Lecture Verification of Infinite-State Systems

Winter 2026/2027

Prof. Dr. Karin Quaas

Institut für Informatik
Universität Leipzig

Lecture 4 - 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

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:	$L(\mathcal{A}) \neq \emptyset$?

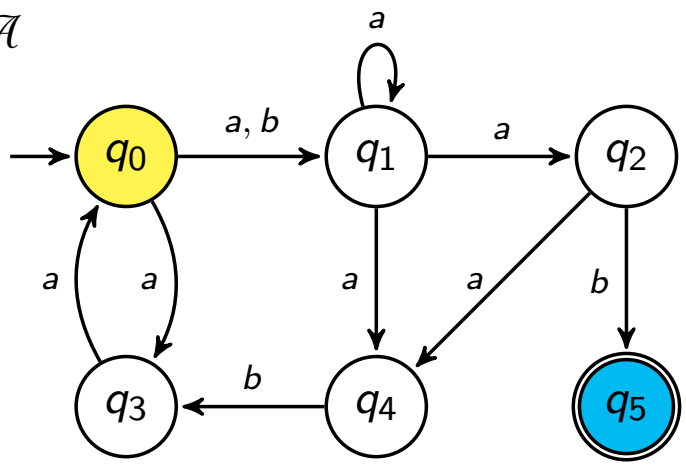
- This is a decision problem
- Is this problem decidable?
- If yes, what is the computational complexity?

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

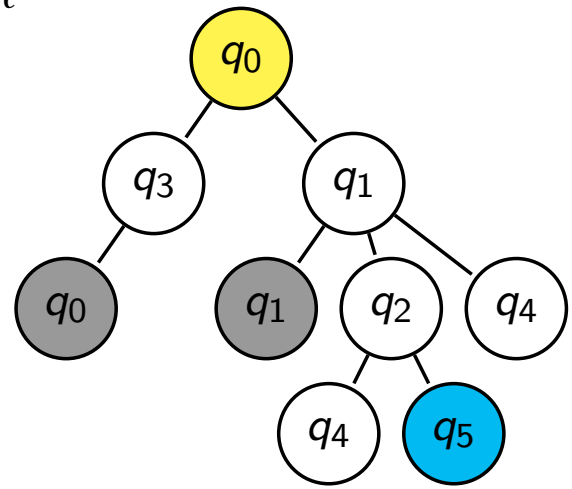
Also note that deciding nonemptiness basically reduces to deciding whether there exists some initialized accepting run of the automaton on some word.

Nonemptiness for Finite Automata is in **P**

$\mathcal{A}$



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 $\mathcal{A}$.

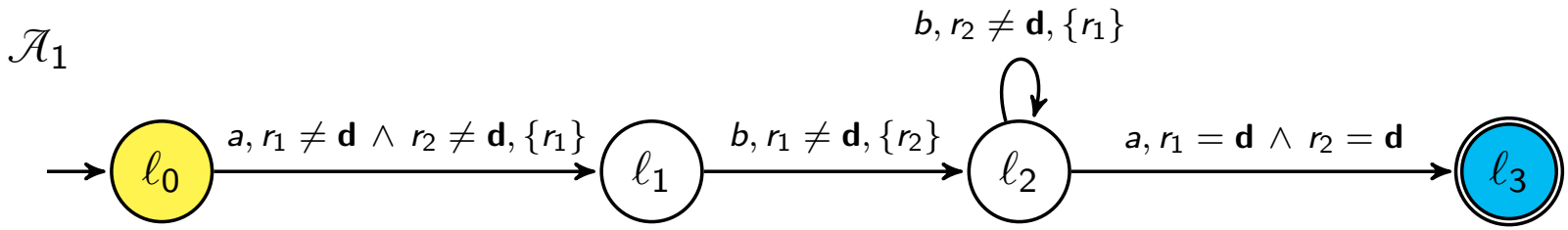
NL Algorithm for Nonemptiness

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

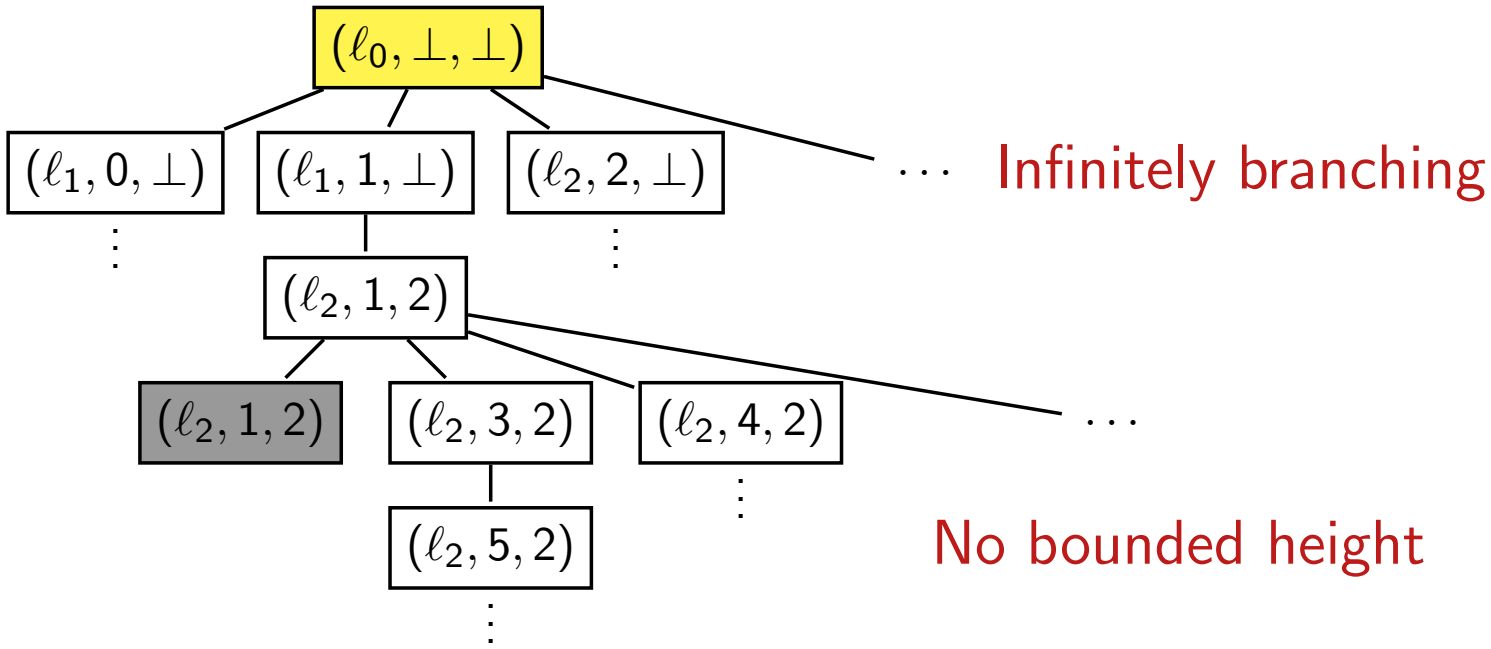
1: choose $q \in Q_{\text{in}}$
2: $x \leftarrow q$
3: **while** $x \notin Q_{\text{acc}}$ **do**
4: choose a transition $(x, a, p) \in \delta$
5: $x \leftarrow p$
6: **end while**
7: **return** YES

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 $Q = \{1, \ldots, n\}$).

