Lecture Verification of Infinite-State Systems

Winter 2026/2027

Prof. Dr. Karin Quaas

Institut für Informatik
Universität Leipzig

Lecture 3 - Register Automata

Schedule

	Thursday 9-11	Friday 13-15
12.10.-18.10.		Lecture 1: Introduction / Register Automata
19.10.-25.10.	Exercise 1	Lecture 2: Register Automata
26.10.-1.11.	Exercise 2	Lecture 3: Register Automata
2.11.-8.11.	Exercise 3	Lecture 4: Register Automata
9.11.-15.11.	Exercise 4	Lecture 5: Register Automata / Two-Counter Mac
16.11.-22.11.	Exercise 5	Lecture 6: Register Automata
23.11.-29.11.	Exercise 6	Lecture 7: Freeze LTL
30.11.-6.12.	Exercise 7	Lecture 8: Freeze LTL
7.12.-13.12.	Exercise 8	Lecture 9: Freeze LTL / Alternating RA
14.12.-20.12.	Exercise 9	Lecture 10: Alternating RA / WSTS
4.1.-10.1.	Exercise 10	Lecture 11: Alternating RA / WSTS
11.1.-17.1.	Exercise 11	Lecture 12: Alternating RA / WSTS
18.1.-24.1.	Exercise 12	Lecture 13: Alternating RA / Ackermann
25.1.-31.1.	Exercise 13	Lecture 14:
1.2.-7.2.	Exercise 14	Examination

Operations on Data Languages

Given $L, L' \subseteq (\Sigma \times \mathbb{N})^*$, we define

- the union of L and L' by
  $L \cup L' = \{w \in (\Sigma \times \mathbb{N})^* \mid w \in L \text{ or } w \in L'\}$
- the intersection of L and L' by
  $L \cap L' = \{w \in (\Sigma \times \mathbb{N})^* \mid w \in L \text{ and } w \in L'\}$
- the complement of L (also written $\overline{L}$) by
  $(\Sigma \times \mathbb{N})^* \setminus L = \{w \in (\Sigma \times \mathbb{N})^* \mid w \notin L\}$

Closure Properties

Theorem 3

Let $L_1, L_2 \subseteq (\Sigma \times \mathbb{N})^*$ be two data regular languages. Then $L_1 \cup L_2$ and $L_1 \cap L_2$ are data regular, too.

- For the proof, we start with two register automata $\mathcal{A}_1 = (\mathcal{L}^1, \Sigma, \mathcal{R}^1, \mathcal{L}^1_{\text{in}}, E^1, \mathcal{L}^1_{\text{acc}})$ such that $L(\mathcal{A}_1) = L_1$ and $\mathcal{A}_2 = (\mathcal{L}^2, \Sigma, \mathcal{R}^2, \mathcal{L}^2_{\text{in}}, E^2, \mathcal{L}^2_{\text{acc}})$ and $L(\mathcal{A}_2) = L_2$
- Without loss of generality, we may assume $\mathcal{L}^1 \cap \mathcal{L}^2 = \emptyset$ and $\mathcal{R}^1 \cap \mathcal{R}^2 = \emptyset$
- The construction of $\mathcal{A}_\cup$ with $L(\mathcal{A}_\cup) = L_1 \cup L_2$ is simple: set $\mathcal{A}_\cup = (\mathcal{L}^1 \cup \mathcal{L}^2, \Sigma, \mathcal{R}^1 \cup \mathcal{R}^2, \mathcal{L}^1_{\text{in}} \cup \mathcal{L}^2_{\text{in}}, E^1 \cup E^2, \mathcal{L}^1_{\text{acc}} \cup \mathcal{L}^2_{\text{acc}})$
- $\mathcal{A}_\cap$ with $L(\mathcal{A}_\cap) = L_1 \cap L_2$ is constructed via the product construction of $\mathcal{A}_1$ and $\mathcal{A}_2$ (next slide)

Product Construction

Let $\mathcal{A}_1 = (\mathcal{L}^1, \Sigma, \mathcal{R}^1, \mathcal{L}^1_{\text{in}}, E^1, \mathcal{L}^1_{\text{acc}})$ and $\mathcal{A}_2 = (\mathcal{L}^2, \Sigma, \mathcal{R}^2, \mathcal{L}^2_{\text{in}}, E^2, \mathcal{L}^2_{\text{acc}})$.

Define $\mathcal{A}_\cap = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L}_{\text{acc}})$ by

