Lecture Verification of Infinite-State Systems

Winter 2026/2027

Prof. Dr. Karin Quaas

Institut für Informatik
Universität Leipzig

Lecture 1 - Introduction

Welcome

- Verification of Infinite-State Systems is a course in the field of Theoretical Computer Science (TCS)
- Prerequisites: Standard Bachelor courses in TCS
- M. Sc. Info, Kernmodul Theoretische Informatik Modelle, Modultitel "Verifikation von unendlichen Systemen"
- (Old title of this lecture: Logics and Automata for Data Languages)
- Lecture every Friday 1–3 pm in SG 3-11
- Exercises every Thursday 9–11 am in SG 3-11
- Written Exam at the end of this term (60 Minutes)
- Teaching Language: English or German

How to approach this course?

- Attending the lectures and exercise is not necessary but recommended
- Lecture presentation, exercise solutions and extra proofs are in the respective Moodle course:
  `https://moodle2.uni-leipzig.de/course/view.php?id=62797`
- Please also bring some electronic device if possible (for electronic quizzes and peer instruction)
- Weekly exercises with exercise sheets that I recommend to do before the exercise
- Group works in lectures and exercises
- Bonus point policy:
  - In the exam, you can achieve 60 points
  - You can obtain 6 bonus points during the semester:
  - 6 mini tests (one bonus point each) scattered randomly over the semester
  - So please bring a pen!
- You can always ask questions!

Learning Goals of the Course

You will (can)

- get to know basic formalisms for modelling infinite-state systems (automata, logics)
- get to know and apply basic techniques for handling infinite-state systems
- recall and deepen knowledge about finite automata and (temporal) logics
- practise elementary complexity classes beyond P and NP
- practise how to do reductions from one decision problem to another
- learn about non-elementary complexity classes (fast-growing hierarchy)

Motivation

How can computer scientists guarantee that computer programs are correct?



A simple software bug (integer overflow) lead in 1996 to the explosion of the rocket Ariane 5, forty seconds after its launch. Costs of this failure: around $370 million...

List of famous software bugs:
`https://en.wikipedia.org/wiki/List_of_software_bugs`

What is Verification?

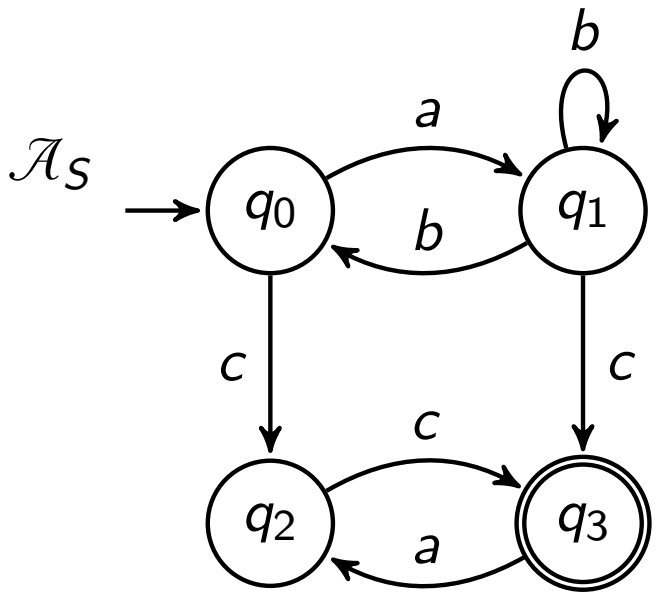
- We aim to automatically prove that a system always behaves as intended
- Verification: given a system S and a specification φ, determine whether

$$S \models \varphi$$

- Examples for specifications:
  - "The system never reaches an error state."
  - "Every request is eventually answered."
  - "A process never enters a critical section while another process is inside."
- Verification is not the same as testing!
  - Testing: does S behave correctly on these executions?
  - Verification: does S behave correctly on all executions?

Why and How can Verification be Automatic?

Suppose the system S is modelled as finite automaton, e.g.



and the specification φ is

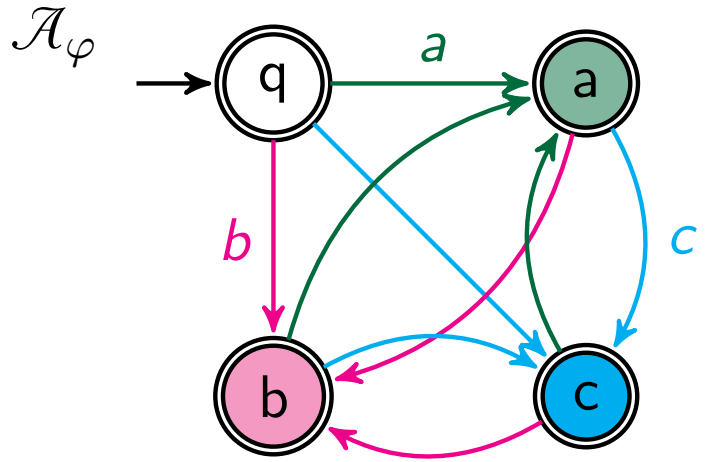
"In every execution of S no two consecutive letters are equal."