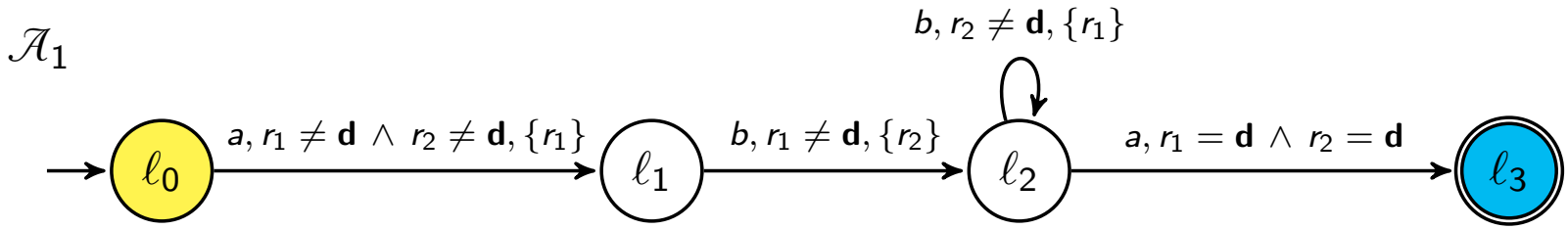
Nonemptiness for Register Automata - Example



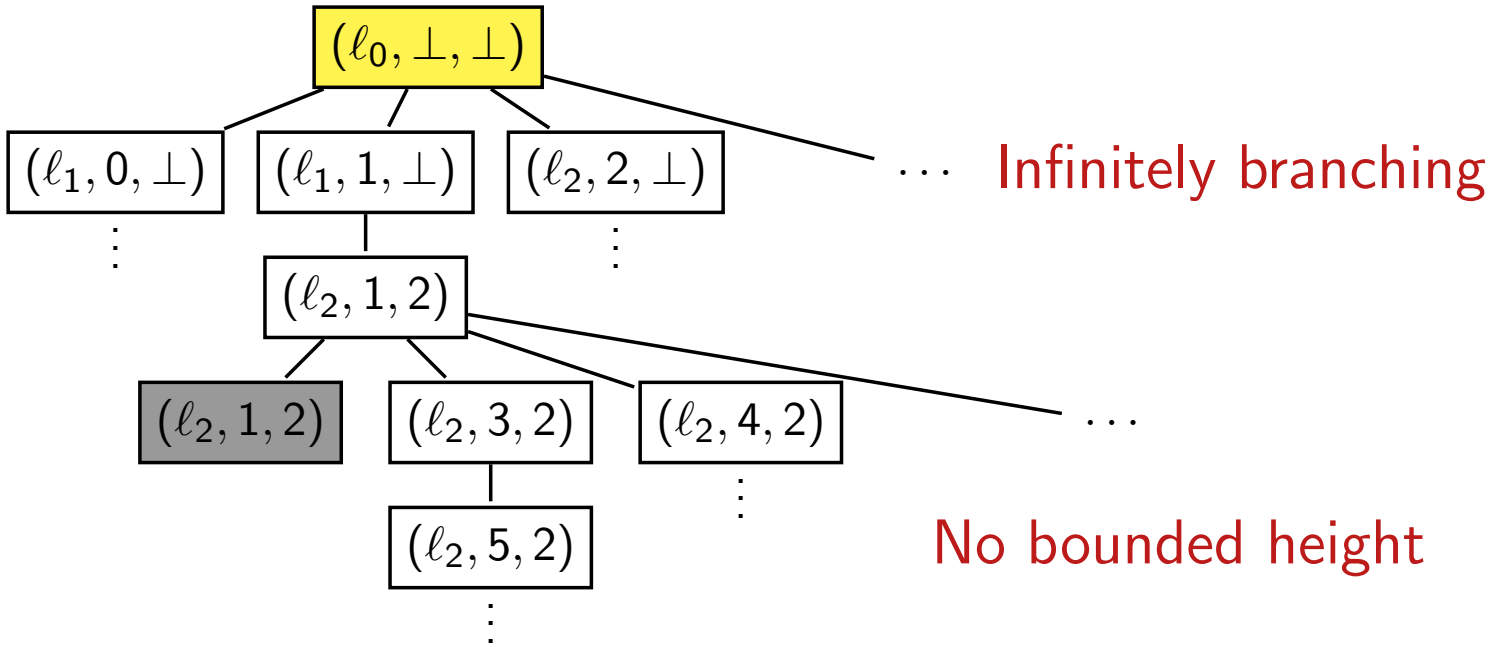
Let us borrow the idea for finite automata and compute the unfolding of $\mathcal{A}_1$ with respect to $(\ell_0, \bot, \bot)$



Nonemptiness for Register Automata - Example



Really???



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

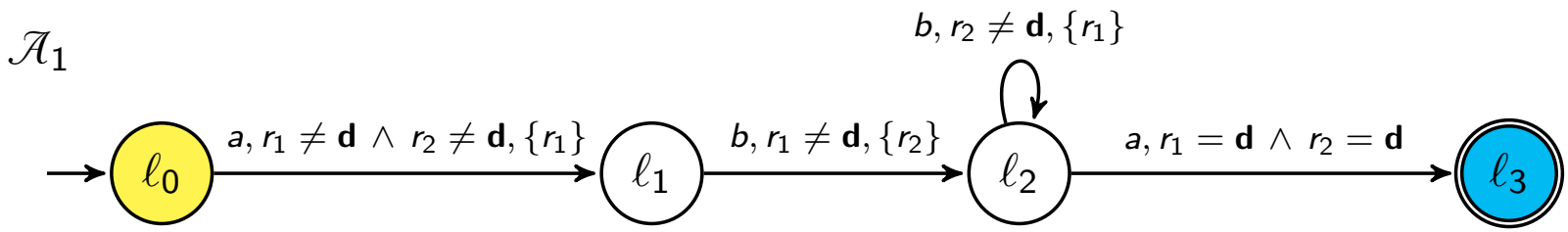
Let us check that $\equiv$ is indeed an equivalence relation. Let $\nu, \nu_1, \nu_2 \in \mathbb{N}_\perp^{\mathcal{R}}$ be register valuations.

