Complexity over arbitrary structures

Komplexitätsbetrachtungen über beliebigen Strukturen

Christine Gaßner
Greifswald

Bonn
November 15, 2010

Complexity over arbitrary structures

1. The uniform model of computation
2. Diagonalization techniques and halting problems
3. Structures with $P \neq NP$
4. Structures and oracles with $P^A \neq NP^A$ and $P^A = NP^A$
5. An idea for the construction of structures with $P = NP$

gassnerc@uni-greifswald.de

The uniform BSS model of computation

Example constants operations relation

A ring: $\mathbb{R} = (\mathbb{R}\,;\, \overbrace{0, 1}^{\text{constants}}\,;\, \overbrace{+, -, \cdot}^{\text{operations}}\,;\, \overbrace{\leq}^{\text{relation}})$

$\mathbb{R} = (\mathbb{R}\,;\, \mathbb{R}\,;\, +, -, \cdot\,;\, \leq)$ (⇨ BSS model)

infinite signature

A computation:

$s := 0;$

for $i := 1$ to n do

(index) integer

any arity

$\{$

$s := s + x_i\,;$

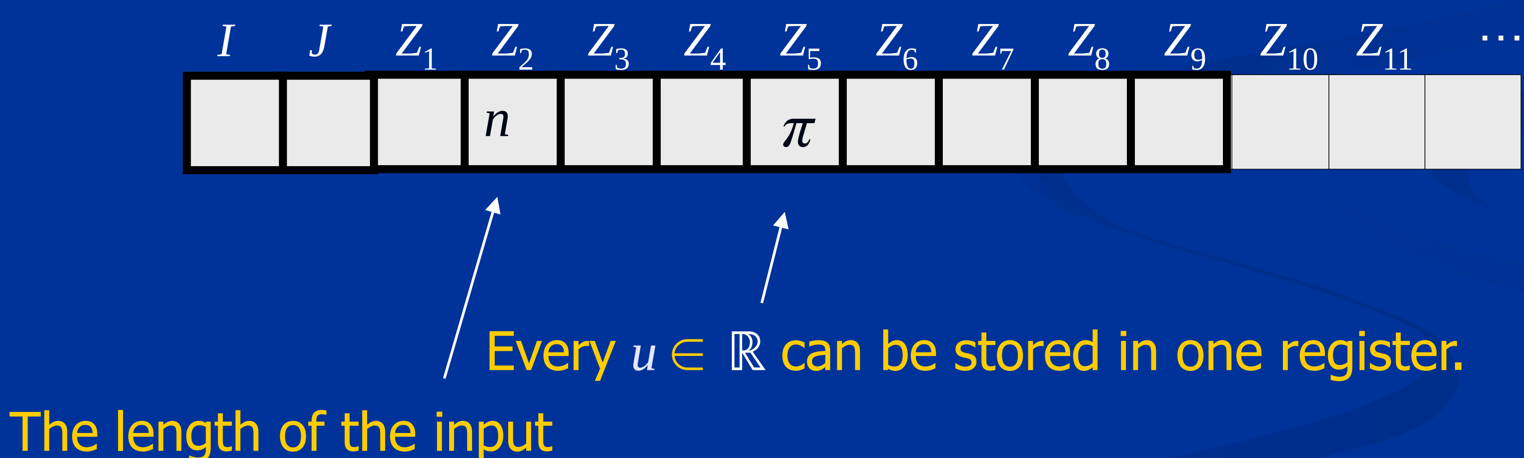
$\}$

real number (element of the structure)

The uniform BSS model of computation

Registers for elements of $\mathbb{R}$: $Z_1, Z_2, Z_3, \ldots$

Registers for indices: I, J

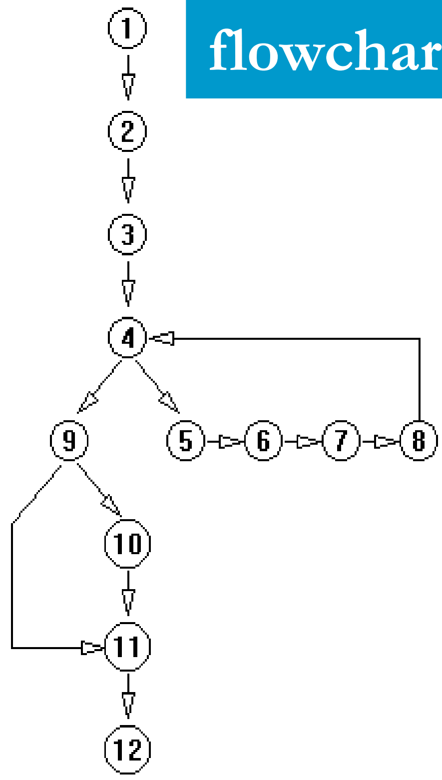


Representations of programs

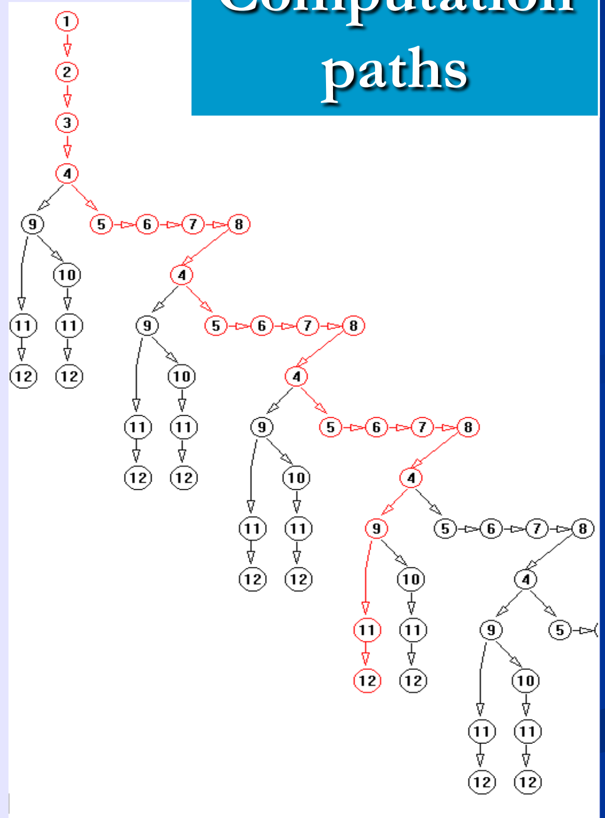
A program

1: Input: $(x_1, \ldots, x_n)$.
Guess: $(y_1, \ldots, y_n) \in \{0, 1\}$.
2: $I_2 := I_1 + 1$;
3: $Z_1 := 1 + Z_1 * Z_{I_2}$;
4: if $I_1 = I_3$ then goto 9
else goto 5;
5: $I_2 := I_2 + 1$;
6: $I_3 := I_3 + 1$;
7: $Z_1 := Z_1 + Z_{I_3} * Z_{I_2}$;
8: goto 4;
9: if $Z_1 = 1$ then goto 11
else goto 10;
10: $Z_1 := 0$;
11: $I_1 := 1$;
12: Output: Z_1.

A flowchart



Computation paths



gassnerc@uni-greifswald.de

The uniform model of computation

A structure: $\Sigma = (U\,;\, c_1,\ldots,c_u\,;\; f_1,\ldots,f_v\,;\; R_1,\ldots,R_w, =)$

$\Sigma = (U\,;\,(c_i)_{i\in F}\,;\,(f_i)_{i\in G}\,;\,(R_i)_{i\in H}\,)$

Computation: $l\!:\; Z_k := f_j(Z_{k_1},\ldots,Z_{k_{m_j}});$

$l\!:\; Z_k := c_j;$

Branching: $l\!:$ if $R_j(Z_{k_1},\ldots,Z_{k_{n_j}})$ then goto l_1 else goto l_2;

$l\!:$ if $Z_k = Z_j$ then goto l_1 else goto l_2;

Copy: $l\!:\; Z_{I_k} := Z_{I_l};$

Index computation: $I_k := 1$; $I_k := I_k + 1$; if $I_k = I_j$ then goto l_1 else goto l_2;

gassnerc@uni-greifswald.de

The uniform model of computation

A structure: $\Sigma = (U;\, c_1,\ldots, c_u;\; f_1,\ldots, f_v;\; R_1,\ldots, R_w, =)$

$\Sigma = (U;\, (c_i)_{i\in F};\, (f_i)_{i\in G};\, (R_i)_{i\in H})$

Computation: $l\colon\; Z_k := f_j(Z_{k_1},\ldots, Z_{k_{m_j}});$

$l\colon\; Z_k := c_j;$

Branching: $l\colon$ if $R_j(Z_{k_1},\ldots, Z_{k_{n_j}})$ then goto l_1 else goto l_2;

$l\colon$ if $Z_k = Z_j$ then goto l_1 else goto l_2;

Copy: $l\colon\; Z_{I_k} := Z_{I_j};$

Index computation: $I_k := 1;\;\; I_k := I_k + 1;\;\;$ if $I_k = I_j$ then goto l_1 else goto l_2;

The uniform model of computation

A structure: $\Sigma = (U\,;\, c_1,\ldots,c_u\,;\; f_1,\ldots,f_v\,;\; R_1,\ldots,R_w, =)$

$\Sigma = (U\,;\,(c_i)_{i\in F}\,;\,(f_i)_{i\in G}\,;\,(R_i)_{i\in H}\,)$

Computation:

$l\!:\; Z_k := f_j(Z_{k_1},\ldots,Z_{k_{m_j}});$

$l\!:\; Z_k := c_j;$

Branching:

$l\!:$ if $R_j(Z_{k_1},\ldots,Z_{k_{n_j}})$ then goto l_1 else goto l_2;

[$l\!:$ if $Z_k = Z_j$ then goto l_1 else goto l_2;]

Copy: $l\!:\; Z_{I_k} := Z_{I_j};$

Index computation: $I_k := 1;\;\; I_k := I_k + 1;\;\;$ if $I_k = I_j$ then goto l_1 else goto l_2;

The uniform model of computation

A structure: $\Sigma = (U\,;\, c_1,\ldots, c_u\,;\ f_1,\ldots, f_v\,;\ R_1,\ldots, R_w, =)$

$\Sigma = (U\,;\, (c_i)_{i\in F}\,;\, (f_i)_{i\in G}\,;\, (R_i)_{i\in H}\,)$

Computation: $l\!:\ Z_k := f_j(Z_{k_1},\ldots, Z_{k_{m_j}});$

$l\!:\ Z_k := c_j;$

Branching: $l\!:$ if $R_j(Z_{k_1},\ldots, Z_{k_{n_j}})$ then goto l_1 else goto l_2;