Why and How can Verification be Automatic?

- The specification φ

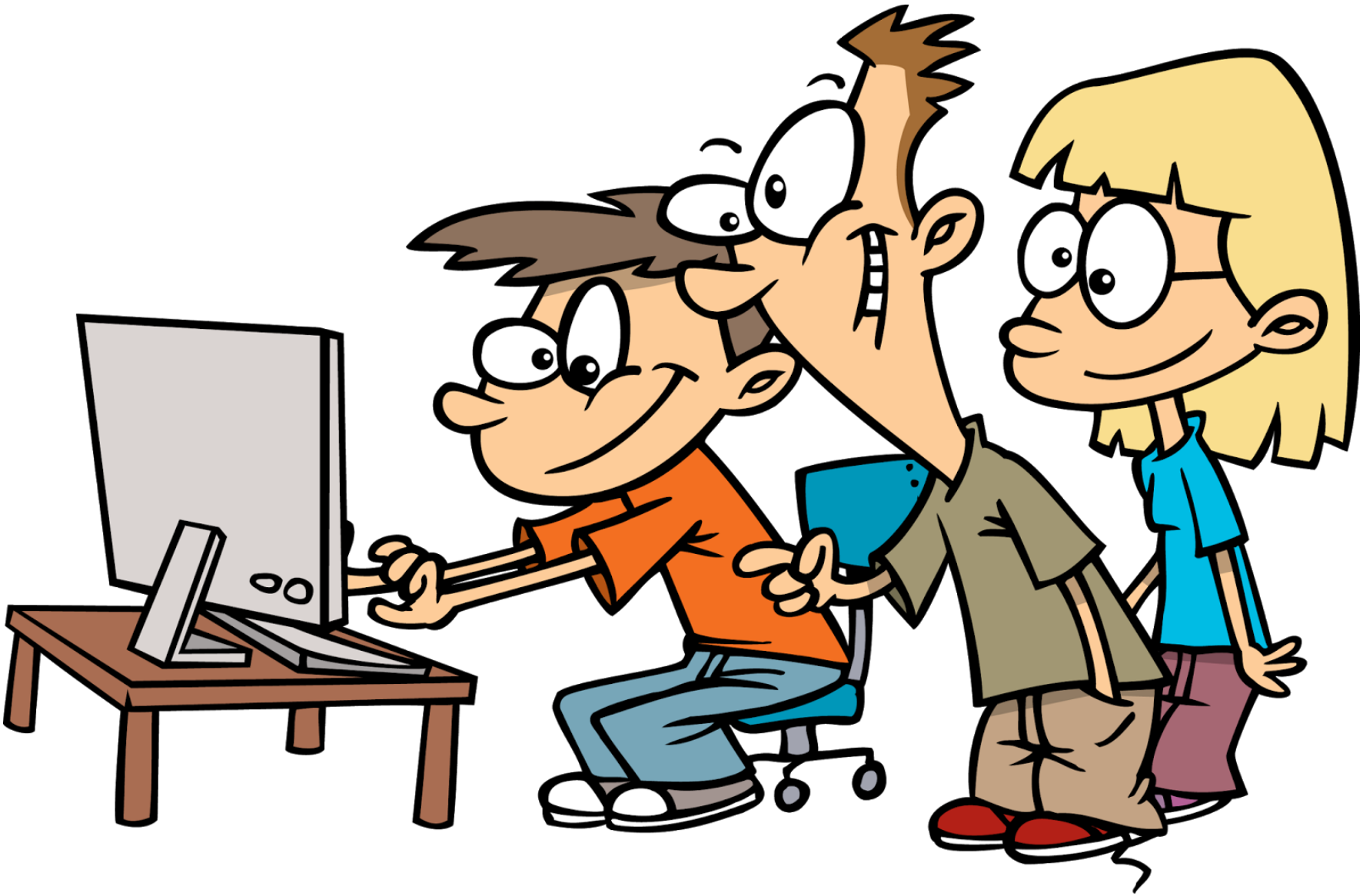
  "In every execution of S no two consecutive letters are equal."