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

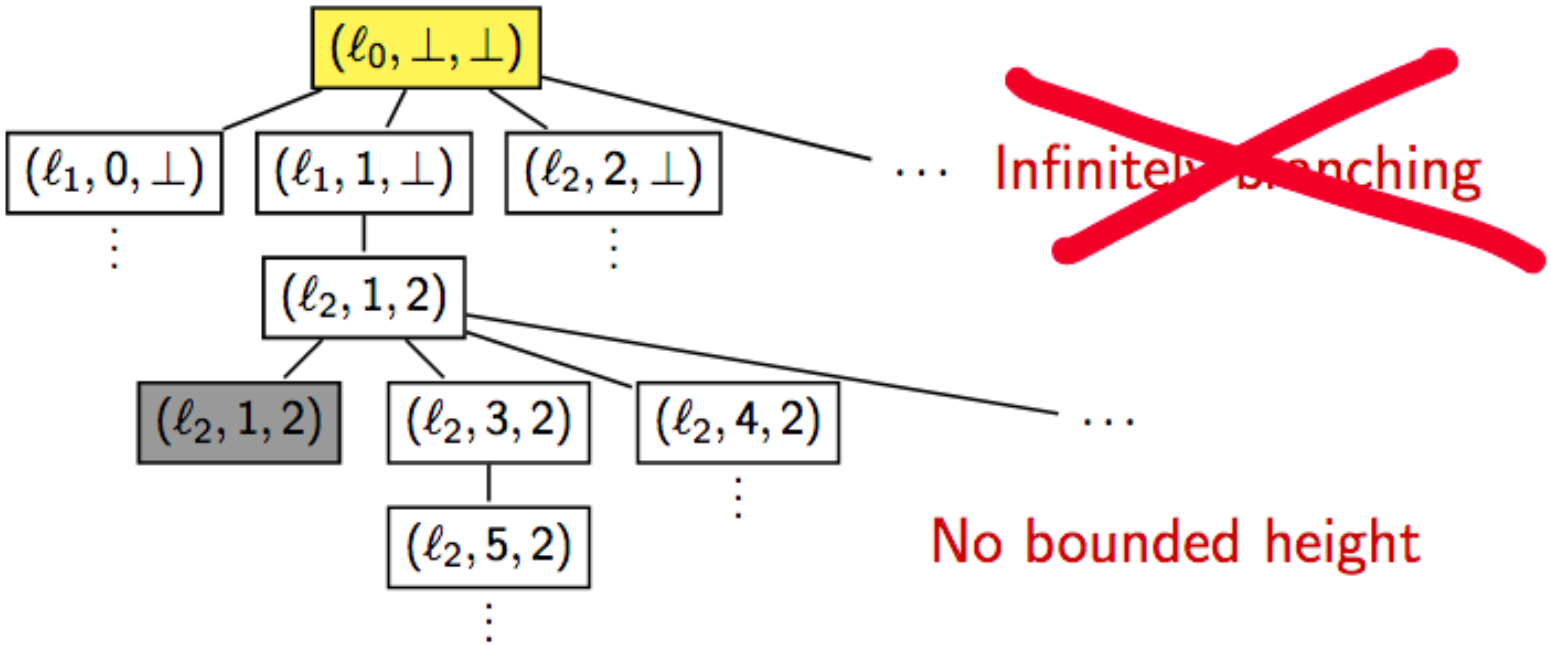
Symmetry Suppose $\nu_1 \equiv \nu_2$. Hence there exists mapping f such that $\nu_2 = f(\nu_1)$. Since f is bijective, we have $\nu_1 = f^{-1}(\nu_2)$. Hence $\nu_2 \equiv \nu_1$.

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 $\nu \equiv \nu_2$.

Towards Finite Branching



Lemmas 7 and 8 imply that we can avoid infinite branching



Towards Finite Branching

- If $(\ell, \nu_1) \xrightarrow{a,d} (\ell', \nu_1')$ and $\nu_1 \equiv \nu_2$, then there exists edge $(\ell, a, \phi, \lambda, \ell')$ such that $\nu_1, d \models \phi$ and $\nu_2 = \nu_1[\lambda \leftarrow d]$, and $\nu_2 = f(\nu_1)$ for some bijective mapping $f : \mathbb{N}_\perp \to \mathbb{N}_\perp$.
- By Lemma 7, for $\nu_1, d \models \phi$ implies $\nu_2, f(d) \models \phi$, so that $(\ell, \nu_2) \xrightarrow{a,f(d)} (\ell', \nu_2')$.
- By Lemma 8, we have $\nu_2' = \nu_2[\lambda \leftarrow f(d)] = f(\nu_1)[\lambda \leftarrow f(d)] = f(\nu_1[\lambda \leftarrow d]) = \nu_2$ so that $\nu_1' \equiv \nu_2'$.

In the following, we would like to construct a data abstract automaton whose behaviour depends on the equivalence classes induced by $\equiv$ (instead of concreteconcrete valuations).

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 $\{\{r_1\}, \{r_2, r_3\}, \{r_4\}_{\perp}\}$

Pingo Quiz

Let $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$.

Which 4-type is represented by the set $\{\{r_1\}, \{r_2, r_4\}, \{r_3\}, \{r_2\}_\perp\}$?

1. $(2, \perp, 1, \perp)$
2. $(1, 2, 3, 4)$
3. $(5, 7, 8, 7)$
4. None of them.