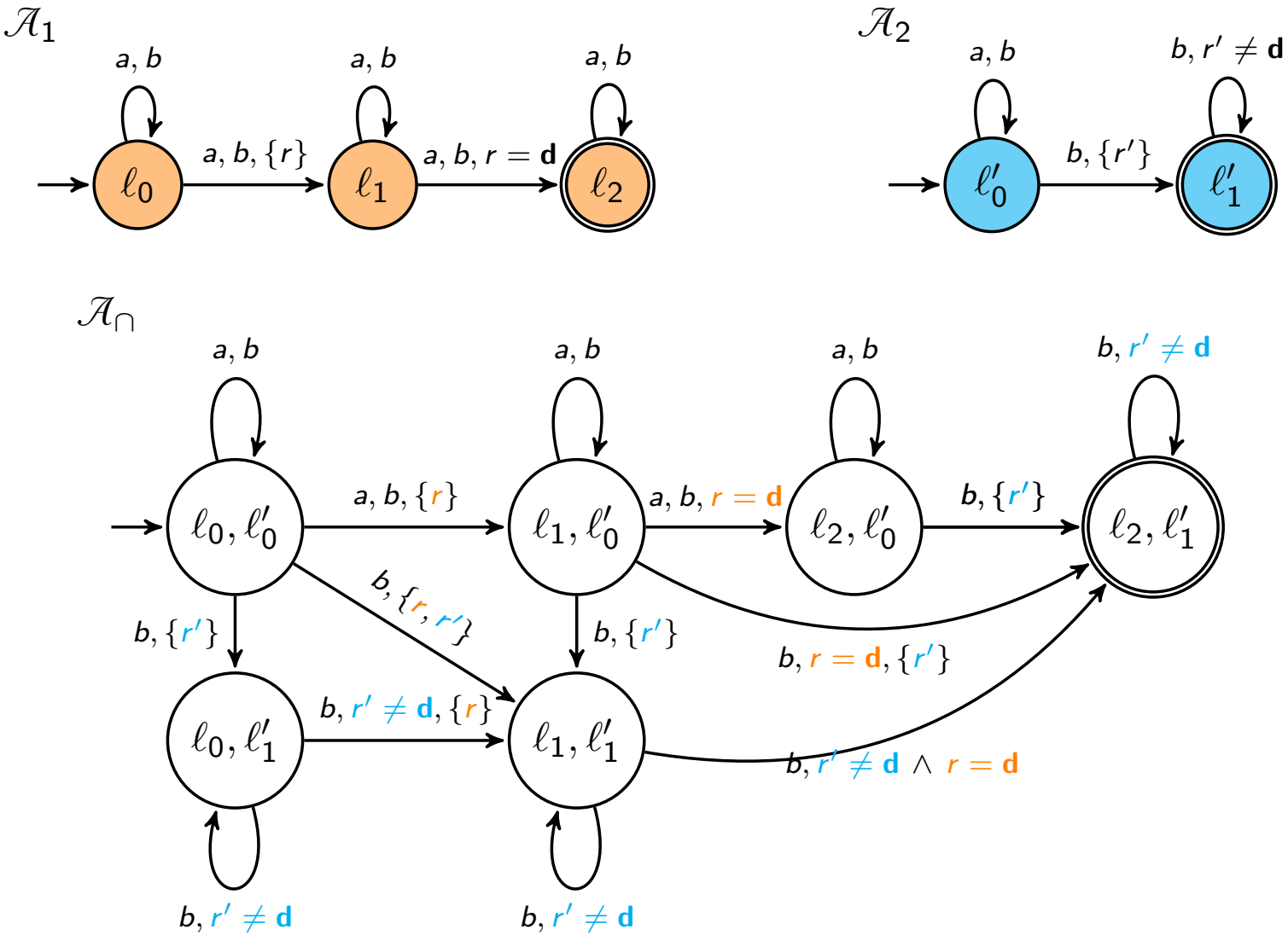
- $\mathcal{L} = \mathcal{L}^1 \times \mathcal{L}^2$
- $\mathcal{R} = \mathcal{R}^1 \cup \mathcal{R}^2$
- $\mathcal{L}_{\text{in}} = \mathcal{L}^1_{\text{in}} \times \mathcal{L}^2_{\text{in}}$ and $\mathcal{L}_{\text{acc}} = \mathcal{L}^1_{\text{acc}} \times \mathcal{L}^2_{\text{acc}}$
- $E = \{((\ell_1, \ell_2), a, \phi_1 \wedge \phi_2, \lambda_1 \cup \lambda_2, (\ell'_1, \ell'_2)) \mid$ there exists $(\ell_i, a, \phi_i, \lambda_i, \ell'_i) \in E^i$ for all $i \in \{1, 2\}\}$

For all data words $w \in (\Sigma \times \mathbb{N})^*$,

there exists an initialized accepting run of $\mathcal{A}_\cap$ on w, $\iff$ there exists an initialized accepting run of $\mathcal{A}_1$ on w and there exists an initialized accepting run of $\mathcal{A}_2$ on w.

Hence $L(\mathcal{A}_\cap) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.

Product Construction Example



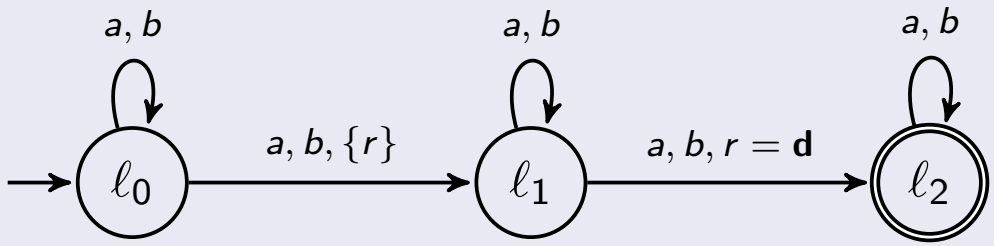
No Closure Under Complement

Theorem 4

There exist data regular languages $L \subseteq (\Sigma \times \mathbb{N})^*$ whose complement data language $\overline{L}$ is not data regular.

Proof.

The data language
$L_7 = \{(a_1, d_1) \dots (a_n, d_n) \mid d_i \neq d_j \text{ for all } 1 \leq i < j \leq n, n \geq 0\}$
(expressing the nonce (number only once) property) is the complement of the data language accepted by the following register automaton.



But L_7 cannot be accepted by any register automaton, see Lecture 2. □

Closure under Complement (Deterministic Case)

Theorem 5

Let $L \subseteq (\Sigma \times \mathbb{N})^*$ be a data language that can be accepted by a deterministic register automaton. Then its complement $\overline{L}$ can be accepted by a deterministic register automaton, too.

Proof.

By Lemma 2, there exists deterministic complete register automaton $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L}_{\text{acc}})$ with $L(\mathcal{A}) = L$. Define $\mathcal{B} = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L} \setminus \mathcal{L}_{\text{acc}})$. Let us prove $L(\mathcal{B}) = \overline{L(\mathcal{A})}$.

Let $w \in L(\mathcal{B})$. Then there exists an initialized accepting run ρ of $\mathcal{B}$ on w. By construction, ρ is also an initialized run of $\mathcal{A}$ on w, but not accepting. Since $\mathcal{A}$ is deterministic, there is no other initialized run of $\mathcal{A}$ on w. Hence $w \notin L(\mathcal{A})$. Hence $w \in \overline{L(\mathcal{A})}$.
Let $w \in \overline{L(\mathcal{A})}$. Since $\mathcal{A}$ is complete, there exists some initialized run of $\mathcal{A}$ on w (not accepting). The same run is then an initialized accepting run of $\mathcal{B}$ on w. Hence $w \in L(\mathcal{B})$. □

A data regular language that cannot be accepted by any deterministic register automaton

Theorem 6

There exist data regular languages that cannot be accepted by any deterministic register automaton.

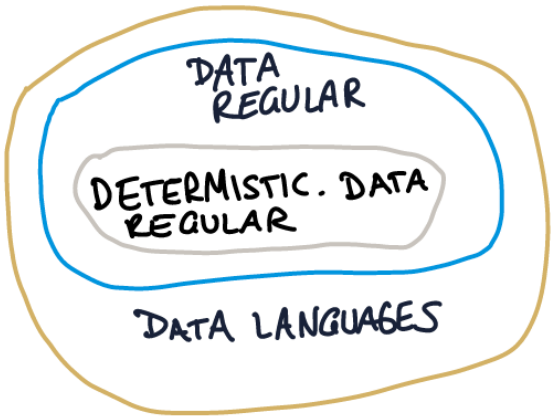
That is, we cannot determinize register automata, in general.