[$l\!:$ if $Z_k = Z_j$ then goto l_1 else goto l_2;]

Copy: $l\!:\ Z_{I_k} := Z_{I_j};$

Index computation: $I_k := 1;$ $I_k := I_k + 1;$ if $I_k = I_j$ then goto l_1 else goto l_2;

gassnerc@uni-greifswald.de

The uniform model of computation

A structure: $\Sigma = (U\,;\, c_1,\ldots, c_u\,;\; f_1,\ldots, f_v\,;\; R_1,\ldots, R_w, =)$

$\Sigma = (U\,;\, (c_i)_{i\in F}\,;\, (f_i)_{i\in G}\,;\, (R_i)_{i\in H}\,)$

Computation: $l\!: \; Z_k := f_j(Z_{k_1},\ldots, Z_{k_{m_j}});$

$l\!: \; Z_k := c_j;$

Branching: $l\!: \;$ if $\; R_j(Z_{k_1},\ldots, Z_{k_{n_j}})\;$ then goto l_1 else goto l_2;

[$l\!: \;$ if $\; Z_k = Z_j\;$ then goto l_1 else goto l_2;]

Copy: $l\!: \; Z_{I_k} := Z_{I_j};$

Index computation: $I_k := 1;\;\; I_k := I_k + 1;\;\;$ if $\; I_k = I_j\;$ then goto l_1 else goto l_2;

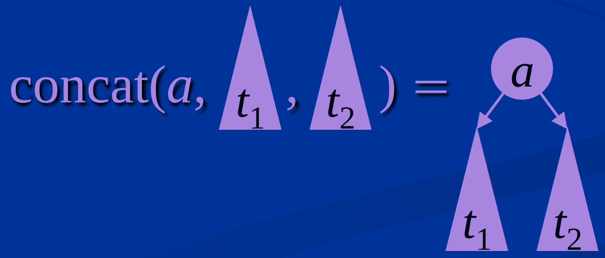
Examples for several structures

$\mathbb{Z}_2 = (\{0, 1\}; 0, 1; +, \cdot\ ; =)$ (⇨ Turing machines)

$\mathbb{R} = (\mathbb{R}; \mathbb{R}; +, -, \cdot\ ; \leq)$ (⇨ BSS model)

$\Sigma_{\text{string}} = (\{0, 1\}^*; \varepsilon, 0, 1; \text{add}, \text{sub}_l, \text{sub}_r; =)$

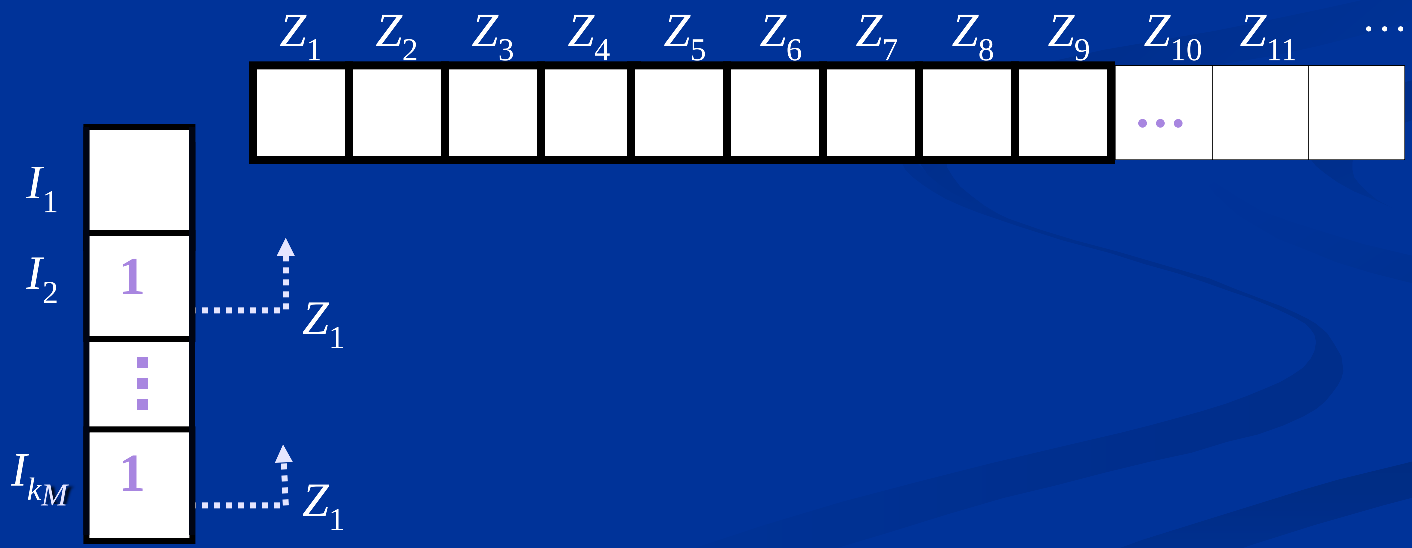
$\Sigma_{\text{tree}} = (\text{tree}(\mathbb{R}); \text{nil}; \text{concat}, \text{root}, \text{sub}_l, \text{sub}_r; =)$



gassnerc@uni-greifswald.de

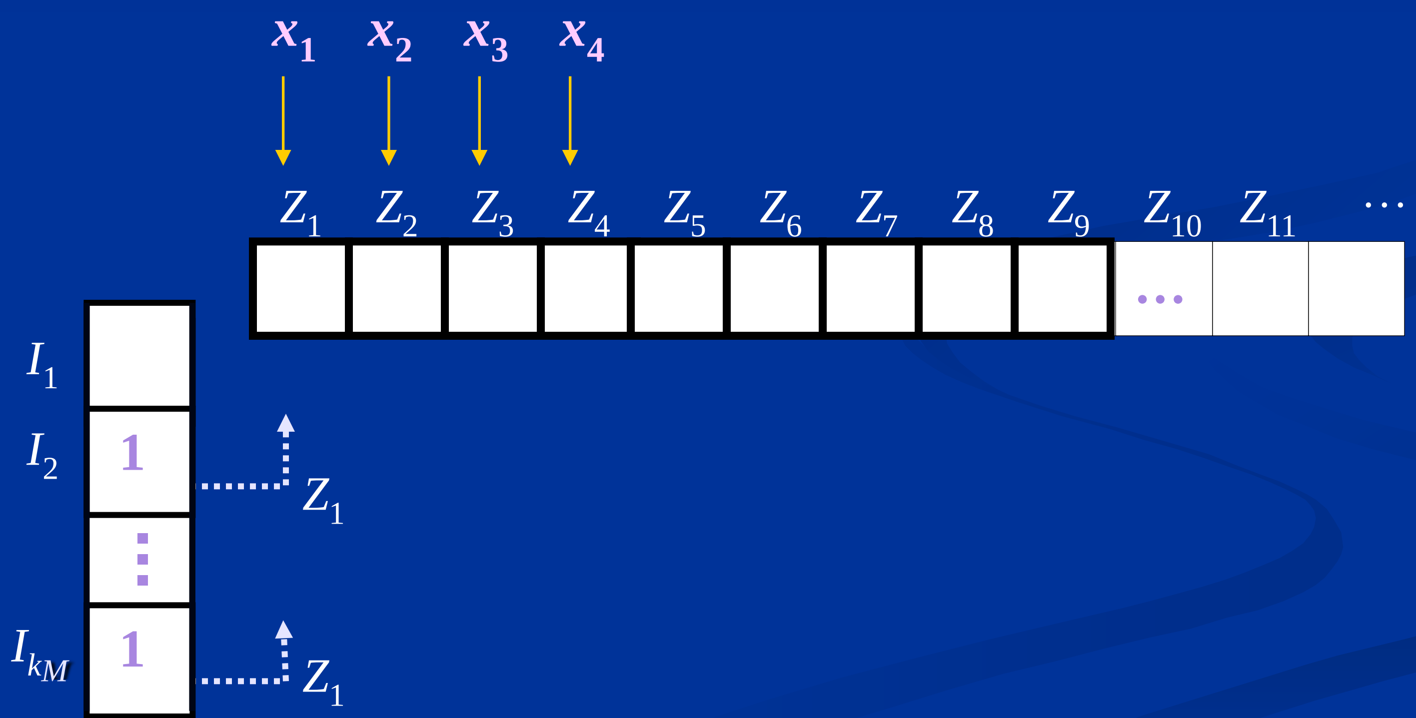
The machine and the input

The input: $(Z_1,\ldots, Z_n) := (x_1,\ldots, x_n);\ I_1 := n;\ \ I_2 := 1;\ldots ;\ I_{k_M} := 1;$