Number of Abstract Register Valuations

The number of n-types is bounded above by some exponential in n, called the Bell number $\mathbf{B}(n)$ of n

For instance, if $\mathscr{R} = \{r_1, r_2, r_3\}$, there are five 3-types:

- $\{\{r_1, r_2, r_3\}\}$,
- $\{\{r_1\}, \{r_2, r_3\}\}$,
- $\{\{r_1, r_2\}, \{r_3\}\}$,
- $\{\{r_1, r_3\}\{r_2\}\}$,
- $\{\{r_1\}, \{r_2\}, \{r_3\}\}$,

Some of these register values may be undefined, e.g., we need to add

- $\{\{r_1, r_2, r_3\}_\perp\}$,
- $\{\{r_1\}, \{r_2, r_3\}_\perp\}$ and $\{\{r_1\}_\perp, \{r_2, r_3\}\}$

That is, for each type at most $|\mathscr{R}|$ many versions

Hence, $|\mathbb{A}(\mathscr{R})|$ is bounded above by $\mathbf{B}(n) + n \cdot \mathbf{B}(n)$ (still exponential in n)

Abstract Transitions

We define an abstract transition relation $\rightarrow$ over $(\mathcal{L} \times \mathbb{A}(\mathcal{R})) \times \Sigma \times (\mathcal{L} \times \mathbb{A}(\mathcal{R}))$ as follows:

Let $(\ell, a, \phi, \lambda, \ell') \in E$ be an edge, and let A, A' be two abstract register valuations. Define

$$(\ell, A) \xrightarrow{a} (\ell', A') \quad :\Longleftrightarrow \quad \begin{array}{l}\text{there exist register valuation } \nu, \nu' \text{ and } d \in \mathbb{N} \text{ such} \\ \text{that } \nu \in A,\ \nu' \in\ A' \text{ and } (\ell, \nu) \xrightarrow{a,d} (\ell', \nu')\end{array}$$

Example 1

Let $(\ell, a, r_2 = \mathbf{d} \wedge r_3 \neq \mathbf{d}, \{r_1\}, \ell') \in E$.

- $(\ell, \{\{r_1, r_4\}, \{r_2, r_3\}\}) \xrightarrow{a} (\ell', ?)$
- $(\ell, \{\{r_1, r_3, r_4\}_\perp, \{r_2\}\}) \xrightarrow{a} (\ell', ?)$
- $(\ell, ?) \xrightarrow{a} (\ell', \{\{r_3\}, \{r_4\}, \{r_1\}, \{r_2\}\})$
- $(\ell, ?) \xrightarrow{a} (\ell', \{\{r_3\}, \{r_4\}, \{r_1, r_2\}\})$

Data Abstract Automaton

Theorem 10

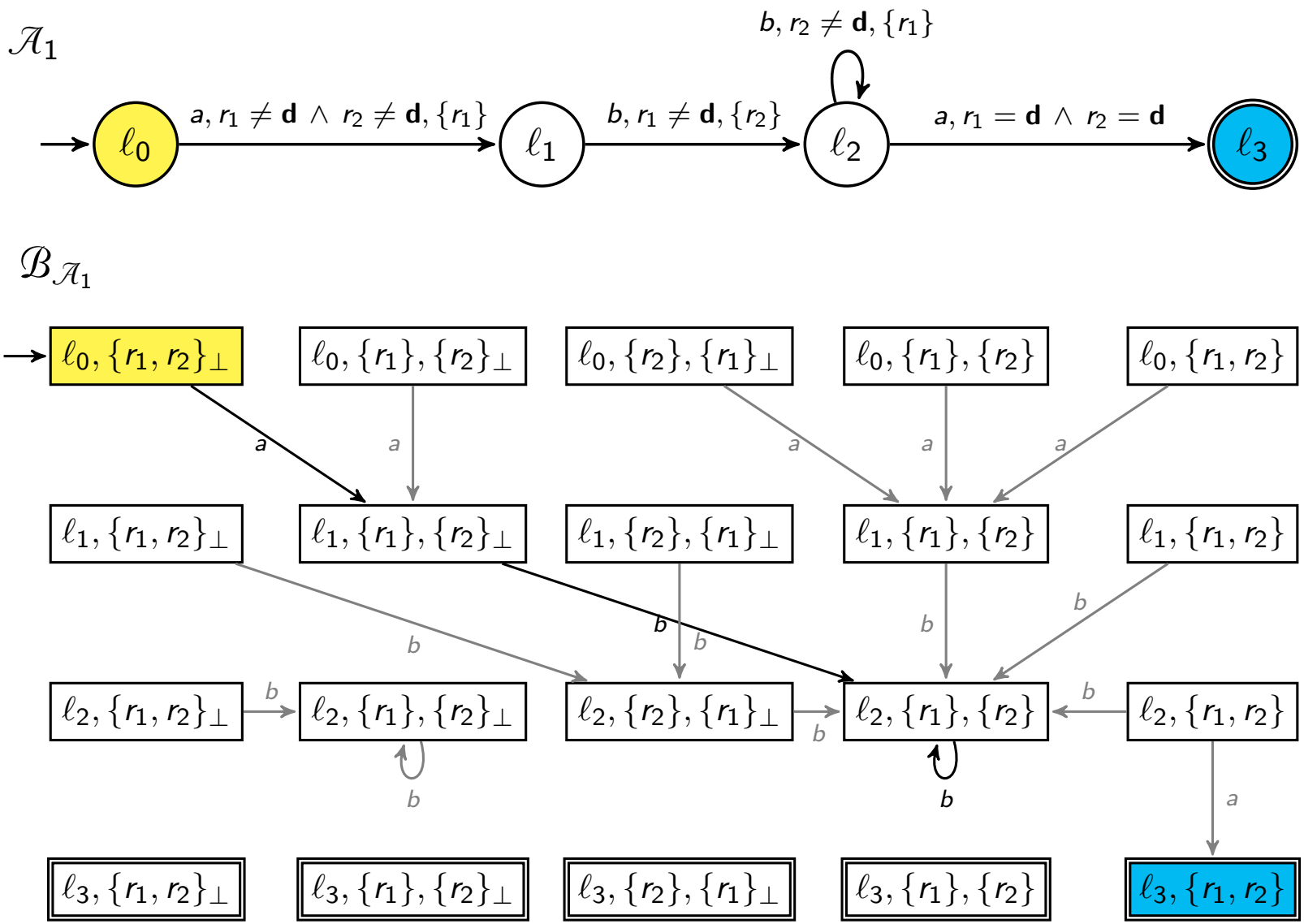
For every register automaton $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L}_{\text{acc}})$ there exists a finite automaton $\mathcal{B}_{\mathcal{A}}$, called the data abstract automaton of $\mathcal{A}$, such that $L(\mathcal{B}_{\mathcal{A}}) \neq \emptyset$ if, and only if, $L(\mathcal{A}) \neq \emptyset$. The number of states in $\mathcal{B}_{\mathcal{A}}$ is bounded exponentially in $|\mathcal{R}|$.