  can easily be modelled as a finite automaton

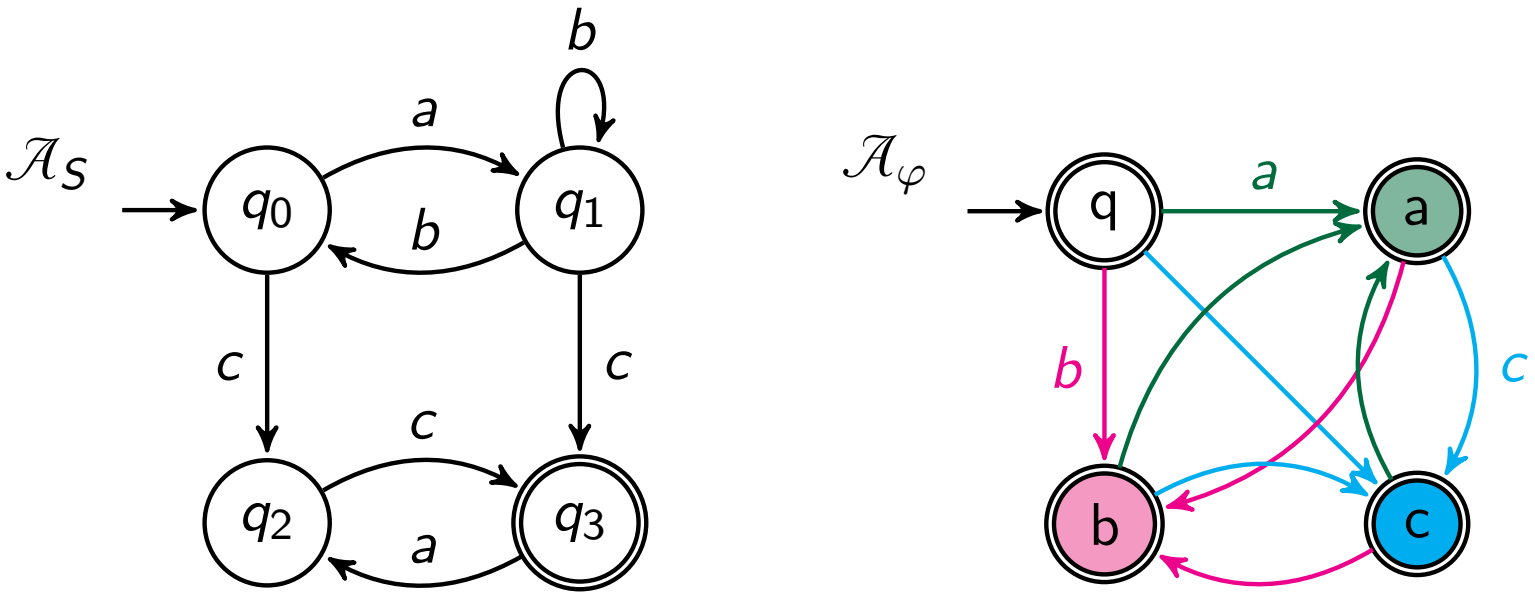


- $\mathcal{A}_\varphi$ simply needs to remember in its states what was the last letter
- S satisfies the specification iff every word accepted by $\mathcal{A}_S$ is also accepted by $\mathcal{A}_\varphi$

Groupwork



Groupwork



- S satisfies the specification iff every word accepted by $\mathcal{A}_S$ is also accepted by $\mathcal{A}_\varphi$

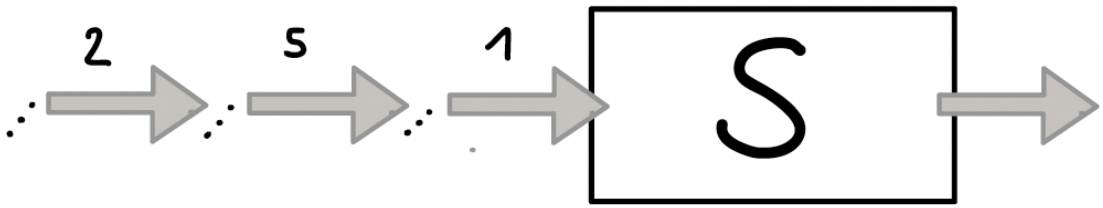
Can you write a program that, for all input automata $\mathcal{A}_S$ and $\mathcal{A}_\varphi$, decides whether every word accepted by $\mathcal{A}_S$ is also accepted by $\mathcal{A}_\varphi$?