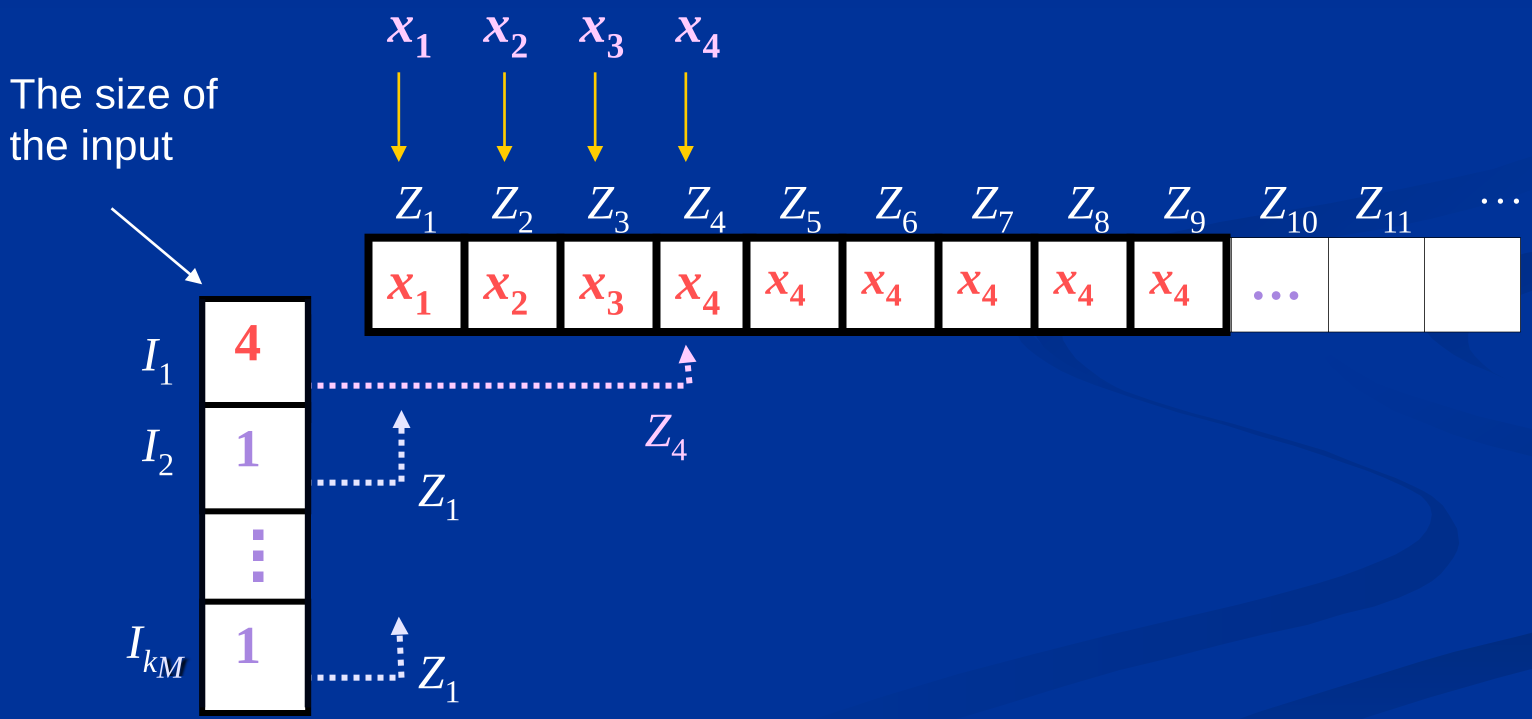
The machine and the input

The input: $(Z_1,\ldots,Z_n) := (x_1,\ldots,x_n);\ I_1 := n;\ \ I_2 := 1;\ldots;\ I_{k_M} := 1;$



The machine and the input

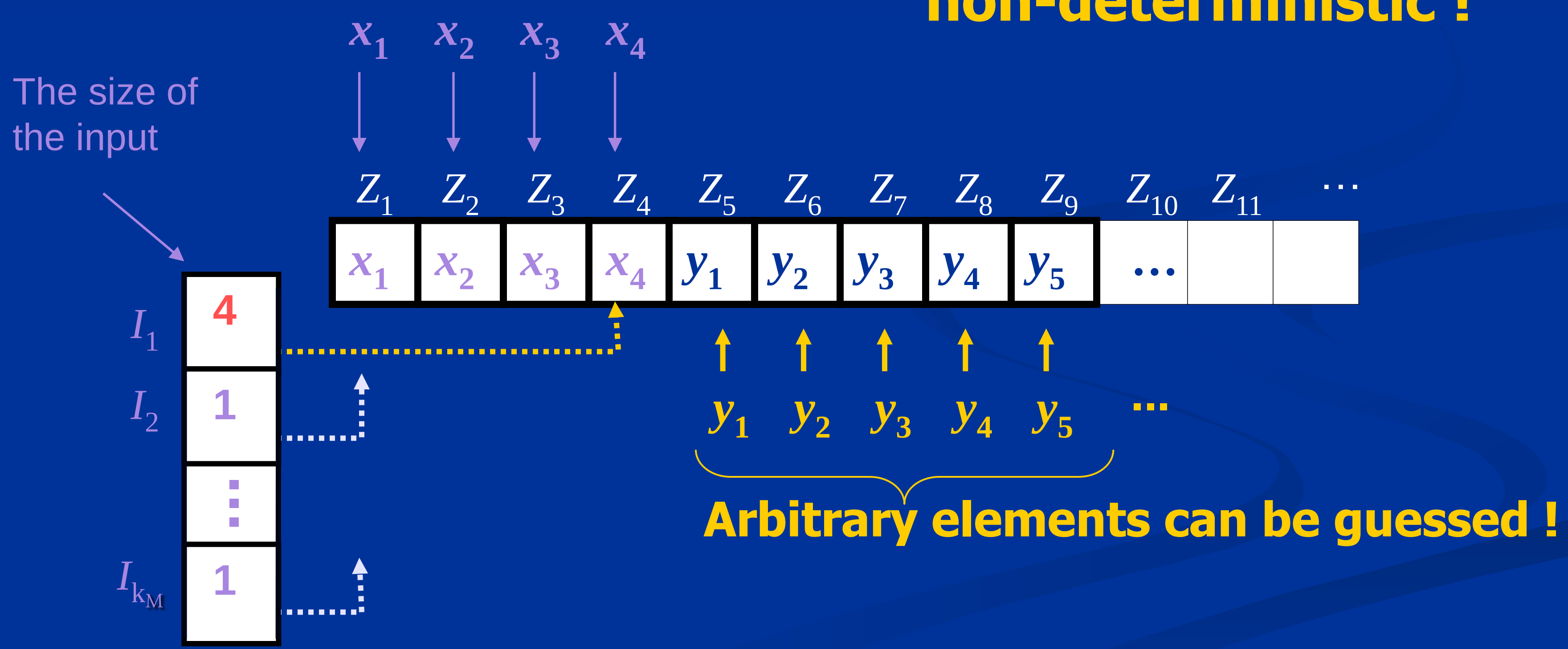
The input: $(Z_1,\ldots,Z_n) := (x_1,\ldots,x_n);\ I_1 := n;\ \ I_2 := 1;\ldots;\ I_{k_M} := 1;$



The non-deterministic machines
(input and guessing)

The guessing: $(Z_{n+1},\ldots,Z_{n+m}) := (y_1,\ldots,y_m) \in U^m$

non-deterministic !

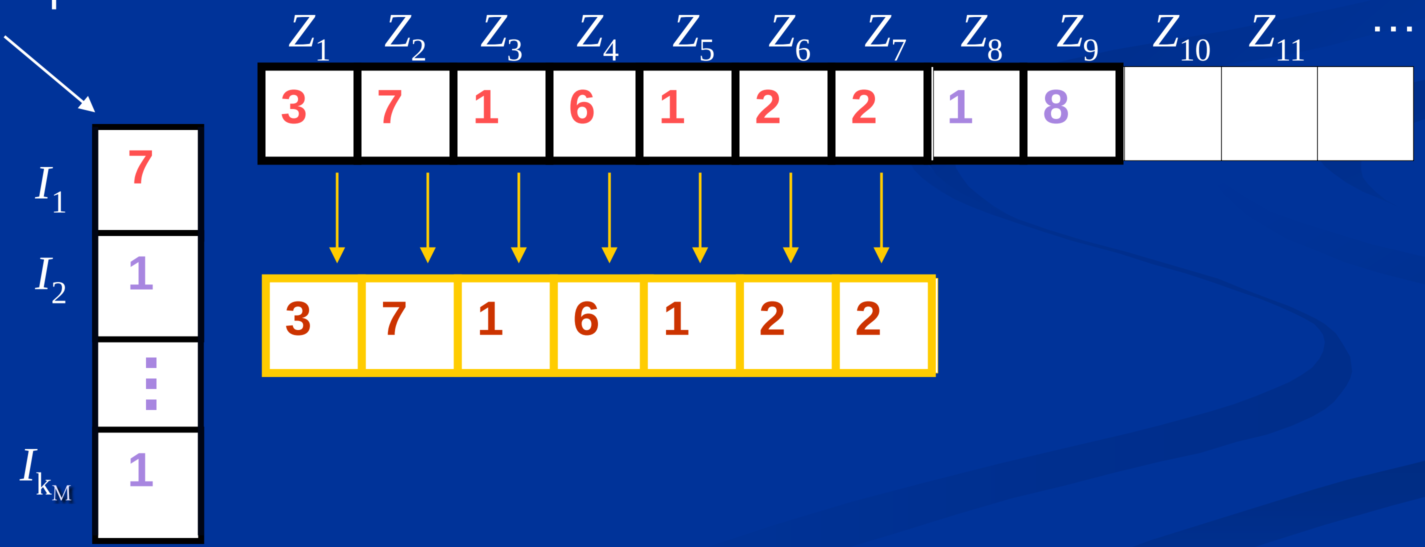


gassnerc@uni-greifswald.de

The output

The output: $(Z_1, \ldots, Z_{I_1})$

The size of the output



The class DEC_Σ

Decidable problems

$B \subseteq U^\infty$, $a, b \in U$ are constants with $a \neq b$.

$B \in \mathrm{DEC}_\Sigma$

- if the characteristic function is computable,

if there is a machine with

Input: $(x_1, \ldots, x_n) \in U^\infty$;

Output a (or halt) if $(x_1, \ldots, x_n) \in B$. Acceptance.

Output b (or no halt) if $(x_1, \ldots, x_n) \notin B$. Deterministic rejection.

gassnerc@uni-greifswald.de

The class DEC_Σ

Decidable problems

$B \subseteq U^\infty$, $a, b \in U$ are constants with $a \neq b$.

$B \in \mathrm{DEC}_\Sigma$

- if the characteristic function is computable,
- if there is a machine with

Input: $(x_1, \ldots, x_n) \in U^\infty$;

Output a (or halt) if $(x_1, \ldots, x_n) \in B$. Acceptance.

Output b (or no halt) if $(x_1, \ldots, x_n) \notin B$. Deterministic rejection.

The class DEC_Σ

Decidable problems

$B \subseteq U^\infty$, $a, b \in U$ are constants with $a \neq b$.

$B \in \mathrm{DEC}_\Sigma$

- if the characteristic function is computable,
- if there is a machine with

Input: $(x_1, \ldots, x_n) \in U^\infty$;

Output a (or halt) if $(x_1, \ldots, x_n) \in B$. Acceptance.

Output b (or no halt) if $(x_1, \ldots, x_n) \notin B$. Deterministic rejection.

The class DEC_Σ

Decidable problems

$B \subseteq U^\infty$, $a, b \in U$ are constants with $a \neq b$.

$B \in \mathrm{DEC}_\Sigma$

- if the characteristic function is computable,
- if there is a machine with

  Input: $(x_1, \ldots, x_n) \in U^\infty$;

  Output a (or halt) if $(x_1, \ldots, x_n) \in B$. Acceptance.

