11.3 Multi-headed Automata

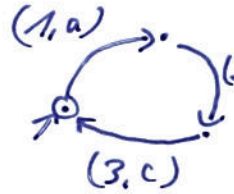
Goal: Develop an automaton model
- ↳ suitable for normal-form computations
- ↳ closed under regular intersection (to check for happens-before cycles)
- ↳ for which emptiness is easy to check

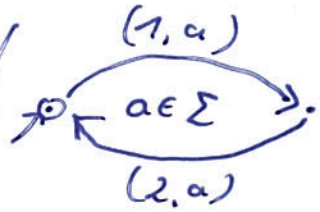
Idea:
- Encode a computation $\sigma_1 \dots \sigma_n$ by simultaneously guessing the parts σ_i.
- Automaton has a head for each part. Transition relation defines the head that produces a letter.

Definition:
- An *n-headed automaton over the (finite) alphabet Σ* is a tuple $A = (S, [1,n] \times \Sigma, \to, s_0, S_F)$ with
  - finite set of *states* S, *initial state* $s_0 \in S$, *final states* $S_F \subseteq S$,
  - *transition relation* $\to \subseteq S \times ([1,n] \times \Sigma) \times S$.
- The *language of A* is

$$\mathcal{L}(A) := \{ \pi_2(\sigma\downarrow(\{1\} \times \Sigma) \dots \sigma\downarrow(\{n\} \times \Sigma)) \mid s_0 \xrightarrow{\sigma} s \in S_F \}$$

Here, $\pi_2((a_1,b_1) \dots (a_k,b_k)) := b_1 \dots b_k$ is the projection to the second component.

Example:
- $\{ a^n b^n c^n \mid n \in \mathbb{N} \} = \mathcal{L}($  $)$

- $\{ ww \mid w \in \Sigma^* \} = \mathcal{L}($  $)$

Both languages are known to be not context-free.

- $\{ w.w^{reverse} \mid w \in \Sigma^* \}$

The language is context-free but cannot be accepted by a multi-headed automaton.

Together, we note that context-free languages and the languages of multi-headed automata are incomparable.

Lemma (Closure under Regular Intersection):
Consider an n-headed automaton A and a finite automaton B over a common alphabet Σ.
There is an n-headed automaton $A \| B$ with

$$\mathcal{L}(A \| B) = \mathcal{L}(A) \cap \mathcal{L}(B).$$

Proof:

Idea:
- Guess the states of B that are reached, when the σ_i have been processed.
- Check the guess in the end.

Construction:

Let $A = (S, [1,n]_* \Sigma, \rightarrow_A, s_0, S_F)$
$B = (Q, \Sigma, \rightarrow_B, q_0, Q_F)$.

Define $A \| B := (\{s_{new}\} \cup S \times Q^n \times Q^n, \rightarrow, s_{new}, F)$

with

$$F := \{ (s, \overline{q_{init}}, \overline{q_{current}}) \mid \overline{q_{current}}(i) = \overline{q_{init}}(i+1) \text{ for all } 1 \le i < n \text{ and } \overline{q_{current}}(n) \in Q_F \text{ and } s \in S_F \}$$

$s_{new} \xrightarrow{\varepsilon} (s_0, \overline{q}, \overline{q})$ for all $\overline{q} \in Q^n$ where $\overline{q}(1) = q_0$

and

$(s, \overline{q_{init}}, \overline{q_{current}}[i := q]) \xrightarrow{a} (s', \overline{q_{init}}, \overline{q_{current}}[i := q'])$,

if $s \xrightarrow{(i,a)}_A s'$ and $q \xrightarrow{a}_B q'$.

Remark:
Multi-headed automata are not
effectively closed under intersection nor under shuffle.

Proof:
Reduce Post's correspondence problem (PCP)
to an intersection of multi-headed automata:
the PCP instance has a solution
iff the intersection is not empty.
Since emptiness for MHA is decidable,
the intersection can at least not be computable. □

Lemma:
Emptiness for MNA is NL-complete.

Proof:
Emptiness amounts to disproving
the existence of a path from initial to final state.
This problem is known as $\overline{PATH}$.
It is in NL by the Immerman-Szelepcsenyi theorem. □

11.4 From Normal-Form Computations to MHA

Goal: Encode the normal-form computations $\tau \in C_{PSO}(P)$ as the language of a multi-headed automaton.

Construction:

Consider program P.