Infinite-State Systems

- Verification of finite-state systems is well-established
- Nice ... but which real-world system can be modelled by finite systems?
- (And specifications may be non-regular, too...)
- Realistic systems use unbounded data structures like counters, stacks, queues, lists, trees ...
- However, the induced state space of such systems is infinite
- In general, reachability problem on infinite graphs is undecidable

Example: Processing Data from an Infinite Data Domain

Suppose the system S processes nonnegative integers (instead of letters a, b, c)



and the specification φ is

"In every execution of S no two consecutive data values are equal."

The Challenge of Data

Consider the input sequence

2 0 0 0 0 0 7

A finite automaton can remember

"I have seen an a"

but how can it remember

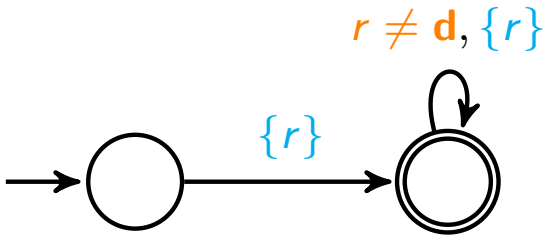
"I have seen the value 2"?

The set of possible data values is $\mathbb{N}$ and hence infinite

We need some finite mechanism for remembering data values.

Register Automata

- Finitely many control states
- Finitely many registers
- Registers may store the data values
- Along transitions one can compare the current data value with stored values
- Registers can be updated with input data value
- Example:



"No two consecutive data values are equal."

Register Automata - Introduction

- Register automata were originally introduced by Kaminski and Francez, 1994 (as finite memory automata)
- It is one of the most studied automaton models for data languages, with active research going on
- Register automata are used to accept data languages
- Register automata extend finite automata with a finite set of registers
- While processing an input data word, register automata can do the following register operations:
  - Store the current input datum into a register
  - Compare the value of a register with the current input datum
- In principle, register automata can be defined over any relational structure
- For simplicity and for a start, we use $(\mathbb{N}; =)$

Data Languages

Given a finite alphabet Σ

- a finite data word over Σ is a finite sequence

$$(a_1, d_1)(a_2, d_2)\dots(a_n, d_n)$$

  such that $a_i \in \Sigma$ and $d_i \in \mathbb{N}$ for all $1 \leq i \leq n$,
- $(\Sigma \times \mathbb{N})^*$ denotes the set of all finite data words over Σ,
- $\varepsilon \in (\Sigma \times \mathbb{N})^*$ denotes the empty word, and
- every subset $L \subseteq (\Sigma \times \mathbb{N})^*$ is a data language over Σ and $\mathbb{N}$.

Register Tests

Let $\mathcal{R} = \{r_1, \ldots, r_n\}$ be a finite set of registers.

$\mathcal{P}(\mathcal{R})$ denotes the powerset of $\mathcal{R}$, that is, the set of all subsets of $\mathcal{R}$

Register tests over $\mathcal{R}$ are defined by the following grammar:

$$\phi ::= \texttt{true} \mid r = \mathbf{d} \mid r \neq \mathbf{d} \mid \phi \wedge \phi$$

where $r \in \mathcal{R}$ and **d** represents the current input datum

$\Phi(\mathcal{R})$ ... the set of all register tests over $\mathcal{R}$

Example 1

- $r_1 = \mathbf{d} \wedge r_2 \neq \mathbf{d}$
- $r_2 = \mathbf{d} \wedge r_2 \neq \mathbf{d}$

Register Automata: Syntax

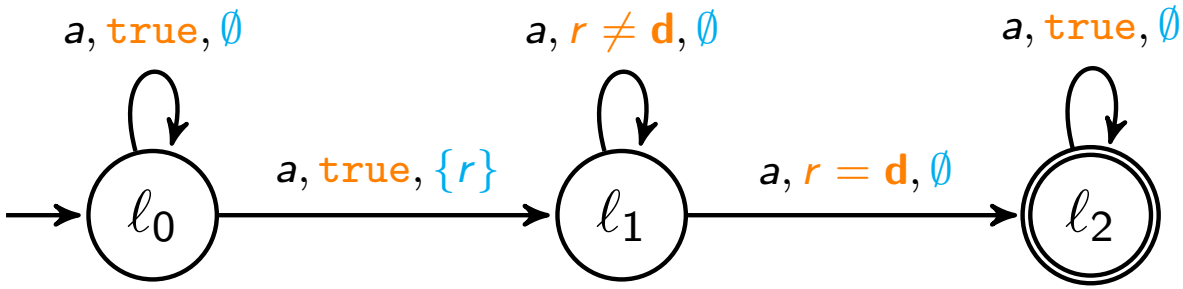
Let Σ be a finite alphabet, let $\mathscr{R}$ be a finite set of registers.

