

The System $\mathcal{SB}$

S1	$P + \mathbf{0} = P$
S2	$P + Q = Q + P$
S3	$P + (Q + R) = (P + Q) + R$
S4	$P + P = P$
R1	$\boldsymbol{\nu} a\, \mathbf{0} = \mathbf{0}$
R2 if $\mu \in \{a, \overline{a}\}$	$\boldsymbol{\nu} a\, \mu.P = \mathbf{0}$
R3 if $\mu \notin \{a, \overline{a}\}$	$\boldsymbol{\nu} a\, \mu.P = \mu.\boldsymbol{\nu} a\, P$
R4	$\boldsymbol{\nu} a\, (PQ) = \boldsymbol{\nu} a\, P + \boldsymbol{\nu} a\, Q$
E	

If $P \stackrel{\text{def}}{=} \sum_{0 \le i \le m} \mu_i.P_i$ and $P \stackrel{\text{def}}{=} \sum_{0 \le j \le n} \mu_j.P_j$, infer

$$P \mid P' = \sum_{0 \le i \le m} \mu_i.(P_i \mid P') + \sum_{0 \le j \le n} \mu_j.\big(P \mid P_j\big) + \sum_{\mu_i = \overline{\mu_j}} \tau.\big(P_i \mid P_j\big)$$

.

A Note on Locality

- $\simeq$ does not change if we switch from single symbols to sequences
- $\simeq_\omega$, however, does
- $P \cong_0 Q$ for all processes P and Q
- $P \cong_{i+1} Q$ if, for all $w \in \mathsf{Act}^\star$,
  1. $P \xrightarrow{w} P'$ implies $Q \xrightarrow{w} Q'$ and $P' \cong_i Q'$;
  2. $Q \xrightarrow{w} Q'$ implies $P \xrightarrow{w} P'$ and $P' \cong_i Q'$;
- the limit is $\cong_\omega := \bigcap_{i \geq 0} \cong_i$ and coincides with $\simeq$ for image-finite processes

Outlook

1. How efficient can we compute bisimilarity? — May 19 Lecture
   - Bisimilarity for finite processes is P-complete
   - Surprisingly, $\cong_i$ for $i \geq 1$ is PSPACE-complete
2. What makes bisimilarity undecidable for CCS? — May 26 Lecture
   - Decidability for BPP and CFP as well
   - Undecidability for Petri nets