Proof.

The complement of L_7 is data regular (see proof of Theorem 4). Let us prove that it cannot be accepted by a deterministic register automaton. Towards contradiction, suppose there exists a deterministic register automaton that accepts the complement of L_7. By Theorem 5, L_7 is data regular, contradiction. □

Overview Closure Properties and Determinizability

	Finite Automata	Register Automata
Closure under Union	YES	YES (Theorem 3)
Closure under Intersection	YES	YES (Theorem 3)
Closure under Complement	YES	NO (Theorem 4)
Completable	YES	YES (Lemma 2)
Determinizable	YES	NO (Theorem 6)



- A data language that is not data regular: $\{(a_1, d_1) \dots (a_n, d_n) \mid d_i \neq d_j \text{ for all } 1 \leq i < j \leq n, n \geq 0\}$
- A data regular language that is not deterministically data regular: $\{(a_1, d_1) \dots (a_n, d_n) \mid d_i = d_j \text{ for some } 1 \leq i < j \leq n, n \geq 2\}$

The Nonemptiness Problem

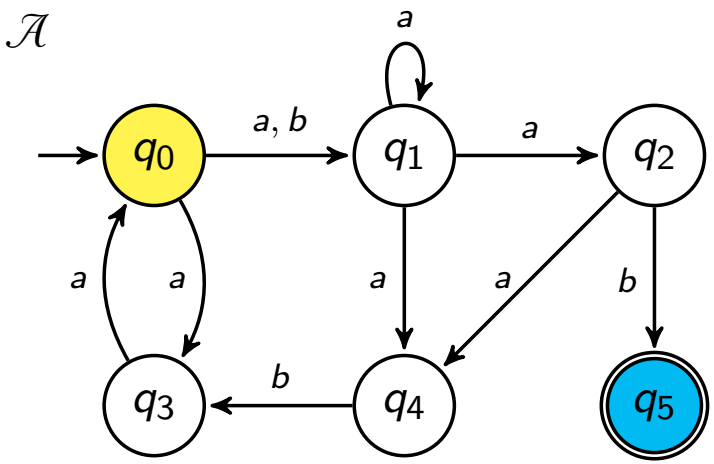
The Nonemptiness Problem	
Input:	A register automaton $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L}_{\text{acc}})$
Question:	$L(\mathcal{A}) \neq \emptyset$?

- This is a decision problem
- Is this problem decidable?
- If yes, what is the computational complexity?

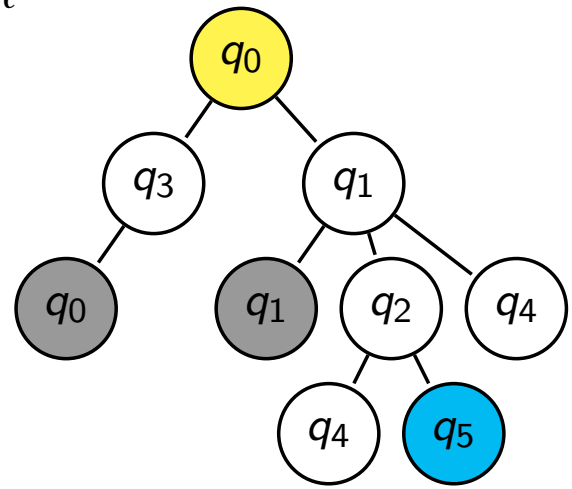
Recall that the nonemptiness problem for finite automata is decidable in **P** (and, even better, it is **NL**-complete).

Also note that deciding nonemptiness basically reduces to deciding whether there exists some initialized accepting run of the automaton on some word.

Nonemptiness for Finite Automata is in **P**



Unfolding of $\mathcal{A}$



Clearly, $L(\mathcal{A}) \neq \emptyset$ iff there exists run of $\mathcal{A}$ starting in q_0 ending in q_5.

Compute a finite prefix of the unfolding of $\mathcal{A}$ with respect to q_0.

The tree is finite branching and the height of this tree is bounded by the number of states of $\mathcal{A}$.

In general, this can be done (also for more than one initial and accepting state) in polynomial time in the size of $\mathcal{A}$.

Nonemptiness for Finite Automata is in **NL**

We can even do better!

A decision problem L is in **NLOGSPACE** (aka **NL**) if L can be decided by a nondeterministic Turing machine that uses working tape space logarithmic in the length k of the input.

Note that for a positive instance of the decision problem there exists at least one accepting computation of the Turing machine (but not every computation must be accepting); for a negative instance of the decision problem there does not exist any accepting computation.