Proof (Construction only).

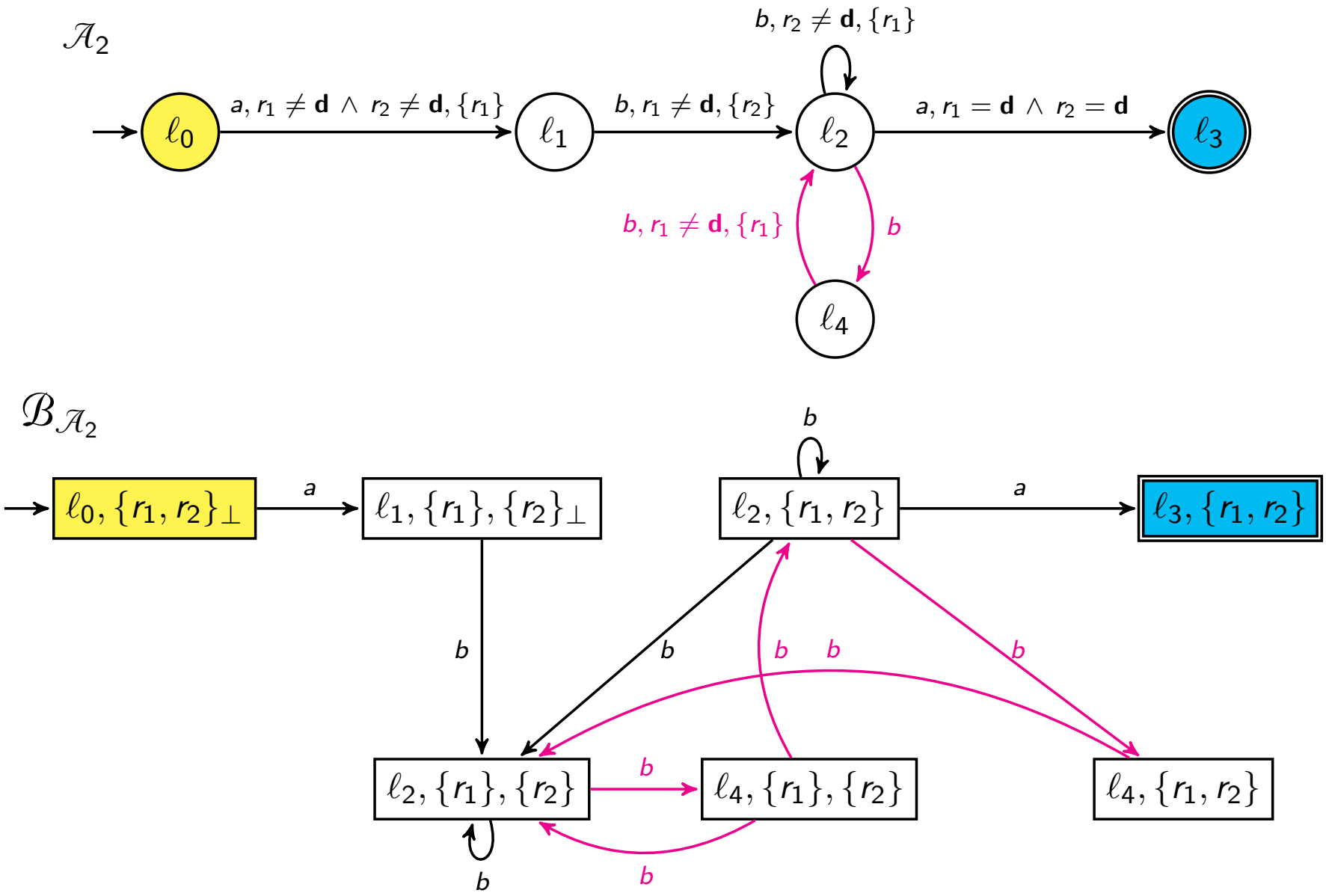
Define $\mathcal{B}_{\mathcal{A}} = (Q, \Sigma, Q_{\text{in}}, \delta, Q_{\text{acc}})$ by

- $Q = \mathcal{L} \times \mathbb{A}(\mathcal{R})$
- $Q_{\text{in}} = \mathcal{L}_{\text{in}} \times \{\mathcal{R}_\perp\}$
- $Q_{\text{acc}} = \mathcal{L}_{\text{acc}} \times \mathbb{A}(\mathcal{R})$
- $((\ell, A), a, (\ell', A')) \in \delta \iff (\ell, A) \xrightarrow{a} (\ell', A')$ as defined before

Data Abstract Automaton of $\mathcal{A}_1$



Data Abstract Automaton of $\mathcal{A}_2$ (reachable states only)



Correctness of the Construction

The construction is correct if it is sound and complete.

Lemma 11 (Completeness)

Let ν_0 be a register valuation and let A_0 be an abstract register valuation such that $\nu_0 \in A_0$. For every run ρ of the form

$$(\ell_0, \nu_0) \xrightarrow{(a_1,d_1)} (\ell_2, \nu_2) \xrightarrow{(a_2,d_2)} \dots \xrightarrow{(a_k,d_k)} (\ell_k, \nu_k)$$

of $\mathcal{A}$ there exist abstract register valuations $A_1, \dots, A_k$ such that

$$(\ell_0, A_0) \xrightarrow{a_1} (\ell_1, A_1) \xrightarrow{a_2} \dots \xrightarrow{a_k} (\ell_k, A_k)$$

is a run of $\mathcal{B}_\mathcal{A}$ and $\nu_i \in A_i$ for all $1 \leq i \leq k$.

Correctness of the Construction II

The construction is correct if it is sound and complete.