  Output b (or no halt) if $(x_1, \ldots, x_n) \notin B$. Deterministic rejection.

gassnerc@uni-greifswald.de

The class DEC_Σ

Decidable problems

$B \subseteq U^\infty$, $a, b \in U$ are constants with $a \neq b$.

$B \in \mathrm{DEC}_\Sigma$

- if the characteristic function is computable,
- if there is a machine with

Input: $(x_1, \ldots, x_n) \in U^\infty$;

Output a (or halt) if $(x_1, \ldots, x_n) \in B$. Acceptance.

Output b (or no halt) if $(x_1, \ldots, x_n) \notin B$. Deterministic rejection.

Halting problems for Σ

$$H_\Sigma = \{(\underbrace{x_1,\ldots,x_n}_{\boldsymbol{x}}, Code(M)) \mid$$

$$\boldsymbol{x} \in U^\infty \;\;\&\;\; M \text{ is a deterministic } \Sigma\text{-machine}$$

$$\&\;\; M \text{ halts on } \boldsymbol{x}\}$$

$$H_\Sigma^{\text{spec}} = \{Code(M) \mid M \text{ is a deterministic } \Sigma\text{-machine}$$

$$\&\;\; M \text{ halts on } Code(M)\}$$

Diagonalization techniques

The undecidability of the Halting problem H_Σ (for Turing machines)

1. The set of machines is countable. Assume that H_Σ^{spec} is decidable.

Halt?	bin(1)	…	bin(i)	…	bin(j)	…	…
M_1	yes / no						
:		…					
M_i			yes				
:				…			
M_j					no		
:						…	
:							
M								↯

⇒ There is an M recognizing the complement of H_Σ^{spec}. ↯

gassnerc@uni-greifswald.de

Diagonalization techniques

The undecidability of the Halting problem H_Σ (for Turing machines)

1. The set of machines is countable. Assume that H_Σ^{spec} is decidable.

Halt?	bin(1)	…	bin(i)	…	bin(j)	…	…
M_1	yes / no						
:		…					
M_i			yes				
:				…			
M_j					no		
:						…	
:							
M							

↯

⇒ There is an M recognizing the complement of H_Σ^{spec}. ↯

Diagonalization techniques

The undecidability of the Halting problem H_Σ (for Turing machines)

1. The set of machines is countable. Assume that H_Σ^{spec} is decidable.

Halt?	bin(1)	⋯	bin(i)	⋯	bin(j)	⋯	⋯
M_1	yes / no						
⋮		⋯					
M_i			yes				
⋮				⋯			
M_j					no		
⋮						⋯	
⋮							
M	no / yes	⋯	no	⋯	yes	⋯	⋯

⇒ There is an M recognizing the complement of H_Σ^{spec}. ↯

Diagonalization techniques

The undecidability of the Halting problem H_Σ (for $(\mathbb{R}; \mathbb{R}; +, -, \cdot; \leq)$)

2. The codes of machines are ordered. Assume that H_Σ^{spec} is decidable.

Halt?	…	…	$Code(M_i)$	…	$Code(M_j)$	…	…
:	…						
:		…					
M_i			yes				
:				…			
M_j					no		
:						…	
:							
M	…	…	no	…	yes	…	…

⇒ There is an M recognizing the complement of H_Σ^{spec}. ⚡

gassnerc@uni-greifswald.de

Diagonalization techniques

The undecidability of the Halting problem H_Σ (for any structure)

3. Σ arbitrary (We can generalize the result; CCA 2008.)

Assume: H_Σ is decidable.

$\Rightarrow$ H_Σ^{spec} is decidable.

$\Rightarrow$ The complement of H_Σ^{spec} is semi-decidable by a Σ-machine M.

$\Rightarrow$ M halts on $Code(M)$

$\Leftrightarrow$ M does not halt on $Code(M)$.

$\Rightarrow$ ↯

gassnerc@uni-greifswald.de

The class P_Σ

Computation in polynomial time

A machine M decides a problem in polynomial time

if there is some polynomial p_M such that

$$M \text{ halts for } \boldsymbol{x} = (x_1, \ldots, x_n) \text{ within } p_M(n) \text{ steps.}$$

Each instruction is executed within one fixed time unit.

⇨ $P_\Sigma \subseteq DEC_\Sigma$ ($P_\Sigma \triangleq$ problems are decidable in polynomial time)

gassnerc@uni-greifswald.de

The class NP_Σ

The non-deterministic instructions

The non-deterministic:

guess(Z_k); Arbitrary elements can be guessed!

⇨ $P_\Sigma \subseteq NP_\Sigma$

Non-deterministic acceptance: output a by means of guessed elements.

Non-deterministic rejection: if the input cannot be accepted.

$NP_\Sigma \not\subseteq DEC_\Sigma \Rightarrow P_\Sigma \neq NP_\Sigma$ wegen $P_\Sigma \subseteq DEC_\Sigma$ und $P_\Sigma \subseteq NP_\Sigma$.

gassnerc@uni-greifswald.de

The class NP_Σ

The non-deterministic instructions

The non-determinism:

guess(Z_k); Arbitrary elements can be guessed!

$\Rightarrow \quad P_\Sigma \subseteq NP_\Sigma$

Non-deterministic acceptance: output a by means of guessed elements.
Non-deterministic rejection: if the input cannot be accepted.

$NP_\Sigma \not\subseteq DEC_\Sigma \Rightarrow P_\Sigma \neq NP_\Sigma$ wegen $P_\Sigma \subseteq DEC_\Sigma$ und $P_\Sigma \subseteq NP_\Sigma$.

gassnerc@uni-greifswald.de

Some $P_\Sigma \stackrel{?}{=} NP_\Sigma$ problems for several structures

Σ	$P_\Sigma = NP_\Sigma$?
$(\mathbb{C};\mathbb{C};+,-,\cdot;\ =)$	**?**
$(\mathbb{R};\mathbb{R};+,-,\cdot;\ \leq)$	**?**
$(\mathbb{R};\mathbb{R};+,-,\cdot;\ =)$	no ($\leq$)
$(\mathbb{Q};\mathbb{Q};+,-,\cdot;\leq)$, $(\mathbb{Q};\mathbb{Q};+,-,\cdot;=)$	no (rational square numbers)
$(\mathbb{R};\mathbb{R};+,-;\ \leq)$	**?**
$(\mathbb{R};\mathbb{R};+,-;\ =)$	no (Meer / Koiran)
$(\mathbb{Z};\mathbb{Z};+,-;\leq)$, $(\mathbb{Z};\mathbb{Z};+,-;=)$	no (even integers)
$(\mathbb{Z};1;(\varphi_s)_{s\in\mathbb{Z}};=)$ $\quad \varphi_s(x)=sx$	no (no NP-complete problem)

gassnerc@uni-greifswald.de

Example for $P_\Sigma \neq NP_\Sigma$

$\Sigma = \mathbb{Z} = (\mathbb{Z}; 0, 1; \cdot, +, -; =)$

$A = \{(x,\dots,x) \mid \exists y\ (x = y^2)\}.$

$A \in NP_\Sigma$ (we can guess $y \in \mathbb{Z}$).

$A \notin P_\Sigma$:

Example for $P_\Sigma \neq NP_\Sigma$

$\Sigma = \mathbb{Z} = (\mathbb{Z}; 0, 1; \cdot, +, -; =)$

$A = \{(x, \ldots, x) \mid \exists y\, (x = y^2)\}.$

$A \in NP_\Sigma$ (we can guess $y \in \mathbb{Z}$).

$A \notin P_\Sigma$:

gassnerc@uni-greifswald.de

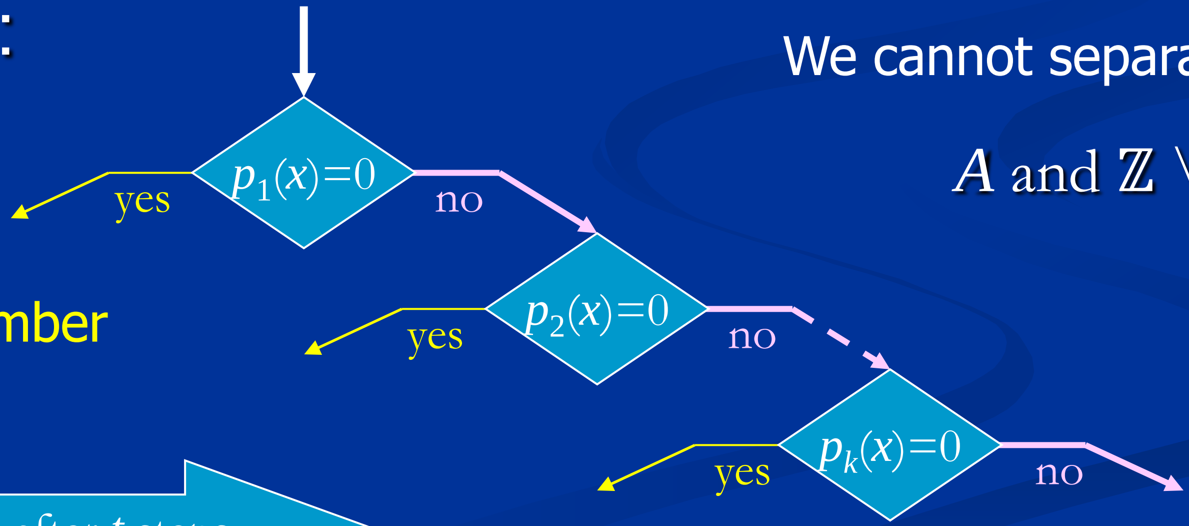
Example for $P_\Sigma \neq NP_\Sigma$

$\Sigma = \mathbb{Z} = (\mathbb{Z}; 0, 1; \cdot, +, -; =)$

$A = \{(x,\ldots,x) \mid \exists y\, (x = y^2)\}.$

$A \in NP_\Sigma$ (we can guess $y \in \mathbb{Z}$).

$A \notin P_\Sigma$:



We cannot separate

A and $\mathbb{Z} \setminus A$.

Only a finite number of zeros.