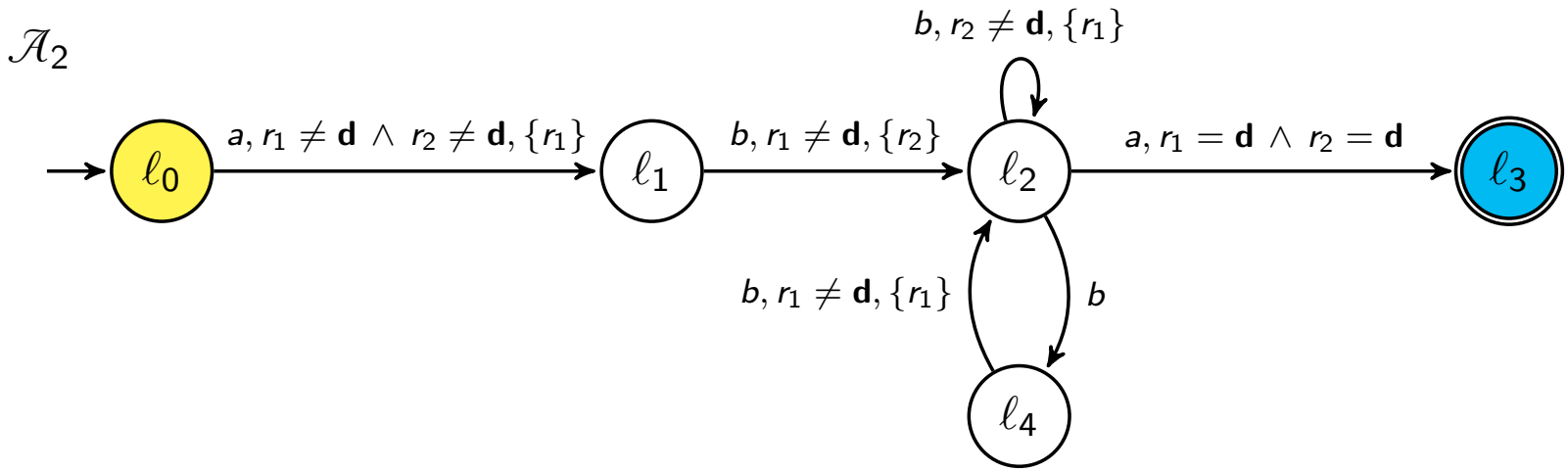
NL Algorithm for Nonemptiness

Let $\mathcal{A} = (Q, \Sigma, Q_{\text{in}}, \delta, Q_{\text{acc}})$ be a finite automaton.

1: choose $q \in Q_{\text{in}}$
2: $x \leftarrow q$
3: **while** $x \notin Q_{\text{acc}}$ **do**
4: choose a transition $(x, a, p) \in \delta$
5: $x \leftarrow p$
6: **end while**
7: **return** YES

The nondeterministic algorithm can be implemented by a nondeterministic Turing machine that only uses logarithmic space as it only needs to store the current state x on its tape (encoded in binary, assuming that $Q = \{1, \ldots, n\}$).

Nonemptiness for Register Automata - Example

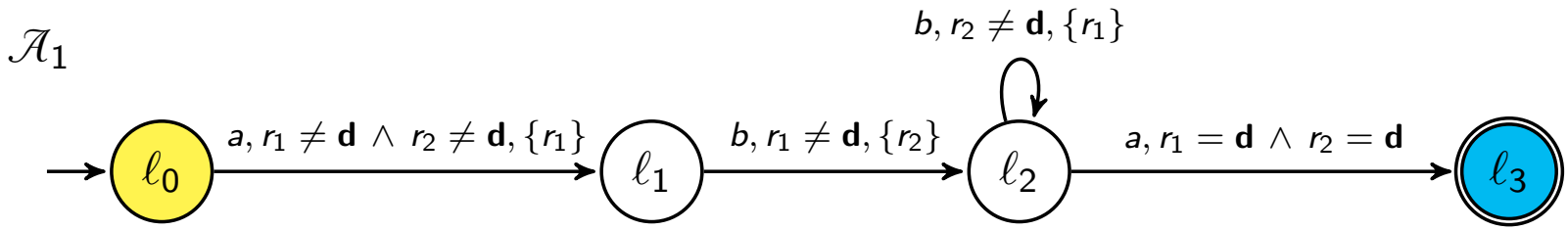


- $L(\mathcal{A}_2) \neq \emptyset$
- For instance, the following is an initialized accepting run of $\mathcal{A}_2$

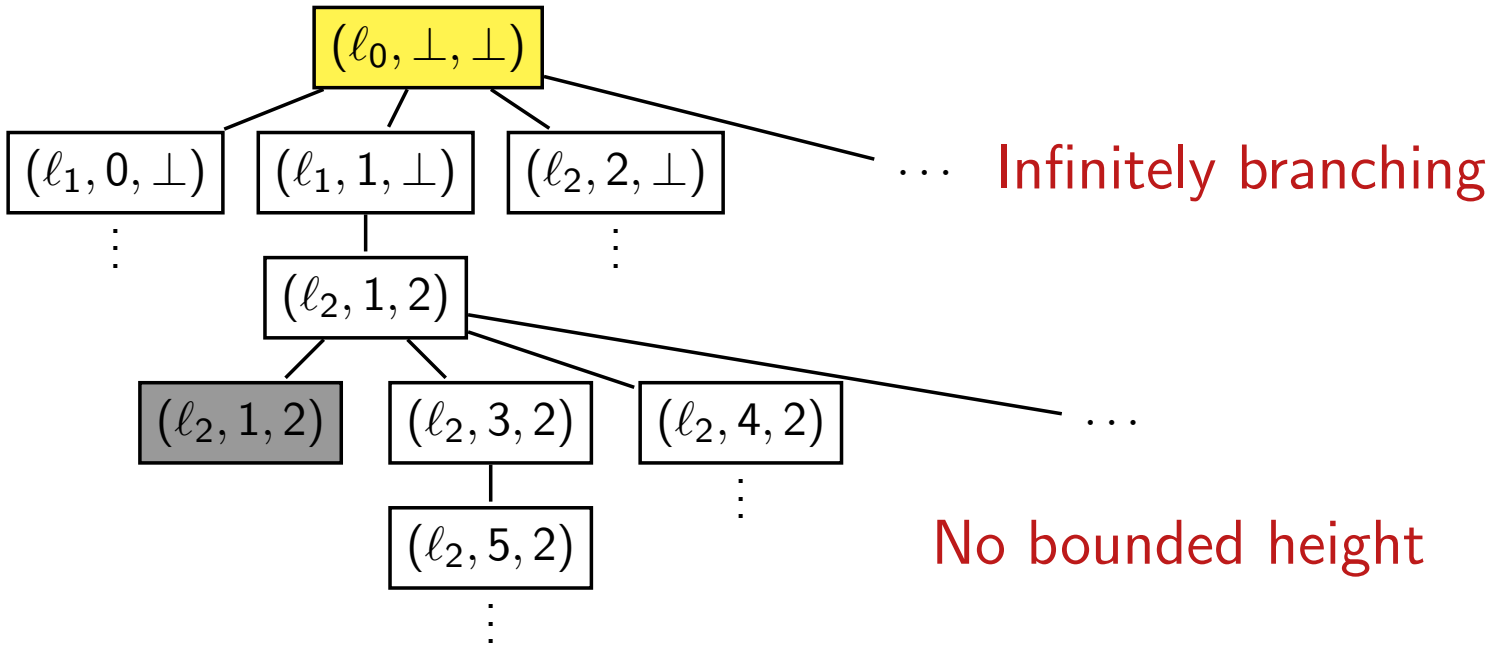
$$(\ell_0, \bot, \bot) \xrightarrow{(a,1)} (\ell_1, 1, \bot) \xrightarrow{(b,2)} (\ell_2, 1, 2) \xrightarrow{(b,2)} (\ell_4, 1, 2) \xrightarrow{(b,2)} (\ell_2, 2, 2) \xrightarrow{(a,2)} (\ell_3, 2, 2)$$