Lemma 12 (Soundness)

Let ν_0 be a register valuation and let A_0 be an abstract register valuation such that $\nu_0 \in A_0$. For every run ρ of the form

$$(\ell_0, A_0) \xrightarrow{a_1} (\ell_1, A_1) \xrightarrow{a_2} \dots \xrightarrow{a_k} (\ell_k, A_k)$$

of $\mathcal{B}_{\mathcal{A}}$ there exist register valuations $\nu_0, \nu_1, \dots, \nu_k$ and data values $d_1, \dots, d_k$ such that

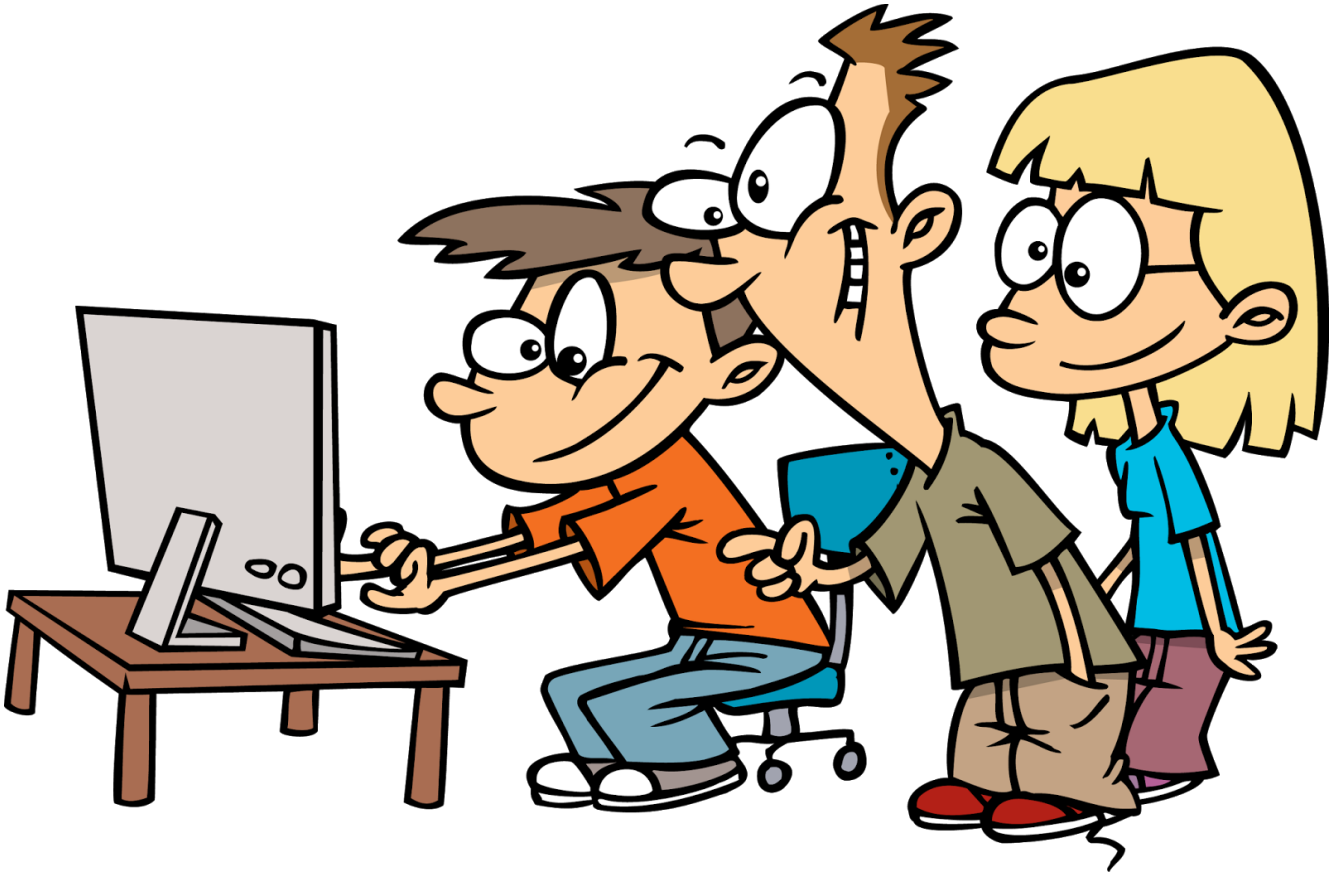
$$(\ell_0, \nu_0) \xrightarrow{(a_1,d_1)} (\ell_1, \nu_1) \xrightarrow{(a_2,d_2)} \dots \xrightarrow{(a_k,d_k)} (\ell_k, \nu_k)$$

is a run of $\mathcal{A}$ such that and $\nu_i \in A_i$ for all $1 \leq i \leq k$.