A register automaton over Σ and $\mathscr{R}$ is a tuple $(\mathscr{L}, \Sigma, \mathscr{R}, \mathscr{L}_{\text{in}}, E, \mathscr{L}_{\text{acc}})$, where

- $\mathscr{L}$ is a finite set of locations,
- $\mathscr{L}_{\text{in}} \subseteq \mathscr{L}$ is the set of initial locations,
- $\mathscr{L}_{\text{acc}} \subseteq \mathscr{L}$ is the set of accepting locations,
- $E \subseteq (\mathscr{L} \times \Sigma \times \Phi(\mathscr{R}) \times \mathscr{P}(\mathscr{R}) \times \mathscr{L})$ is a finite set of edges.

An edge of $\mathscr{A}$ is of the form $(\ell, a, \phi, \lambda, \ell')$, where $\ell \in \mathscr{L}$ is the source location, $a \in \Sigma$ is the letter, $\phi \in \Phi(\mathscr{R})$ is a register test, $\lambda \subseteq \mathscr{R}$ is the set of registers to be updated with the current input datum, and $\ell' \in \mathscr{L}$ is the target location.

Example - Register Automata



This is a register automaton $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L}_{\text{acc}})$, where

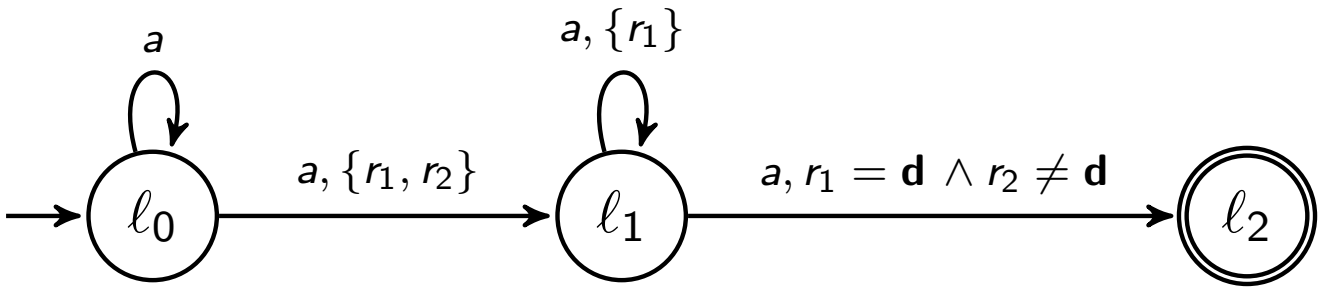
- $\mathcal{L} = \{\ell_0, \ell_1, \ell_2\}$
- $\Sigma = \{a\}$
- $\mathcal{R} = \{r\}$
- $\mathcal{L}_{\text{in}} = \{\ell_0\}$
- $\mathcal{L}_{\text{acc}} = \{\ell_2\}$

- and E consists of the following edges
  - $(\ell_0, a, \texttt{true}, \emptyset, \ell_0)$
  - $(\ell_0, a, \texttt{true}, \{r\}, \ell_1)$
  - $(\ell_1, a, r \neq \mathbf{d}, \emptyset, \ell_1)$
  - $(\ell_1, a, r = \mathbf{d}, \emptyset, \ell_2)$
  - $(\ell_2, a, \texttt{true}, \emptyset, \ell_2)$

We often omit register test $\texttt{true}$ and empty register updates $\emptyset$

We may also write $\downarrow r$ instead of $\{r\}$ to denote that r is to be updated.

Example 2 - Register Automata



This is a register automaton $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L}_{\text{acc}})$, where

- $\mathcal{L} = \{\ell_0, \ell_1, \ell_2\}$
- $\Sigma = \{a\}$
- $\mathcal{R} = \{r_1, r_2\}$
- $\mathcal{L}_{\text{in}} = \{\ell_0\}$
- $\mathcal{L}_{\text{acc}} = \{\ell_2\}$

- and E consists of the following edges
  - $(\ell_0, a, \texttt{true}, \emptyset, \ell_0)$
  - $(\ell_0, a, \texttt{true}, \{r_1, r_2\}, \ell_1)$
  - $(\ell_1, a, \texttt{true}, \{r_1\}, \ell_1)$
  - $(\ell_1, a, r_1 = \mathbf{d} \wedge r_2 \neq \mathbf{d}, \emptyset, \ell_2)$

Register Valuations

Register valuations are used to assign to each register a value in our data domain $\mathbb{N}$

Initially, registers are undefined, and we use $\perp$ to represent this

To formalize this,

- let $\mathbb{N}_\perp$ denote $\mathbb{N} \cup \{\perp\}$
- set $\perp = \perp$, $\perp \neq d$ and $d \neq \perp$ for every $d \in \mathbb{N}$

A register valuation over $\mathscr{R}$ is a mapping $\nu : \mathscr{R} \to \mathbb{N}_\perp$.