- But note that every initialized accepting run must visit ℓ_2 at least twice (as opposed to finite automata)

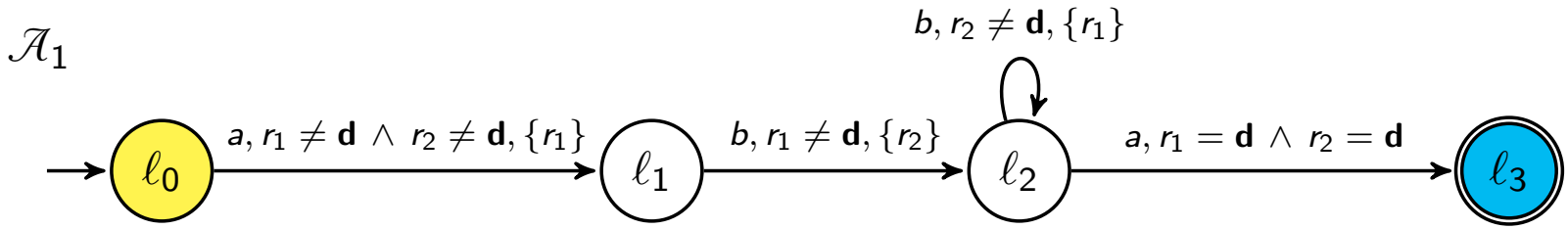
Nonemptiness for Register Automata - Example



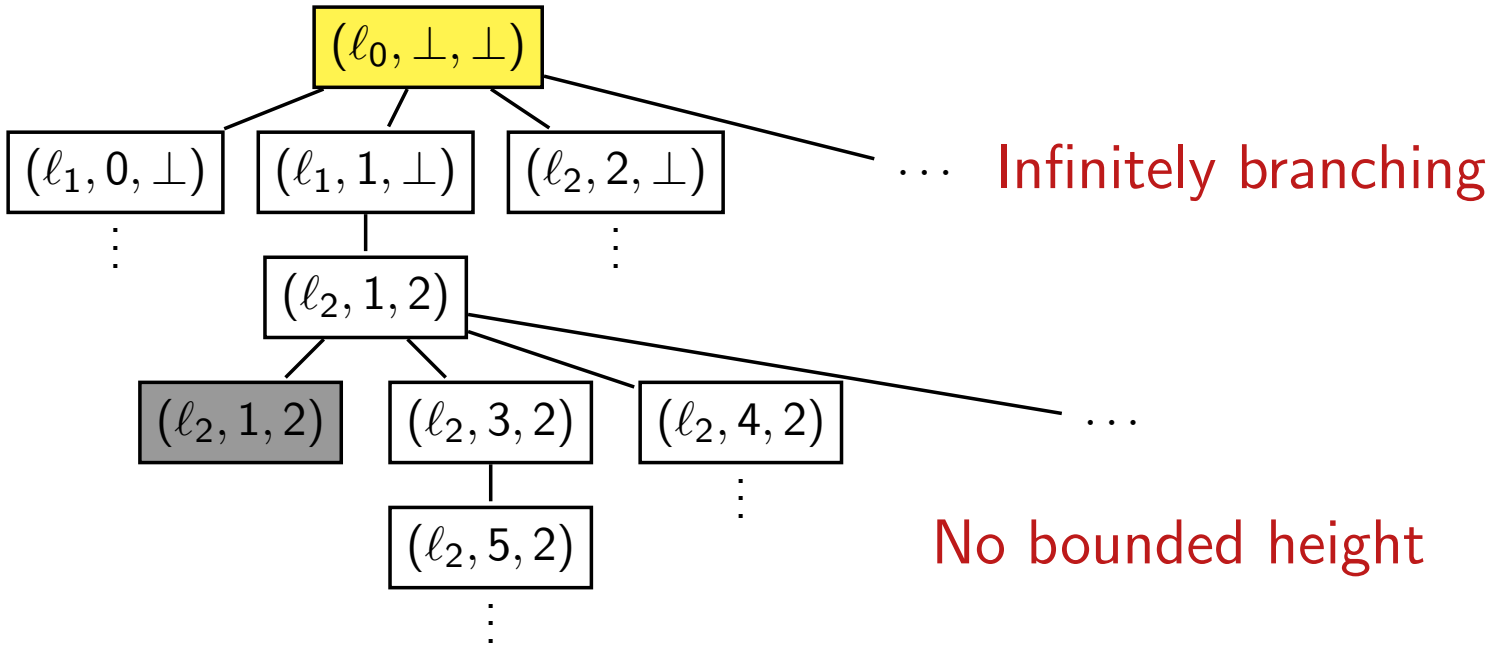
Let us borrow the idea for finite automata and compute the unfolding of $\mathcal{A}_1$ with respect to $(\ell_0, \bot, \bot)$



Nonemptiness for Register Automata - Example



Really???



Infinitely branching

No bounded height

Data doesn't matter I

Let $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$ be a bijective mapping such that $f(\perp) = \perp$.

Given a register valuation $\nu \in \mathbb{N}_\perp^{\mathcal{R}}$, we define $f(\nu)$ to be the valuation defined by $f(\nu)(r) := f(\nu(r))$ for all $r \in \mathcal{R}$.

Lemma 7

For all register valuations $\nu \in \mathbb{N}_\perp^{\mathcal{R}}$, data $d \in \mathbb{N}$, and register tests $\phi \in \Phi(\mathcal{R})$, if $\nu, d \models \phi$, then $f(\nu), f(d) \models \phi$.

Proof.