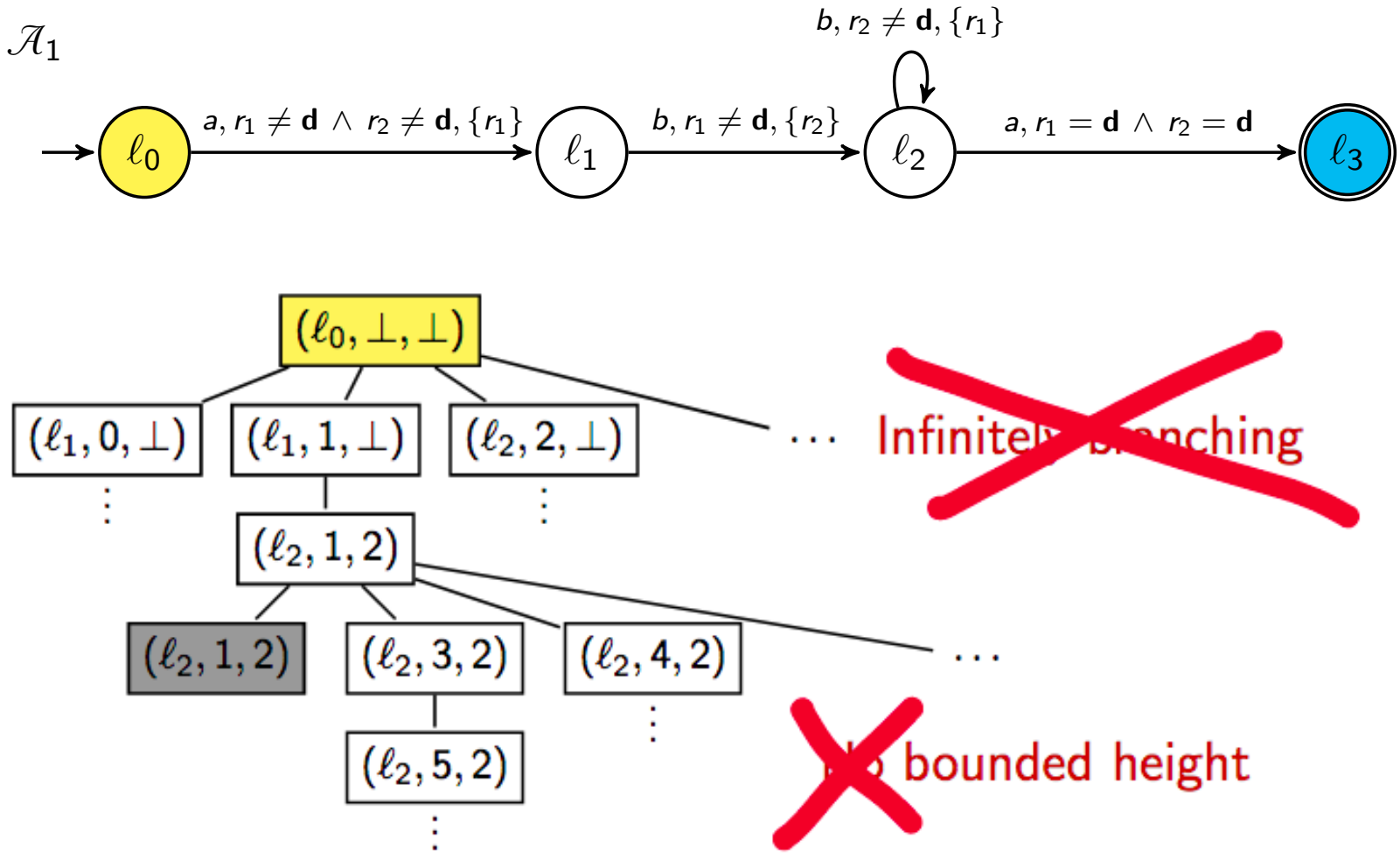
Groupwork

The nonemptiness problem is decidable!

Discuss a potential algorithm for solving the nonemptiness problem. What is the complexity of your algorithm?



Towards Bounding the Height



Towards bounding the height

From the reduction, we can infer a bound on the height of the unfolding tree of $\mathcal{A}$:

- The length of a shortest witness $w \in \Sigma^*$ such that $w \in L(\mathcal{B}_{\mathcal{A}})$ is bounded by the number of states of $\mathcal{B}_{\mathcal{A}}$ (cf. Exercise 1.5)
- The number of states of $\mathcal{B}_{\mathcal{A}}$ is bounded by $|\mathscr{L}| \cdot |\mathbb{A}(\mathscr{R})|$, where $|\mathbb{A}(\mathscr{R})|$ is bounded by $\mathbf{B}(n) + n \cdot \mathbf{B}(n)$, where n is the number of registers of $\mathcal{A}$
- The length of a shortest witness $w \in \Sigma^*$ such that $w \in L(\mathcal{B}_{\mathcal{A}})$ is bounded by $|\mathscr{L}| \cdot (\mathbf{B}(n) + n \cdot \mathbf{B}(n))$, and this is also the bound on the height of the unfolding tree

This also leads us to our first complexity upper bound (next slide)

Complexity - A First Upper Bound

A decision problem P is in the complexity class **EXPTIME** if P can be decided by a deterministic Turing machine in time bounded by $\mathcal{O}(2^{\mathbf{p}(k)})$, where $\mathbf{p}$ is a polynomial function of k, the length of the input

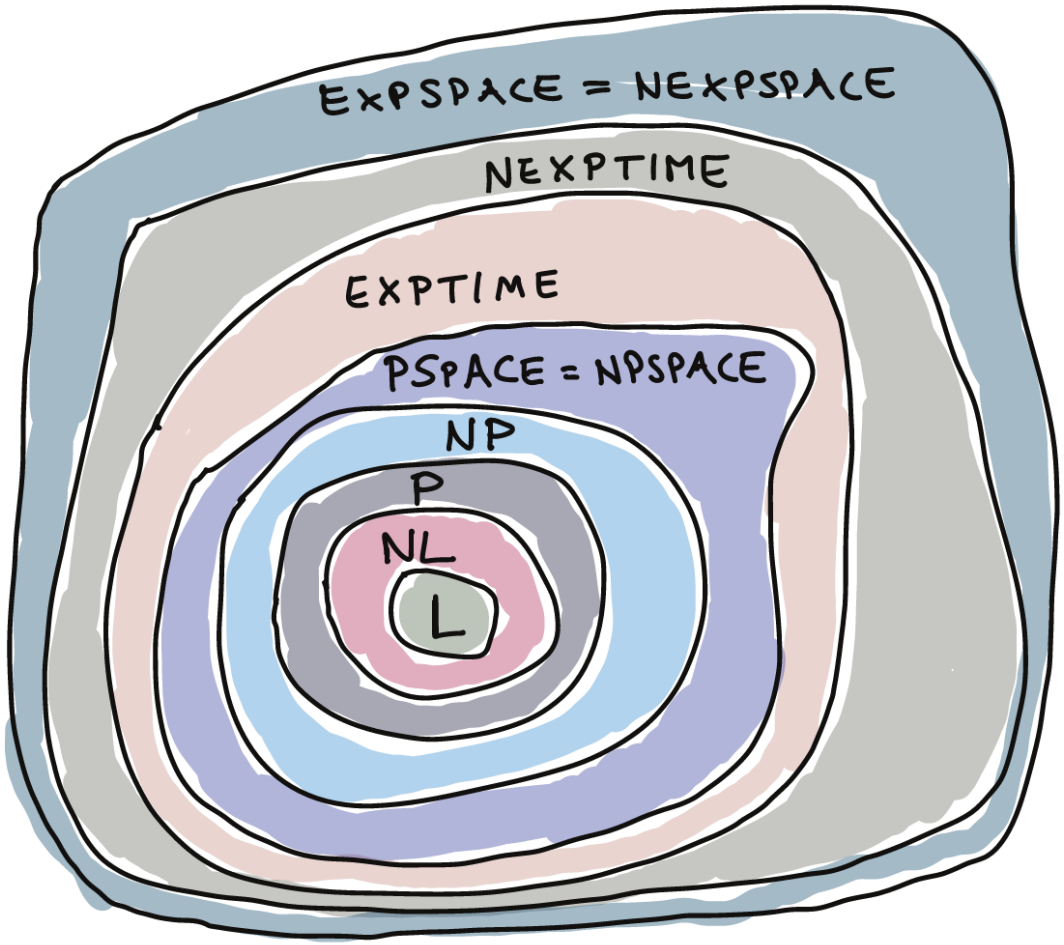
One can construct a deterministic Turing machine that

- computes the complete data abstract automaton $\mathcal{B}_{\mathcal{A}}$ in exponential time (in the size of the input register automaton $\mathcal{A}$), and
- applies the **P**-algorithm for solving the nonemptiness problem on $\mathcal{B}_{\mathcal{A}}$

Hence the nonemptiness problem for register automata is in **EXPTIME**

Can we do better? Or is the nonemptiness problem **EXPTIME**-hard?