We use $\mathbb{N}_\perp^{\mathscr{R}}$ to denote the set of all register valuations over $\mathscr{R}$

We often write register valuations as $|\mathscr{R}|$-tuples of values in $\mathbb{N}_\perp$, e.g., $(0, 1, 8, 0, \perp)$ for $\nu(r_1) = \nu(r_4) = 0$, $\nu(r_2) = 1$, $\nu(r_3) = 8$ and $\nu(r_5) = \perp$

Satisfaction Relation for Register Tests

We define a satisfaction relation, denoted by $\models$, for $\Phi(\mathcal{R})$ on $\mathbb{N}_\perp^{\mathcal{R}} \times \mathbb{N}$, by structural induction on the definition of register tests.

Let $\nu \in \mathbb{N}_\perp^{\mathcal{R}}$ be a register valuation, and let $d \in \mathbb{N}$ be a datum. Then

$$
\begin{aligned}
\nu, d &\models \texttt{true} \\
\nu, d &\models r = \mathbf{d} \iff \nu(r) = d \\
\nu, d &\models r \neq \mathbf{d} \iff \nu(r) \neq d \\
\nu, d &\models \phi \wedge \phi' \iff \nu, d \models \phi \text{ and } \nu, d \models \phi'
\end{aligned}
$$

If $\nu, d \models \phi$, we say ν and d satisfy ϕ

We say that a register test $\phi \in \Phi(\mathcal{R})$ is satisfiable if there exists a register valuation $\nu : \mathcal{R} \to \mathbb{N}_\perp$ and $d \in \mathbb{N}$ that satisfy ϕ

Satisfaction Relation for Register Tests - Examples

Example 2

Let $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$, $\nu = (3, 4, 4, 1)$ and $d = 3$. Then

- $\nu, d \models r_1 = \mathbf{d}$;
- $\nu, d \models r_2 \neq \mathbf{d}$;
- $\nu, d \not\models r_1 \neq \mathbf{d} \wedge r_1 = \mathbf{d}$ (this register test is unsatisfiable)
- $\nu, d \models r_1 = \mathbf{d} \wedge r_3 \neq \mathbf{d}$
- $\nu, d \not\models r_1 \neq \mathbf{d}$;

Satisfiability of Register Test

Satisfiability Problem for Register Tests	
Input:	A set $\mathcal{R} = \{r_1, \ldots, r_n\}$ and a register test $\phi \in \Phi(\mathcal{R})$
Question:	Is ϕ satisfiable?

- This is a decision problem (yes or no)
- Is this problem decidable?
- What is the computational complexity?

Theorem 1

The satisfiability problem for register tests is in **P**.

Exercise 1.4.

Two Special Register Valuations

The initial register valuation is $\nu_\perp$ defined by $\nu_\perp(r) = \perp$ for all $r \in \mathcal{R}$

Given a register valuation ν, a datum $d \in \mathbb{N}$, and a set $\lambda \subseteq \mathcal{R}$, we define $\nu[\lambda \leftarrow d]$ to be the register valuation defined by

$$\nu[\lambda \leftarrow d](r) = \begin{cases} d & \text{if } r \in \lambda \\ \nu(r) & \text{otherwise} \end{cases}$$

Example 3

Let $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$. Then we have
$(\nu_\perp[\{r_1, r_4\} \leftarrow 5])[\{r_2, r_4\} \leftarrow 3] = (5, 3, \perp, 3)$

Pingo Quiz

For which d does $\nu, d \models \phi$ hold if

- $\nu = (\nu_\perp[\{r_2, r_3\} \leftarrow 5])[\{r_1, r_2\} \leftarrow 1]$, and
- ϕ is of the form $r_4 \neq \mathbf{d} \wedge r_2 \neq \mathbf{d}$.

A $d = 1$

B $d = 5$

C none of them

D all of them

Register Automata: Semantics

Let $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L}_{\text{acc}})$.

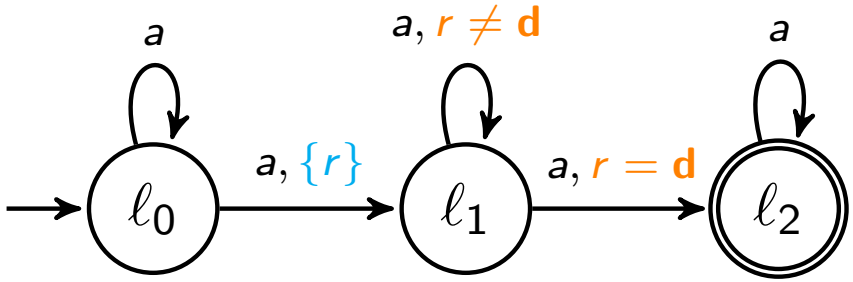
A state of $\mathcal{A}$ is a pair (ℓ, ν), where $\ell \in \mathcal{L}$ and $\nu \in \mathbb{N}_\perp^{\mathcal{R}}$ is a register valuation.