The proof is by induction on the structure of ϕ. For the induction base, suppose ϕ is atomic. We give the proof for ϕ of the form $r = \mathbf{d}$ (the other cases are similar). Suppose ϕ is of the form $r = \mathbf{d}$. By $\nu, d \models \phi$, we have $\nu(r) = d$. Hence $f(\nu(r)) = f(d)$. Hence $f(\nu), f(d) \models \phi$. For the induction step, suppose $\phi = \phi_1 \wedge \phi_2$. Then $\nu, d \models \phi$ implies $\nu, d \models \phi_1$ and $\nu, d \models \phi_2$. By induction hypothesis, we have $f(\nu), f(d) \models \phi_1$ and $f(\nu), f(d) \models \phi_2$. Hence $f(\nu), f(d) \models \phi$. □

Data doesn't matter I - Example

Let $\nu = (1, \perp)$ and $d = 2$. Define $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$ by

$$f(n) = \begin{cases} n & \text{if } n \in \mathbb{N}_\perp \setminus \{1, 2, 3\} \\ 2 & \text{if } n = 1 \\ 3 & \text{if } n = 2 \\ 1 & \text{if } n = 3 \end{cases}$$

Hence $f(\nu) = (2, \perp)$ and $f(d) = 3$, and $f(f(\nu)) = (3, \perp)$ and $f(f(d)) = 1$.

We clearly have $\nu, d \models r_1 \neq \mathbf{d}$.

By Lemma 7, $f(\nu), f(d) \models r_1 \neq \mathbf{d}$ and $f(f(\nu)), f(f(d)) \models r_1 \neq \mathbf{d}$

Indeed, $(2, \perp), 3 \models r_1 \neq \mathbf{d}$ and $(3, \perp), 1 \models r_1 \neq \mathbf{d}$.

Data doesn't matter II

Let $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$ be a bijective mapping such that $f(\perp) = \perp$.

Given a register valuation $\nu \in \mathbb{N}_\perp^{\mathcal{R}}$, we define $f(\nu)$ to be the valuation defined by $f(\nu)(r) := f(\nu(r))$ for all $r \in \mathcal{R}$.

Lemma 8

For all register valuations $\nu \in \mathbb{N}_\perp^{\mathcal{R}}$, sets $\lambda \subseteq \mathcal{R}$, and data $d \in \mathbb{N}$, we have $f(\nu[\lambda \leftarrow d]) = f(\nu)[\lambda \leftarrow f(d)]$.

Proof.

We distinguish two cases:

- Suppose $r \in \lambda$. Then $\nu[\lambda \leftarrow d](r) = d$. Then $f(\nu[\lambda \leftarrow d](r)) = f(d)$. Clearly, $f(\nu)[\lambda \leftarrow f(d)](r) = f(d)$, too.
- Suppose $r \notin \lambda$. Then $\nu[\lambda \leftarrow d](r) = \nu(r)$. Then $f(\nu[\lambda \leftarrow d](r)) = f(\nu(r))$. Clearly, $f(\nu)[\lambda \leftarrow f(d)](r) = f(\nu(r))$, too.

Data doesn't matter II - Example (ctd)

Let $\nu = (1, \perp)$, $d = 2$, and $\lambda = \{r_2\}$. Define $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$ by

$$f(n) = \begin{cases} n & \text{if } n \in \mathbb{N}_\perp \setminus \{1, 2, 3\} \\ 2 & \text{if } n = 1 \\ 3 & \text{if } n = 2 \\ 1 & \text{if } n = 3 \end{cases}$$

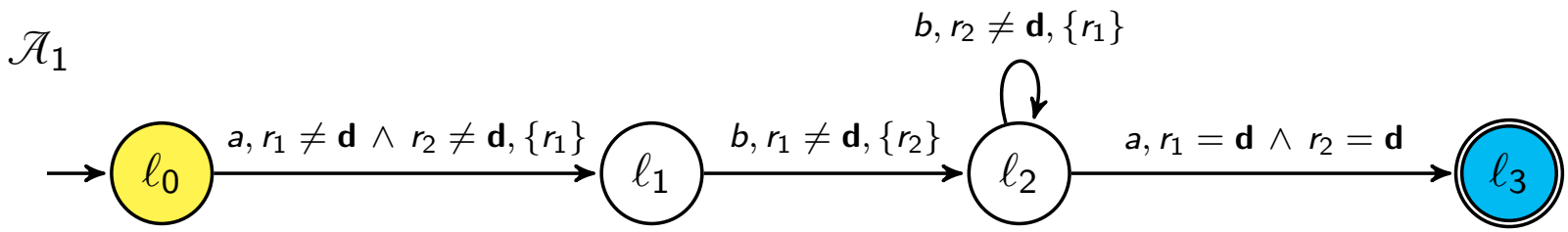
Hence $f(\nu) = (2, \perp)$ and $f(d) = 3$.

We clearly have $\nu[\lambda \leftarrow d] = (1, 2)$.

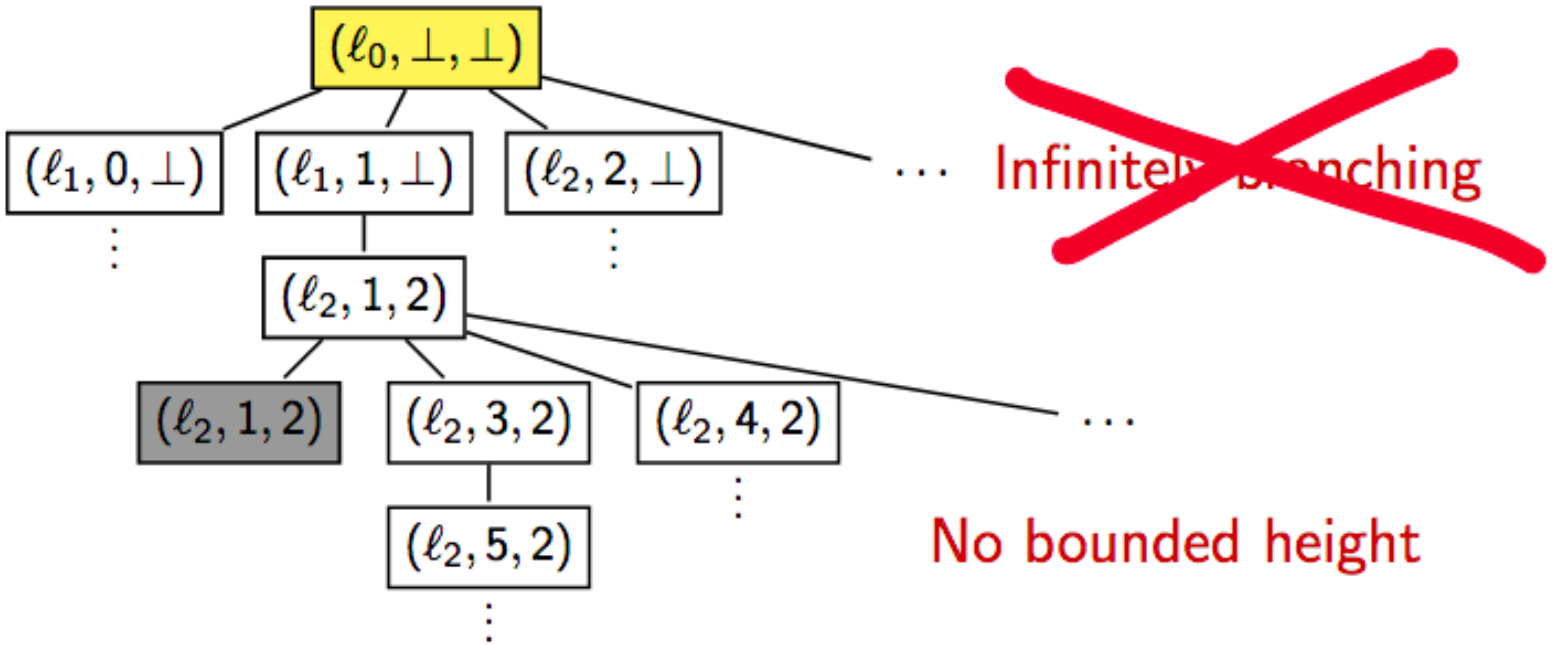
By Lemma 8, $f(\nu[\lambda \leftarrow d]) = f(\nu)[\lambda \leftarrow f(d)]$