The machine halts after t steps.

gassnerc@uni-greifswald.de

Some P_Σ-NP_Σ problems

for structures over numbers

Σ	$P_\Sigma = DNP_\Sigma$?	$DNP_\Sigma = NP_\Sigma$?
$(\mathbb{C}\,;\,\mathbb{C}\,;\,+,-,\cdot\,;\ =)$	**?**	**?**
$(\mathbb{R}\,;\,\mathbb{R}\,;\,+,-,\cdot\,;\ \leq)$	**?**	**?**
$(\mathbb{R}\,;\,\mathbb{R}\,;\,+,-,\cdot\,;\ =)$	**?**	no ($\leq$)
$(\mathbb{R}\,;\,\mathbb{R}\,;\,+,-\,;\ \leq)$	**?**	yes (Koiran)
$(\mathbb{R}\,;\,\mathbb{R}\,;\,+,-\,;\ =)$	no (Meer)	yes (Koiran)
$(\mathbb{Z}\,;\,\mathbb{Z}\,;\,+,-\,;\,\leq)$	**?**	no (even integers)
$(\mathbb{Z}\,;\,\mathbb{Z}\,;\,+,-\,;\,=)$	no (for groups)	no (even integers)

DN $\triangleq$ digitally non-deterministic:
$y_1,\ldots,y_m \in \{0\,,\,1\}$

Example for $P_\Sigma \neq DNP_\Sigma$

$\Sigma = \mathbb{Z}_{add} = (\mathbb{Z}; 0, 1; +, - ; =)$

D ≙ digitally.

$A = \cup_{n\geq 1}\{(x,\ldots,x) \in \mathbb{Z}^n \mid x < 2^n\}.$

$A \in DNP_\Sigma$ (we can guess the binary code of x: $(y_1,\ldots,y_n) \in \{0,1\}^n$).

gassnerc@uni-greifswald.de

Example for $P_\Sigma \neq DNP_\Sigma$

$\Sigma = \mathbb{Z}_{add} = (\mathbb{Z}; 0, 1; +, - \ ; =)$

$D \triangleq$ digitally.

$A = \cup_{n\geq 1}\{(x,\ldots,x) \in \mathbb{Z}^n \mid x < 2^n\}.$

$A \in DNP_\Sigma$ (we can guess the binary code of x: $(y_1,\ldots,y_n)\in\{0,1\}^n$).

Example for $P_\Sigma \neq DNP_\Sigma$

$\Sigma = \mathbb{Z}_{add} = (\mathbb{Z}; 0, 1; +, - ; =)$

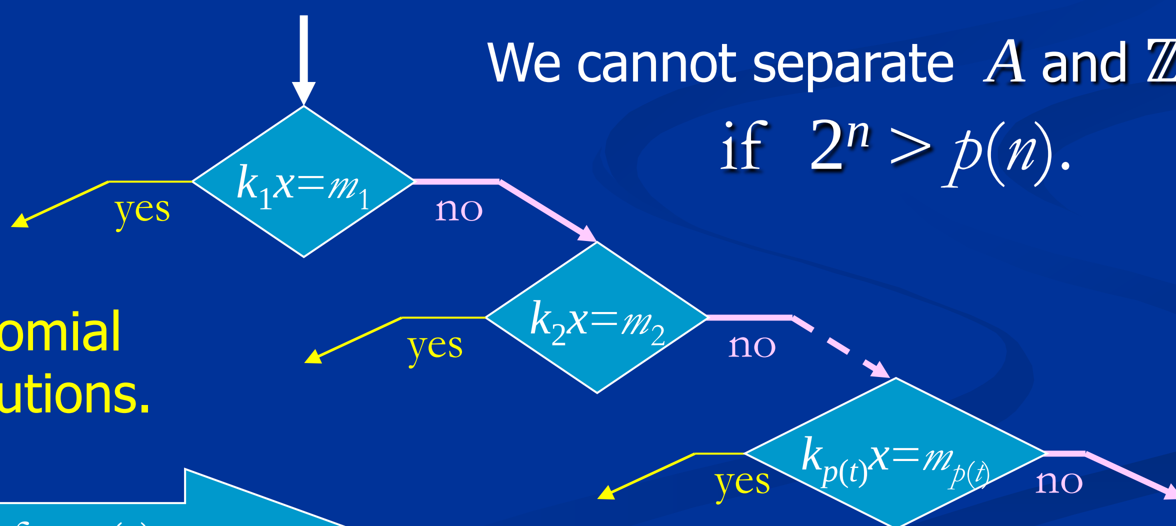
$A = \cup_{n \geq 1} \{(x, \ldots, x) \in \mathbb{Z}^n \mid x < 2^n\}.$

D ≙ digitally.

$A \in DNP_\Sigma$ (we can guess the binary code of x: $(y_1, \ldots, y_n) \in \{0,1\}^n$).

$A \notin P_\Sigma$:

We cannot separate A and $\mathbb{Z}^n \setminus A$ if $2^n > p(n)$.



Only a polynomial number of solutions.

The machine halts after $p(n)$ steps

gassnerc@uni-greifswald.de

DEC_Σ , P_Σ , NP_Σ

The size of an input $(x_1,\dots, x_n)$: $\boldsymbol{n}$.

⇨ Every $u \in U$ can be stored in one register.

The execution of any instruction: **one time unit** (one step).

⇨ The execution of one operation $\triangleq$ one time unit.

Computation in polynomial time: **output after $p(n)$ steps** for any input $(x_1,\dots, x_n)$ and some polynomial p.

⇨ $\mathrm{P}_\Sigma \subseteq \mathrm{NP}_\Sigma$

⇨ $\mathrm{P}_\Sigma \subseteq \mathrm{DEC}_\Sigma$

⇨ $\mathrm{NP}_\Sigma \not\subseteq \mathrm{DEC}_\Sigma \;\Rightarrow\; \mathrm{P}_\Sigma \neq \mathrm{NP}_\Sigma$

gassnerc@uni-greifswald.de

NP-complete problems

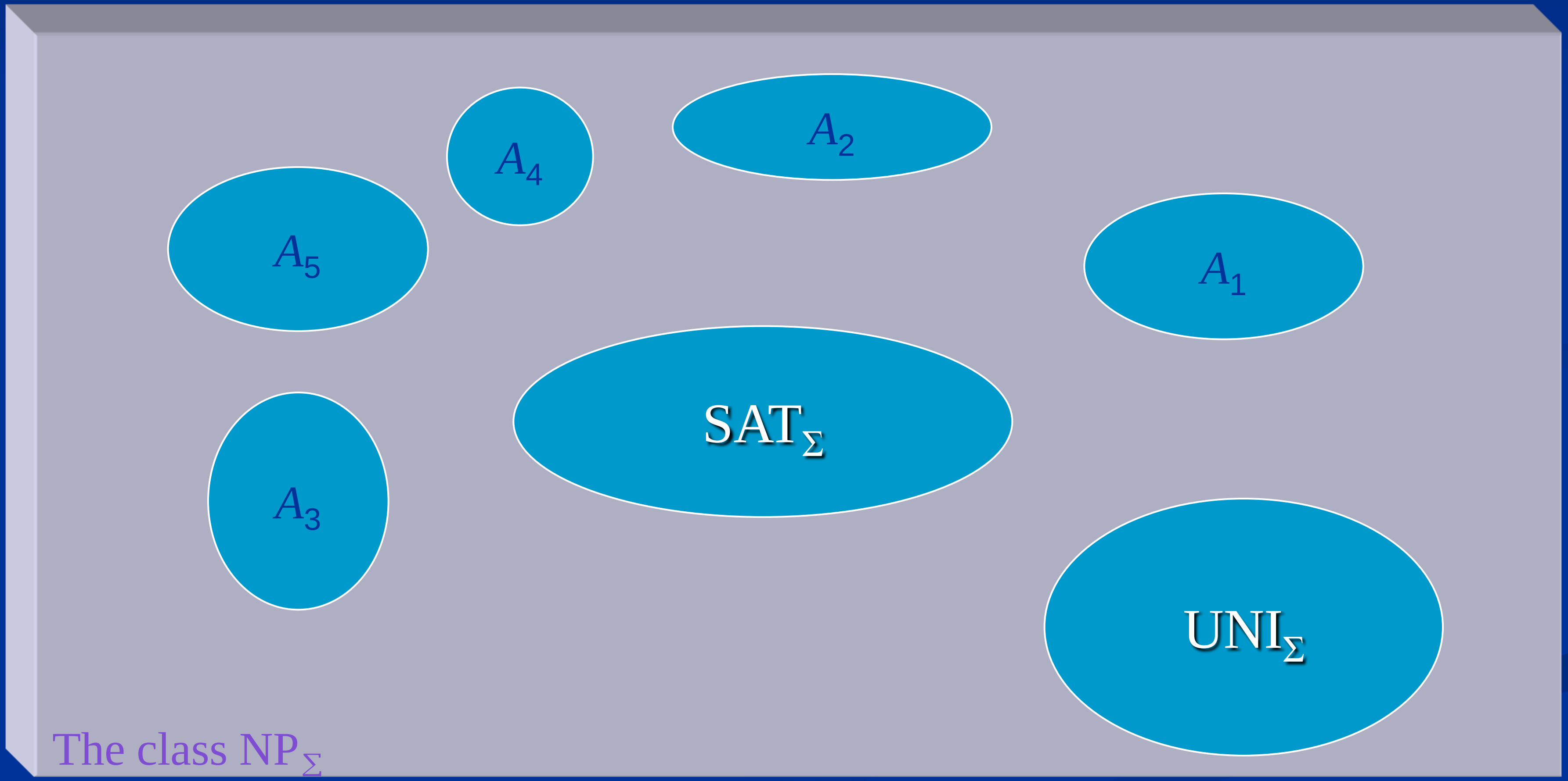
The Satisfiability Problem

$$\mathbf{SAT}_{\Sigma} = \{(\boldsymbol{x}, code(\psi)) \in U^{\infty} \mid \psi \text{ quantifier-free } (\neg, \vee, \wedge)\text{-formulae } \& \ \Sigma \models \exists \boldsymbol{Y}\, \psi(\boldsymbol{x}, \boldsymbol{Y})\}$$

The NP-complete problem recognized by a usual universal machine

$$\mathbf{UNI}_{\Sigma} = \{(b,\ldots, b, x_1,\ldots, x_n, Code(M)) \in U^{t+n+k} \mid M \text{ is a non-deterministic } \Sigma\text{-machine } \& \ M \text{ accepts } (x_1,\ldots, x_n) \text{ within } t \text{ steps}\} \subseteq U^{\infty}$$