Define the 3-headed automaton

$$\mathcal{A}_P := (S, [1,3] \times \Gamma_{CT}, \longrightarrow, s_0, S)$$

where

$$S := \{ (pc, val, buf, pwt) \mid$$

$$pc : TID \to LOC$$

$$val : VAR \to DOM$$

$$buf : TID \times DOM \to \underbrace{DOM}_{finite}$$

$$pwt : TID \times DOM \to \{1,2,3\} \}.$$

Idea:

pc = current control state (in τ_i)

val = current variable valuation (in τ_i)

buf = last buffered store per thread pr address

pwt = for each thread and each address
part of the computation
where to generate next store

$$s_0 := \{ (pc_0, val_0, buf_\perp, pwt_1) \mid$$

$$pc_0(t) := l_0^t \qquad val_0(a_t) := 0$$

$$buf_\perp(t,a) := \perp \qquad pwt_1(t,a) := 1 \}.$$

The transition relation is again defined by rules.
In the following, let $pc(t) = l$ and set $pc' := pc[t := l']$.

$$\text{(ldcwly)}\quad \frac{l: r \leftarrow mem[r'] \text{ goto } l',\ val(r') = a,\ buf(t,a) = v}{(pc, val, buf, pwt) \xrightarrow{(1,(t,ld,a))} (pc', val[r := v], buf, pwt)}$$

$$\text{(ldmem)}\quad \frac{l: r \leftarrow mem[r'] \text{ goto } l',\ val(r') = a,\ buf(t,a) = \bot,\ val(a) = v}{(pc, val, buf, pwt) \xrightarrow{(1,(t,ld,a))} (pc', val[r := v], buf, pwt)}$$

$$\text{(stmem)}\quad \frac{l: mem[r] \leftarrow r' \text{ goto } l',\ val(r) = a,\ val(r') = v,\ pwt(t,a) = 1}{(pc, val, buf, pwt) \xrightarrow{(1,\oplus,isu,a)).(1,(t,st,a))} (pc', val[a := v], buf, pwt)}$$

$$\text{(stdelay)}\quad \frac{l: mem[r] \leftarrow r' \text{ goto } l',\ val(r) = a,\ val(r') = v,\ pwt(t,a) = i > 1}{(pc, val, buf, pwt) \xrightarrow{(1,(t,isu,a)).(i,(t,st,a))} (pc', val, buf[(t,a) := v], pwt)}$$

(local) is easy

$$\text{(mfence)}\quad \frac{l: \underline{mfence} \text{ goto } l',\ buf(t,a) = \bot \text{ for all } a \in DOM}{(pc, val, buf, pwt) \xrightarrow{(1,(t,mfence))} (pc', val, buf, pwt)}$$

$$\text{(pwt)}\quad \frac{pwt(t,a) = i,\ j > i}{(pc, val, buf, pwt) \xrightarrow{\varepsilon} (pc, val, buf, pwt[(t,a) := j])}$$

Claim:

$$\mathcal{L}(\Pi_P) = \{\, \tau \in C_{PSO}(P) \mid \tau \text{ is in normal form} \,\}.$$

10.5 Finding Cycles

Goal: Check if a normal-form computation in $\mathcal{L}(\Pi_P)$ has cyclic happens-before.

Problem: Cycles in normal-form computations can be complicated.

Idea: Whenever a computation has a cycle,
it has one where each thread that is involved
is entered and left precisely once.

Lemma:

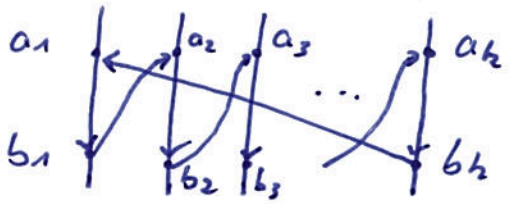
- Computation $\tau \in C_{PSO}(P)$ is violating iff
there is a cycle

$$a_1 \xrightarrow{*}_{po} b_1 \xrightarrow[\substack{src\\st\\cf}]{} a_2 \xrightarrow{*}_{po} b_2 \xrightarrow[\substack{src\\st\\cf}]{} a_3 \dots b_k \xrightarrow[\substack{src\\st\\cf}]{} a_1,$$

with $thread(a_i) = thread(b_i) = t_i \neq t_j = thread(a_j) = thread(b_j)$ for all $i \neq j$

- We call the sequence $t_1 \dots t_k$ the type of the cycle.

Illustration:

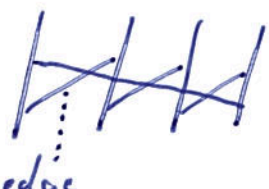


Problem: Even such simpler cycles are hard to detect:
the actions constituting the cycle
may not occur in this order in the computation.

Idea: (1) Guess b_i and a_{i+1} in the computation
(2) Check that $b_i \xrightarrow[st,cf]{src,} a_{i+1}$ holds.

For (1): Extend Π_P to Π_P^* so as to mark b_i and a_{i+1}
by $m \in \mathbb{P}(\{enter, leaves\})$.

For (2): Use a regular intersection,
i.e., one finite automaton for each edge in



Let A^{cyc} be the product of the finite automata A^{edge} for all edges in cycle type cyc.

Theorem:

P is robust iff $L(A_P^*) \cap L(A^{cyc}) = \emptyset$ for all cycle types cyc.