Indeed, $f(1, 2) = (2, 3) = (2, \perp)[\lambda \leftarrow 3] = f(1, \perp)[\lambda \leftarrow f(2)])$

Towards Finite Branching



Lemmas 7 and 8 imply that we can avoid infinite branching



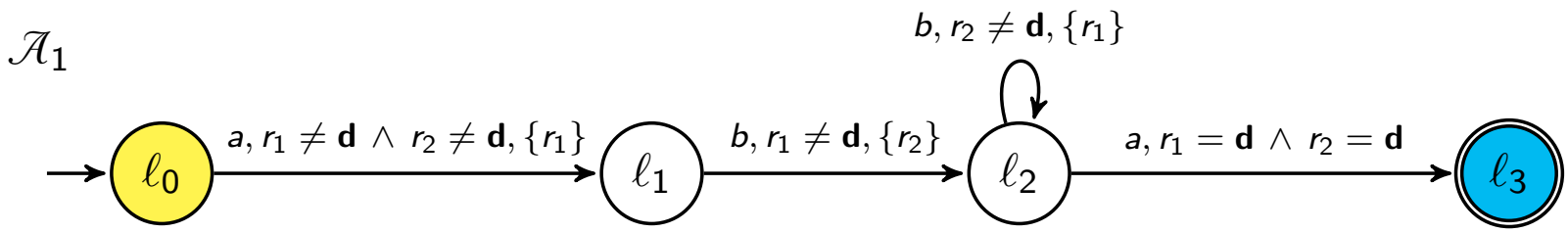
Quiz

Suppose you aim to build the unfolding of a register automaton $\mathcal{A}$ with 5 registers, starting from the current configuration $(\ell, (5, 3, 3, 5, 4))$.

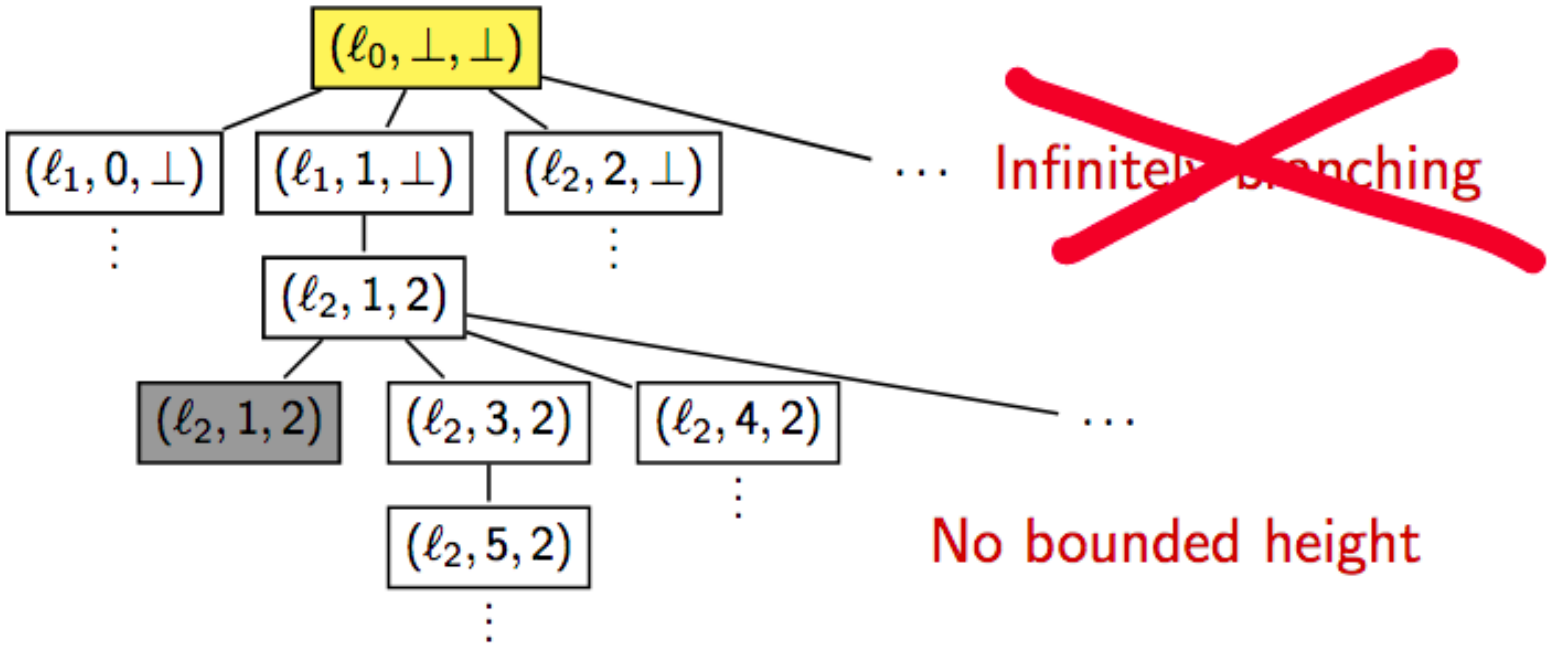
How many data are relevant (do matter), that is, what is an upper bound on the number of relevant data for computing representatives of all successor nodes?