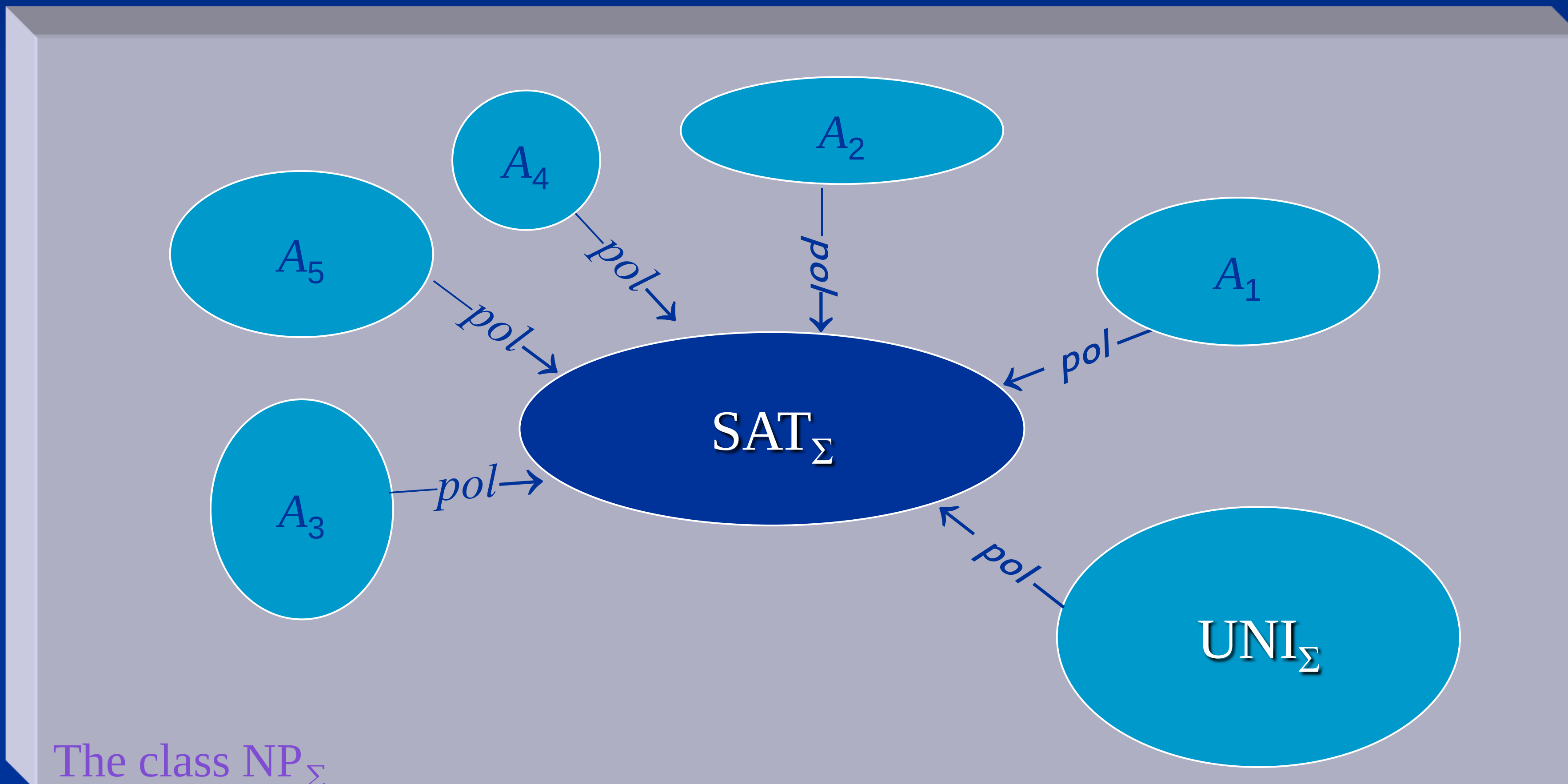
The meaning of these NP-complete problems



gassnerc@uni-greifswald.de

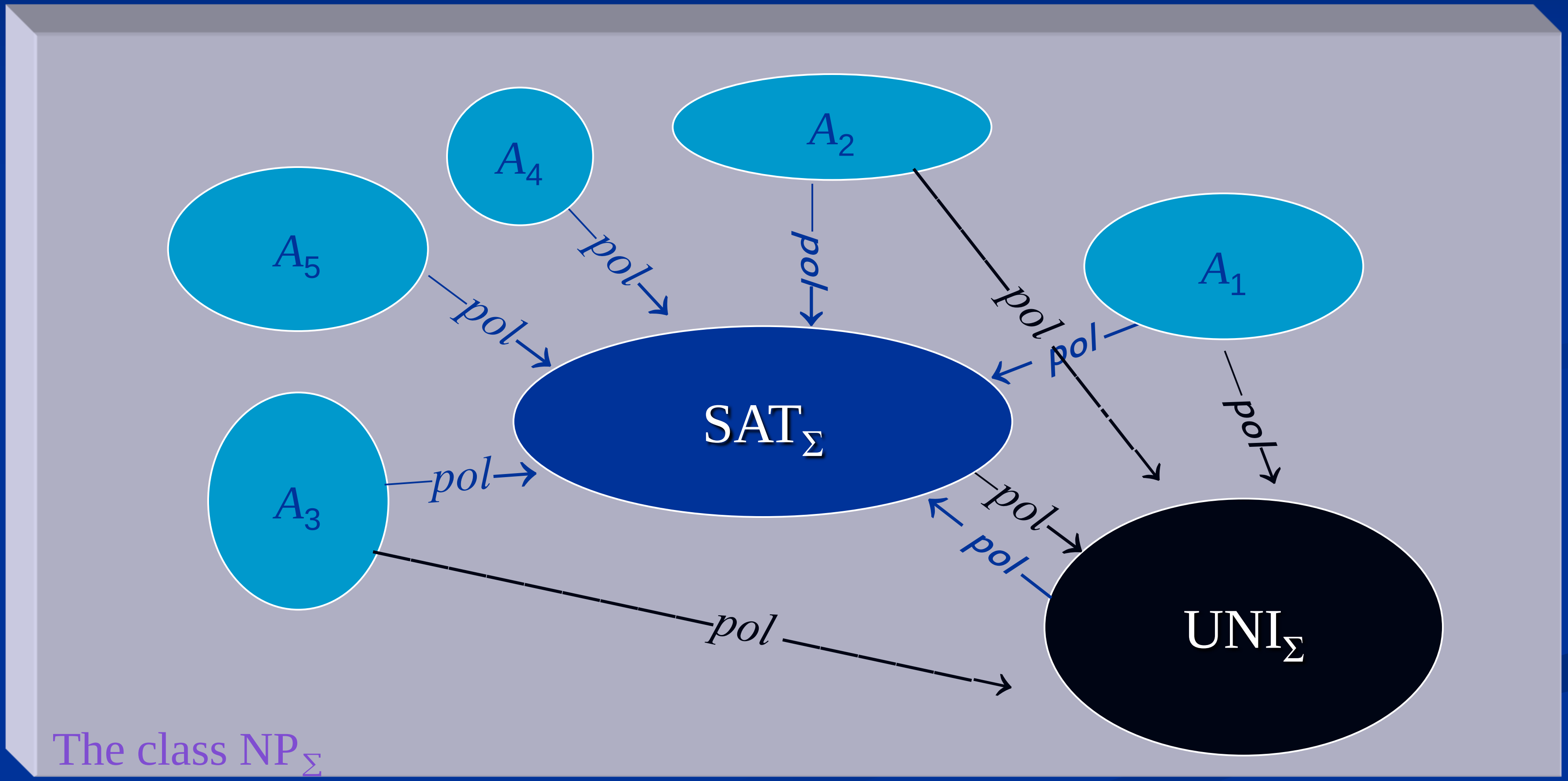
The meaning of these NP-complete problems

Any problem can be reduced to SAT_{Σ} and to UNI_{Σ} in polynomial time.



The meaning of these NP-complete problems

Any problem can be reduced to SAT_Σ and to UNI_Σ in polynomial time.



gassnerc@uni-greifswald.de

Properties of SAT_Σ and UNI_Σ

$SAT_\Sigma \in NP_\Sigma$. SAT_Σ is NP_Σ-complete.

$SAT_\Sigma \in P_\Sigma \quad \Rightarrow P_\Sigma = NP_\Sigma$

$SAT_\Sigma \notin DEC_\Sigma \quad \Rightarrow P_\Sigma \neq NP_\Sigma$

$UNI_\Sigma \in NP_\Sigma$. UNI_Σ is NP_Σ-complete.

$UNI_\Sigma \in P_\Sigma \quad \Rightarrow P_\Sigma = NP_\Sigma$

$UNI_\Sigma \notin DEC_\Sigma \quad \Rightarrow P_\Sigma \neq NP_\Sigma$

Oracle machines

Oracle query:

$$l\text{: if } \underbrace{(Z_1,\ldots,Z_{I_1})} \in B \text{ then goto } l_1 \text{ else goto } l_2\,;$$

The length can be computed by $I_1 := 1$; $I_1 := I_1 + 1$; ….

B oracle, $B \subseteq U^\infty = \cup_{n\geq 1} U^n$

We will define oracles such that

$$\mathrm{P}_\Sigma{}^{Q} \neq \mathrm{NP}_\Sigma{}^{Q},$$

$$\mathrm{P}_\Sigma{}^{O} = \mathrm{NP}_\Sigma{}^{O}.$$

An oracle Q with $P_\Sigma^Q \neq NP_\Sigma^Q$

Using the undecidability of the Halting problem H_Σ

U infinite,
a finite number of operations and relations,

$\{\alpha_1, \alpha_2, \alpha_3, \ldots\} \subseteq U$ enumerable and decidable.

$Q = Q_\Sigma = \{ (\alpha_t, \boldsymbol{x}, Code(M)) \mid$

$\boldsymbol{x} \in U^\infty$ & M is a deterministic Σ-machine

& $M(\boldsymbol{x})\downarrow^t\}$

M accepts $\boldsymbol{x} = (x_1, \ldots, x_n) \in U^\infty$ within t steps.

Proposition (CCA 2008): $H_\Sigma \in NP_\Sigma^Q \setminus P_\Sigma^Q$. $(P_\Sigma^Q \subseteq DEC_\Sigma)$

gassnerc@uni-greifswald.de

An oracle $\mathcal{O}_\Sigma$ with $P_\Sigma^{\mathcal{O}_\Sigma} = NP_\Sigma^{\mathcal{O}_\Sigma}$

A universal oracle:

$$\mathcal{O} = \mathcal{O}_\Sigma = \{ (\overbrace{b,\ldots,b}^{\in U^t}, \boldsymbol{x}, Code(M)) \mid$$

$$\boldsymbol{x} \in U^\infty \;\; \& \;\; M \text{ is a non-deterministic } \Sigma\text{-machine using } \mathcal{O}$$

$$\& \;\; M(\boldsymbol{x})\downarrow^t \}$$

(cp. also Baker, Gill, and Solovay; Emerson; ... for Turing machines...)