Complexity Classes beyond NP



$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NSPACE} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{NEXPTIME} \subseteq \mathbf{EXPSPACE} = \mathbf{NEXPSPACE}$$

The Complexity Classes PSPACE and NSPACE

A decision problem P is in the complexity class **PSPACE** if P can be decided by a deterministic Turing machine in space bounded by $\mathbf{p}(k)$, where $\mathbf{p}$ is a polynomial function of k, the length of the input

A decision problem P is in the complexity class **NSPACE** if P can be decided by a nondeterministic Turing machine in space bounded by $\mathbf{p}(k)$, where $\mathbf{p}$ is a polynomial function of k, the length of the input

Two important results

- **PSPACE** = **NSPACE** (Walter Savitch, 1970)
- **NSPACE** is closed under complement (Neil Immerman, 1988)

We will prove that the nonemptiness problem for register automata is in **PSPACE** (and by Savitch, it suffices to prove that it is in **NSPACE**).

NSPACE Algorithm for Nonemptiness

Let $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L}_{\text{acc}})$ be a register automaton with n registers, and let $\mathcal{B}_{\mathcal{A}} = (Q, \Sigma, Q_{\text{in}}, \delta, Q_{\text{acc}})$ be its data abstract automaton.

The following algorithm returns YES iff $L(\mathcal{A}) \neq \emptyset$.

1: Guess $(\ell_{\text{in}}, A_{\text{in}}) \in Q_{\text{in}}$
2: $(\ell, A) \leftarrow (\ell_{\text{in}}, A_{\text{in}})$
3: **while** $(\ell, A) \notin Q_{\text{acc}}$ **do**
4: Guess a state $(\ell', A') \in Q$ and $a \in \Sigma$ with $((\ell, A), a, (\ell', A')) \in \delta$
5: $(\ell, A) \leftarrow (\ell', A')$
6: **end while**
7: **return** YES

The **NSPACE** Algorithm - Explanations

- The key idea is to construct $\mathcal{B}_{\mathcal{A}}$ on-the-fly
- That is, we do not completely construct $\mathcal{B}_{\mathcal{A}}$ (as it takes too much time), but only construct what is currently needed
- What is currently needed is the current state (ℓ, A) of $\mathcal{B}_{\mathcal{A}}$ that is part of an initialized accepting run of $\mathcal{B}_{\mathcal{A}}$ (we only need polynomial space to store (ℓ, A) on the tape)

Polyomial Space for Storing a State

Every state (ℓ, A) of $\mathcal{B}_{\mathcal{A}}$ can be stored in space polynomial in $|\mathcal{A}|$:

Standard idea:

- Suppose $\mathcal{L} = \{\ell_1, \ldots \ell_k\}$ and $K = \lceil \log_2 k \rceil$. For every $1 \leq i \leq k$, location ℓ_i can be represented by the word $\text{bin}_K(i-1)$
- Suppose $\mathcal{R} = \{r_1, \ldots, r_n\}$ and $N = \lceil \log_2 n \rceil$. For every $1 \leq i \leq n$, register r_i can be represented by the word $\text{bin}_N(i-1)$
- Abstract register valuation A can be represented as finite word over $\{0, 1, \#, \perp\}$
  - Registers that do not have the same data value are separated by $\#$
  - Registers with the same data value are not separated by $\#$
  - Undefined registers (if any) are at the end and followed by $\perp$ symbol
- For example $\{\{r_2\}, \{r_1, r_4\}, \{r_3\}_\perp\}$ can be written as $10\#0011\#01\perp$
- Every state of $\mathcal{B}_{\mathcal{A}}$ can be stored in space polynomial in $k \cdot n$

The **NSPACE** Algorithm - Explanations

- The key idea is to construct $\mathcal{B}_{\mathcal{A}}$ on-the-fly
- That is, we do not completely construct $\mathcal{B}_{\mathcal{A}}$ (as it takes too much time), but only construct what is currently needed
- What is currently needed is the current state (ℓ, A) of $\mathcal{B}_{\mathcal{A}}$ that is part of an initialized accepting run of $\mathcal{B}_{\mathcal{A}}$ (we only need polynomial space to store (ℓ, A) on the tape)
- The algorithm nondeterministically guesses an initial state (ℓ, A)
- Then it nondeterministically guesses another state (ℓ', A') and checks whether $((\ell, A), a, (\ell', A')) \in \delta$ (can be done in **P**)

The **NSPACE** Algorithm - Explanations

- The key idea is to construct $\mathcal{B}_{\mathcal{A}}$ on-the-fly
- That is, we do not completely construct $\mathcal{B}_{\mathcal{A}}$ (as it takes too much time), but only construct what is currently needed
- What is currently needed is the current state (ℓ, A) of $\mathcal{B}_{\mathcal{A}}$ that is part of an initialized accepting run of $\mathcal{B}_{\mathcal{A}}$ (we only need polynomial space to store (ℓ, A) on the tape)
- The algorithm nondeterministically guesses an initial state (ℓ, A)
- Then it nondeterministically guesses another state (ℓ', A') and checks whether $((\ell, A), a, (\ell', A')) \in \delta$ (can be done in **P**)
- If $L(\mathcal{A}) \neq \emptyset$, then there exists an execution of this algorithm that halts with ("YES")

Correctness of the **NSPACE** Algorithm

For the correctness of the algorithm, we need to prove

$$\text{The algorithm answers ``YES'' if, and only if, } L(\mathcal{A}) \neq \emptyset$$

By Theorem 10 $(L(\mathcal{A}) \neq \emptyset) \iff L(\mathcal{B}_{\mathcal{A}}) \neq \emptyset)$, it suffices to prove

$$\text{The algorithm answers ``YES'' if, and only if, } L(\mathcal{B}_{\mathcal{A}}) \neq \emptyset$$

This is left as an exercise for you!

Recall that this is a nondeterministic algorithm, that is, there are two cases:

- all runs do not answer with "YES", or
- at least one run answers with "YES".

Nonemptiness is in **PSPACE**

We have just proved a better upper bound than **EXPTIME**!

Theorem 13

The nonemptiness problem for register automata is in **PSPACE**.

Proof.

We have presented an algorithm that solves the nonemptiness problem in nondeterministic polynomial space, that is, in **NSPACE**. By Savitch's theorem, the problem is in **PSPACE**. □

Can we do better? Or is the nonemptiness problem PSPACE-hard?

$$\mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NSPACE} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{NEXPTIME}$$