1. 3
2. 4
3. 5
4. 6
5. One cannot say

Towards Finite Branching



Lemmas 7 and 8 imply that we can avoid infinite branching



Data doesn't matter

Let $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$ be a bijective mapping such that $f(\perp) = \perp$.

Given a data word $w = (a_1, d_1) \dots (a_n, d_n)$, we define $f(w)$ to be the data word defined by $f(w) := (a_1, f(d_1)) \dots (a_n, f(d_n))$.

Lemma 9

For all register automata $\mathcal{A}$ and w, if $w \in L(\mathcal{A})$, then $f(w) \in L(\mathcal{A})$.

Proof.

Suppose $w = (a_1, d_1) \dots (a_n, d_n) \in L(\mathcal{A})$. Then there exists an initialized accepting run of $\mathcal{A}$ on w of the form $(\ell_0, \nu_0) \xrightarrow{(a_1,d_1)} (\ell_1, \nu_1) \xrightarrow{(a_2,d_2)} \dots \xrightarrow{(a_n,d_n)} (\ell_n, \nu_n)$. For every $1 \leq i \leq n$, there exists some edge $(\ell_{i-1}, a_i, \phi_i, \lambda_i, \ell'_i)$ such that $\nu_{i-1}, d_i \models \phi_i$ and $\nu_i = \nu_{i-1}[\lambda_i \leftarrow d_i]$. By Lemma 7, $f(\nu_{i-1}), f(d_i) \models \phi_i$. By Lemma 8, $f(\nu_i) = f(\nu_{i-1})[\lambda_i \leftarrow f(d_i)]$. Hence $(\ell_0, f(\nu_0)) \xrightarrow{(a_1,f(d_1))} (\ell_1, f(\nu_1)) \xrightarrow{(a_2,f(d_2))} \dots \xrightarrow{(a_n,f(d_n))} (\ell_n, f(\nu_n))$ is an initialized accepting run on $f(w)$. □

L_1 cannot be accepted - Proof revisited

We give a simple proof that $L_1 = \{(a,5)(b,7)\}$ is not data regular. Towards contradiction, suppose there exists some register automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L_1$. Then $(a,5)(b,7) \in L(\mathcal{A})$. Define $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$ by

$$f(n) = \begin{cases} n & \text{if } n \in \mathbb{N}_\perp \setminus \{5,7\} \\ 7 & \text{if } n = 5 \\ 5 & \text{if } n = 7 \end{cases}$$

f is clearly a bijection. By Lemma 9, $f(w) = (a,7)(b,5) \in L(\mathcal{A})$, too. Contradiction.

Data Equivalence

Next we define an equivalence relation $\equiv$ on on $\mathbb{N}_\perp^{\mathcal{R}}$

$$\nu_1 \equiv \nu_2 \iff \text{there exists some bijective mapping } f : \mathbb{N}_\perp \to \mathbb{N}_\perp \text{ such that } f(\perp) = \perp \text{ and } \nu_2 = f(\nu_1)$$

For instance, $(0, 4, 4, \perp, 8) \equiv (1, 8, 8, \perp, 2) \equiv (3, 1, 1, \perp, 9)$

Let us check that $\equiv$ is indeed an equivalence relation. Let $\nu, \nu_1, \nu_2 \in \mathbb{N}_\perp^{\mathcal{R}}$ be register valuations.

Reflexitivity $\nu \equiv \nu$ by $\mathrm{id} : \mathbb{N}_\perp \to \mathbb{N}_\perp : x \mapsto x$ for all $x \in \mathbb{N}_\perp$.

Symmetry Suppose $\nu_1 \equiv \nu_2$. Hence there exists mapping f such that $\nu_2 = f(\nu_1)$. Since f is bijective, we have $\nu_1 = f^{-1}(\nu_2)$. Hence $\nu_2 \equiv \nu_1$.

Transitivity Suppose $\nu \equiv \nu_1$ and $\nu_1 \equiv \nu_2$. Hence there exist mappings f and g such that $\nu_1 = f(\nu)$ and $\nu_2 = g(\nu_1)$. Then $f \circ g$ is bijective, too, and $\nu_2 = g(f(\nu))$, so that indeed $\nu \equiv \nu_2$.

Abstract Register Valuations

The equivalence class of a register valuation ν is denoted by $[\nu]_{\equiv}$.

For example,

- $[(\bot, \bot, \bot, \bot)]_{\equiv} = \{(\bot, \bot, \bot, \bot)\}$
- $[(0, 4, 4, \bot)]_{\equiv} = \{(d_1, d_2, d_2, \bot) \mid d_1, d_2 \in \mathbb{N} \text{ such that } d_1 \neq d_2\}$
- $[(0, 4, 4, 3)]_{\equiv} = \{(d_1, d_2, d_2, d_3) \mid d_1, d_2, d_3 \in \mathbb{N} \text{ such that } d_1 \neq d_2, d_1 \neq d_3, d_2 \neq d_3\}$
- $[4]_{\equiv} = \mathbb{N}$

Note that every equivalence class (except for $[(\bot, \bot, \bot, \bot)]_{\equiv}$) contains infinitely many register valuations

An abstract register valuation is an equivalence class induced by $\equiv$

We use $\mathbb{A}(\mathcal{R})$ to denote the set of all abstract register valuations

Representation of Abstract Register Valuations

Let $\mathcal{R} = \{r_1, \ldots, r_n\}$.

Abstract register valuations in $\mathbb{A}(\mathcal{R})$ can be represented by *n*-types, that is, equivalence relations over $\mathcal{R}$.

For example, $[(0, 4, 4, 3)]_{\equiv}$ is represented by $\{\{r_1\}, \{r_2, r_3\}, \{r_4\}\}$

In register valuations $\nu : \mathcal{R} \to \mathbb{N}_{\perp}$, we distinguish between undefined register values $\nu(r) = \perp$ and register values $\nu(r)$ in $\mathbb{N}$.

We keep this distinction in abstract register valuations by marking the set of registers whose values are undefined

For instance, $[(0, 4, 4, \perp)]_{\equiv}$ is represented by $\{\{r_1\}, \{r_2, r_3\}, \{r_4\}_{\perp}\}$