Proposition (CCA 2008): $P_\Sigma^{\mathcal{O}} = NP_\Sigma^{\mathcal{O}}$.

gassnerc@uni-greifswald.de

An oracle $\mathcal{O}_\Sigma$ containing only tuples of length 1 with $P_\Sigma^{\mathcal{O}_\Sigma} = NP_\Sigma^{\mathcal{O}_\Sigma}$?

Structures over strings

$\Sigma = (U^*; \varepsilon, a, b, c_3,\ldots, c_u; \; add, sub_\mathrm{l}, sub_\mathrm{r}, \; f_1,\ldots, f_v; R_1,\ldots,R_w, \; =)$

$(d_1,\ldots, d_k) \in U^k \subset U^\infty$ stored in k registers

!

$s = d_1 \cdots d_k \in U^*$ stored in one register

$d \in U$

$add(s, d) = sd$ $\quad sub_\mathrm{l}(sd) = s$ $\quad sub_\mathrm{r}(sd) = d$

gassnerc@uni-greifswald.de

An oracle $\mathcal{O}_\Sigma$ containing only tuples of length 1 with $\mathrm{P}_\Sigma^{\mathcal{O}_\Sigma} = \mathrm{NP}_\Sigma^{\mathcal{O}_\Sigma}$?

Recall: $\mathrm{P}_\Sigma^{\mathcal{O}_\Sigma} = \mathrm{NP}_\Sigma^{\mathcal{O}_\Sigma}$ and $\mathrm{P}_\Sigma^{\mathcal{Q}_\Sigma} \neq \mathrm{NP}_\Sigma^{\mathcal{Q}_\Sigma}$ for

$$\mathcal{O}_\Sigma = \{\, (\underbrace{b,\ldots,b}_{t\,\times}, \boldsymbol{x}, \mathit{Code}(M)) \mid \boldsymbol{x} \in (U^*)^\infty \;\&\; M \text{ is a non-deterministic } \Sigma\text{-machine using } \mathcal{O}_\Sigma \;\&\; M(\boldsymbol{x})\downarrow^t \}$$

$$\mathcal{Q}_\Sigma = \{\, (\underbrace{b\cdots b}_{t\,\times}, \boldsymbol{x}, \mathit{Code}(M)) \mid \boldsymbol{x} \in (U^*)^\infty \;\&\; M \text{ is a deterministic } \Sigma\text{-machine} \;\&\; M(\boldsymbol{x})\downarrow^t \}$$

Theorem (CCA 2008): There is not an oracle $\mathcal{O}$ with

$$b\cdots b\cdot \mathrm{string}(\boldsymbol{x})\cdot \mathrm{string}(\mathit{Code}(M)) \in \mathcal{O}$$

$$\Leftrightarrow \quad \boldsymbol{x} \in (U^*)^\infty \;\&\; M \text{ is a non-deterministic } \Sigma\text{-machine using } \mathcal{O} \;\&\; M(\boldsymbol{x})\downarrow^t .$$

No set !

Structures with P = NP

An additional relation R
on padded codes
of the elements of a universal oracle O
with $P_{\Sigma}^{O} = NP_{\Sigma}^{O}$

P = NP for

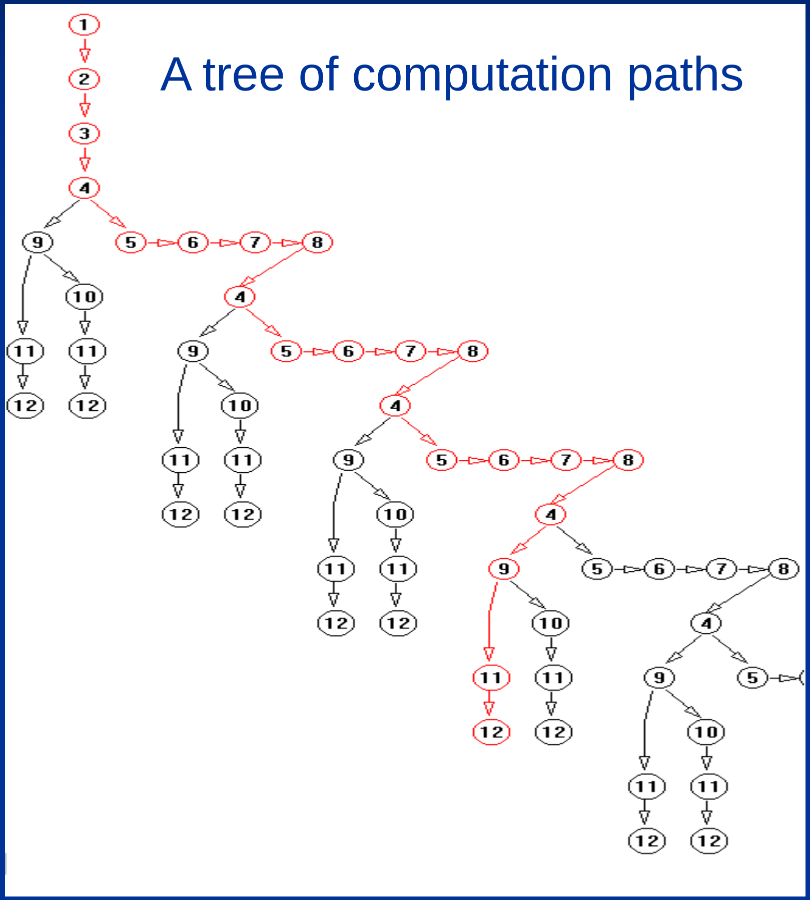
P = NP for

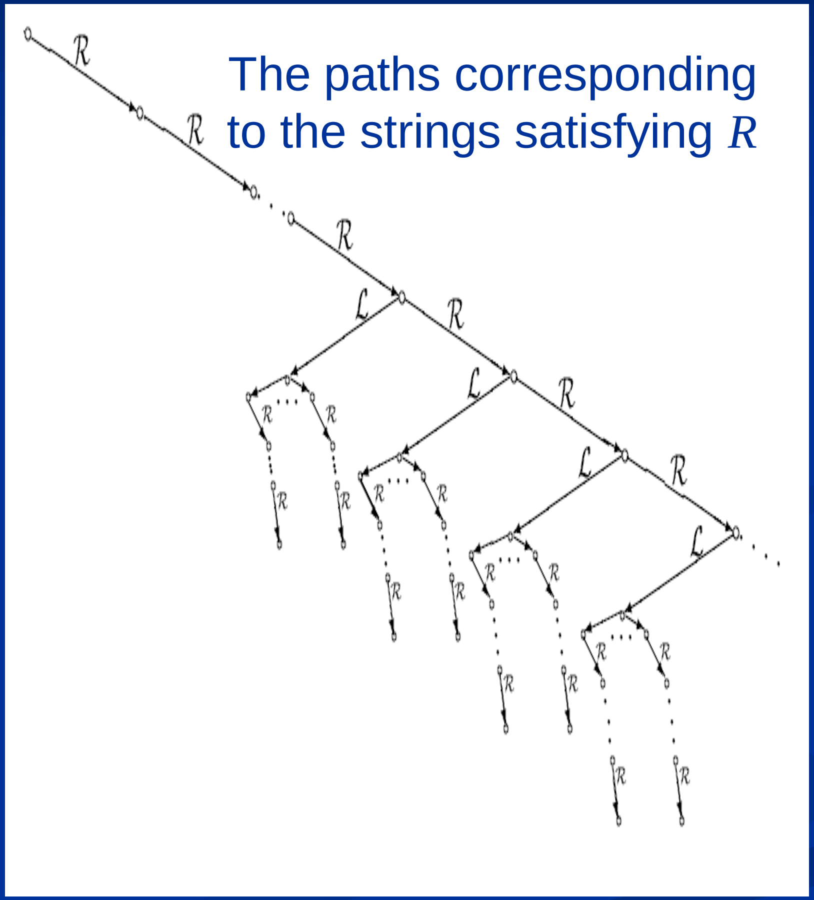
Binary trees
with decidable identity relation
(Gaßner, Dagstuhl 2004)

Strings
with identity relation and relation R recursively defined by means of $SAT_{\Sigma R}$
(Gaßner, CiE 2006)