Define the transition relation $\longrightarrow \subseteq ((\mathcal{L} \times \mathbb{N}_\perp^{\mathcal{R}}) \times (\Sigma \times \mathbb{N}) \times (\mathcal{L} \times \mathbb{N}_\perp^{\mathcal{R}}))$ by

$$\langle (\ell, \nu), (a, d), (\ell', \nu') \rangle \in \longrightarrow$$
$$:\Longleftrightarrow$$
there exists some edge $(\ell, a, \phi, \lambda, \ell') \in E$
such that $\nu, d \models \phi$ and $\nu' = \nu[\lambda \leftarrow d]$

We may write $(\ell, \nu) \xrightarrow{(a,d)} (\ell', \nu')$ instead of $\langle (\ell, \nu), (a, d), (\ell', \nu') \rangle \in \longrightarrow$

Transition Relation - Example



- $(\ell_0, \bot) \xrightarrow{(a,2)} (\ell_0, \bot)$ as there exists edge $(\ell_0, a, \texttt{true}, \emptyset, \ell_0) \in E$ and $\bot, 2 \models \texttt{true}$ and $\bot = \bot[\emptyset \leftarrow 2]$
- $(\ell_0, \bot) \xrightarrow{(a,3)} (\ell_1, 3)$ as there exists edge $(\ell_0, a, \texttt{true}, \{r\}, \ell_1) \in E$ and $\bot, 3 \models \texttt{true}$ and $3 = \bot[\{r\} \leftarrow 3]$
- $(\ell_1, 3) \xrightarrow{(a,8)} (\ell_1, 3)$ as there exists edge $(\ell_1, a, r \neq \mathbf{d}, \emptyset, \ell_1) \in E$ and $3, 8 \models r \neq \mathbf{d}$ and $3 = 3[\emptyset \leftarrow 8]$

Register Automata: Semantics (II)

Let $\mathcal{A} = (\mathcal{L}, \Sigma, \mathcal{R}, \mathcal{L}_{\text{in}}, E, \mathcal{L}_{\text{acc}})$. A run of $\mathcal{A}$ is a finite sequence of transitions

$$(\ell_0, \nu_0) \xrightarrow{(a_1,d_1)} (\ell_1, \nu_1) \xrightarrow{(a_2,d_2)} \dots \xrightarrow{(a_n,d_n)} (\ell_n, \nu_n)$$

We say that this run is a run on the data word $(a_1, d_1)(a_2, d_2) \dots (a_n, d_n)$.

The run is initialized if $\ell_0 \in \mathcal{L}_{\text{in}}$ and $\nu_0 = \nu_\perp$.

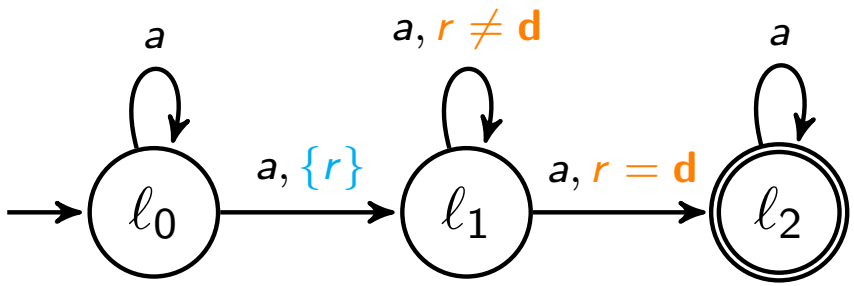
The run is accepting if $\ell_n \in \mathcal{L}_{\text{acc}}$.

The data language accepted by $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of data words $w \in (\Sigma \times \mathbb{N})^*$ such that there exists an initialized, accepting run of $\mathcal{A}$ on w.

A data language $L \subseteq (\Sigma \times \mathbb{N})^*$ is called data regular if there exists some register automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L$.

Example - Register Automata

$\mathcal{A}$



The following

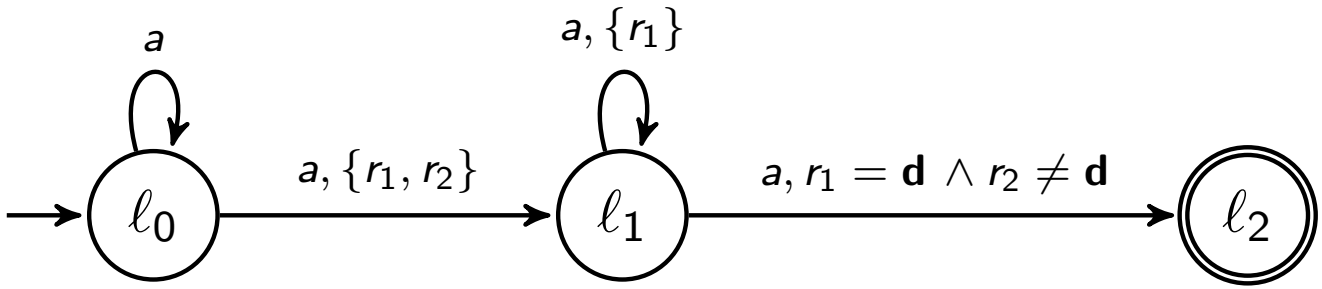
$$(\ell_0, \perp) \xrightarrow{(a,2)} (\ell_0, \perp) \xrightarrow{(a,3)} (\ell_1, 3) \xrightarrow{(a,8)} (\ell_1, 3) \xrightarrow{(a,4)} (\ell_1, 3) \xrightarrow{(a,3)} (\ell_2, 3)$$

is a run on the data word $(a,2)(a,3)(a,8)(a,4)(a,3)$

$$L(\mathcal{A}) = \{(a,d_1)(a,d_2)\dots(a,d_n) \mid \text{there exist } 1 \leq i < j \leq n \text{ such that } d_i = d_j \text{ and } d_k \neq d_i \text{ for all } i < k < j, n \geq 2\}$$

Example 2 - Register Automata

$\mathcal{A}$



The following

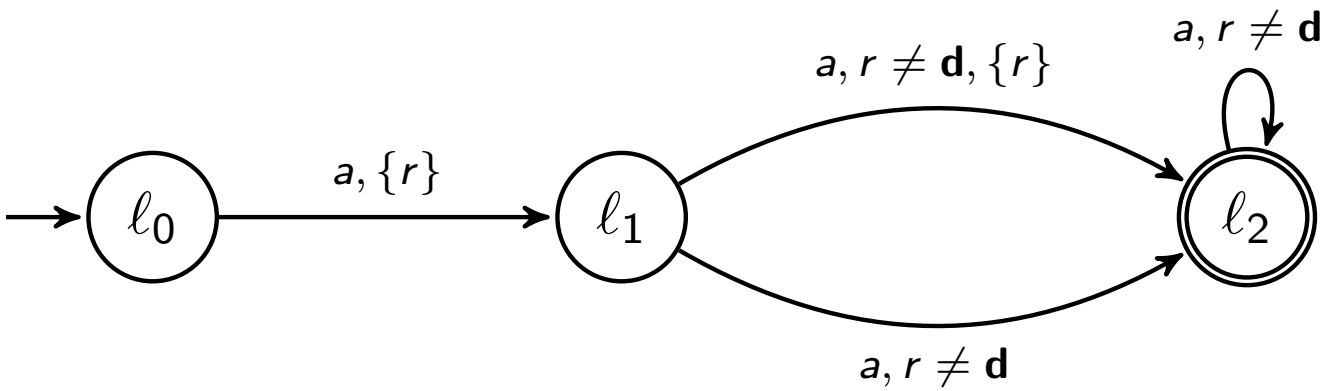
$$(\ell_0, \perp, \perp) \xrightarrow{(a,2)} (\ell_0, \perp, \perp) \xrightarrow{(a,3)} (\ell_1, 3, 3) \xrightarrow{(a,2)} (\ell_1, 2, 3) \xrightarrow{(a,4)} (\ell_1, 4, 3) \xrightarrow{(a,4)}$$

is a run on the data word $(a, 2)(a, 3)(a, 2)(a, 4)(a, 4)$

$L(\mathcal{A}) = \{(a_1, d_1) \dots (a_n, d_n) \mid$ there exist $1 \leq i < n - 1$ such that $d_i \neq d_n$ and $d_{n-1} = d_n, n \geq 3\}$

Quiz

$\mathcal{A}$



Which of the following data words is *not* accepted by $\mathcal{A}$?

A $(a, 5)(a, 8)(a, 5)$

B $(a, 5)(a, 8)(a, 8)(a, 8)$

C $(a, 5)(a, 8)(a, 5)(a, 8)$

D $(a, 5)(a, 8)(a, 7)(a, 8)$

E All of them are accepted by $\mathcal{A}$