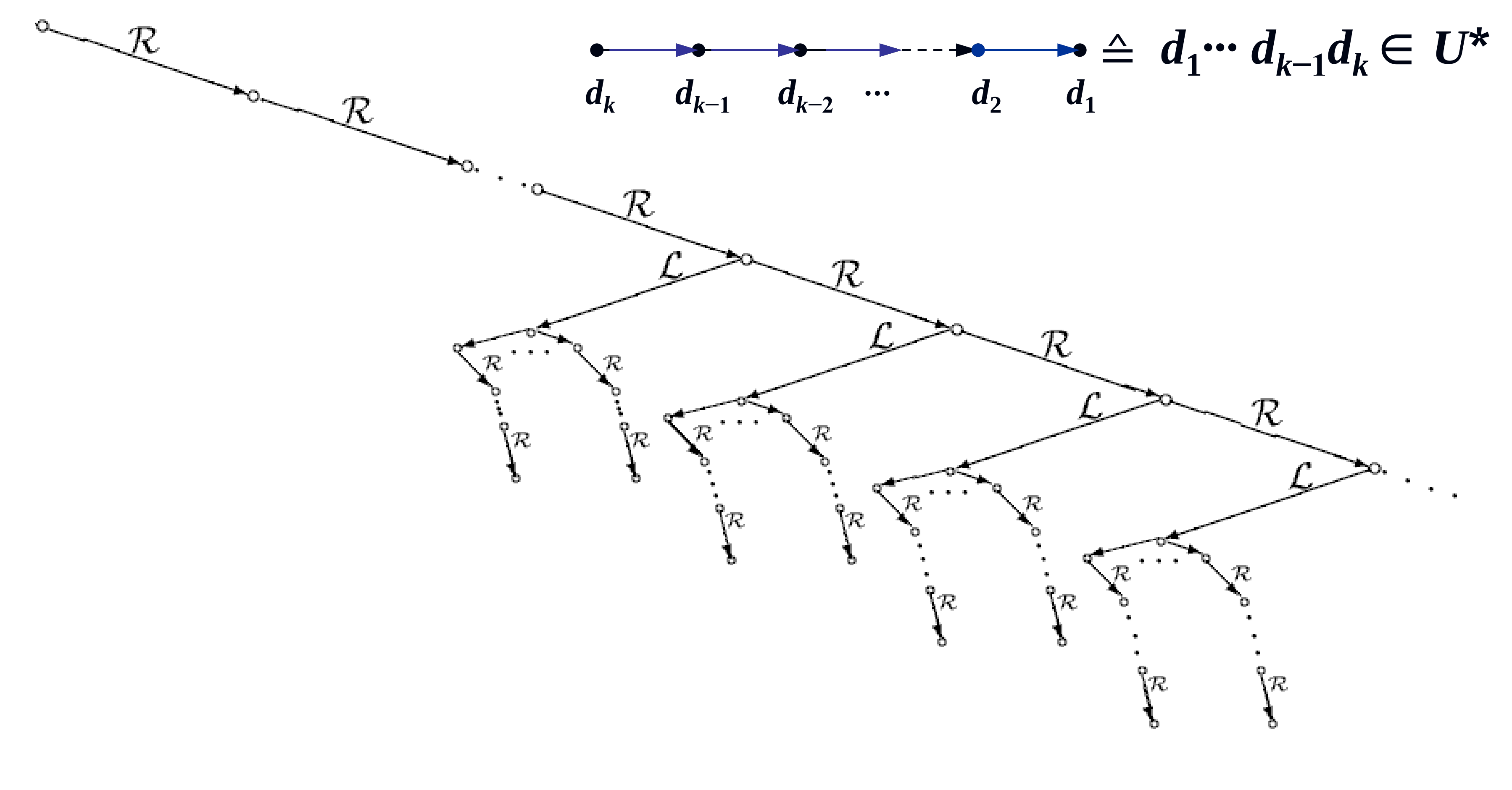
Strings
with operations for adding and deleting the last character
(Gaßner, CiE 2007)

The idea - similar trees

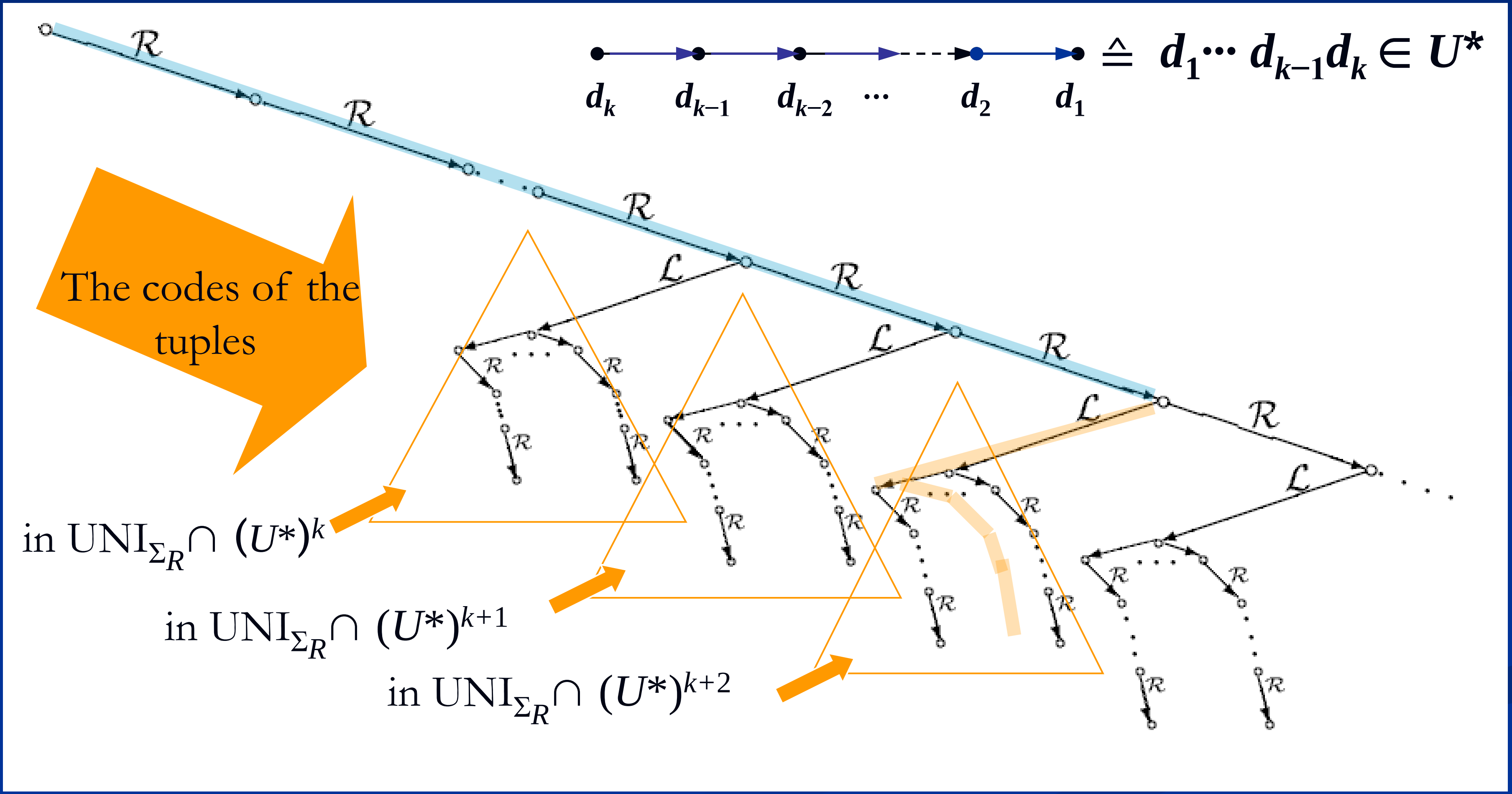




The description of UNI_{Σ_R} by a tree



The description of UNI_{Σ_R} by a tree

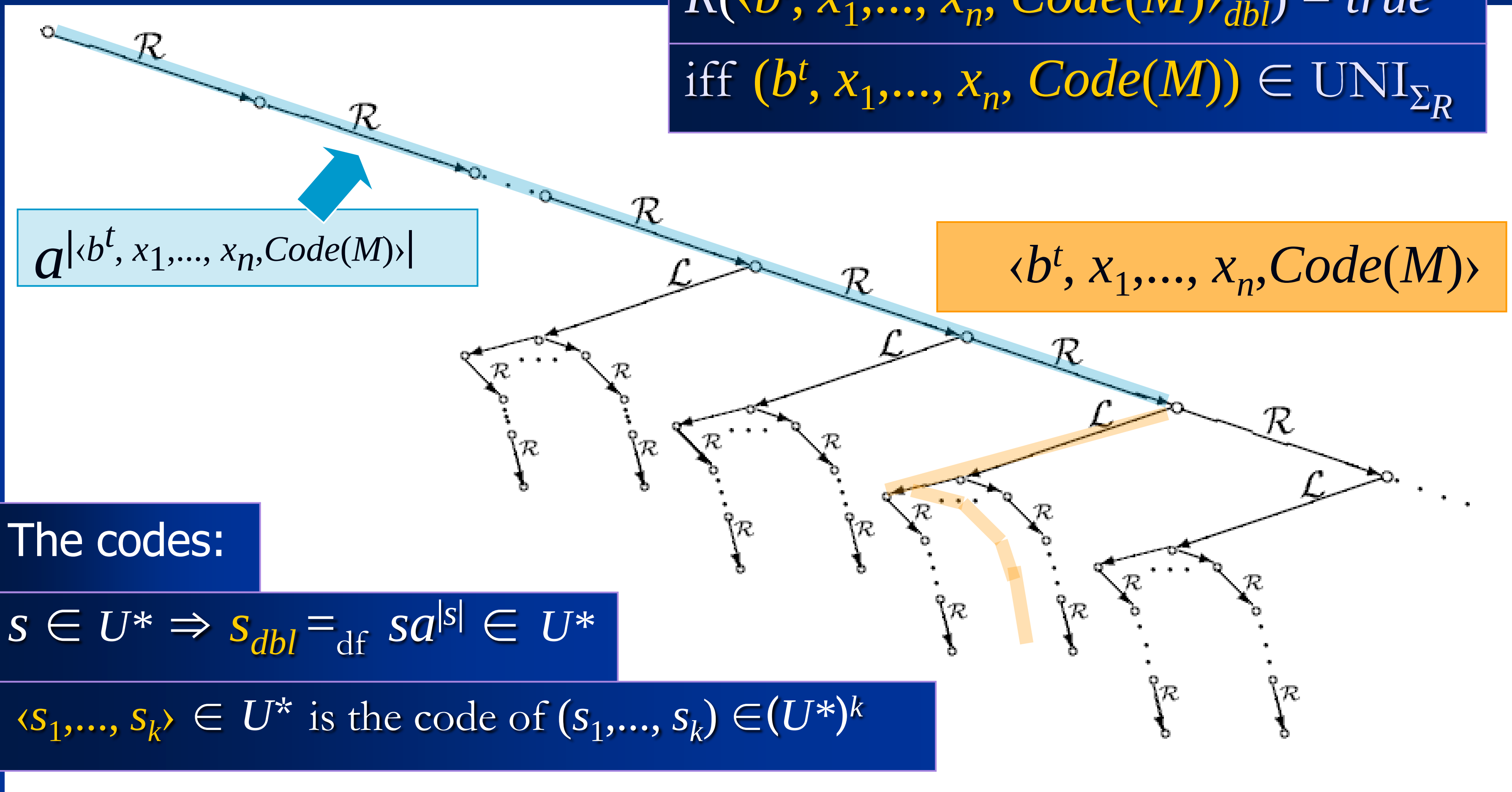


gassnerc@uni-greifswald.de

Some R with $\mathrm{UNI}_{\Sigma_R} \in \mathrm{DEC}_{\Sigma_R}$

$R(\langle b^t, x_1,\ldots, x_n, Code(M)\rangle_{dbl}) = true$

iff $(b^t, x_1,\ldots, x_n, Code(M)) \in \mathrm{UNI}_{\Sigma_R}$



The codes:

$s \in U^* \Rightarrow s_{dbl} =_{\mathrm{df}} sa^{|s|} \in U^*$

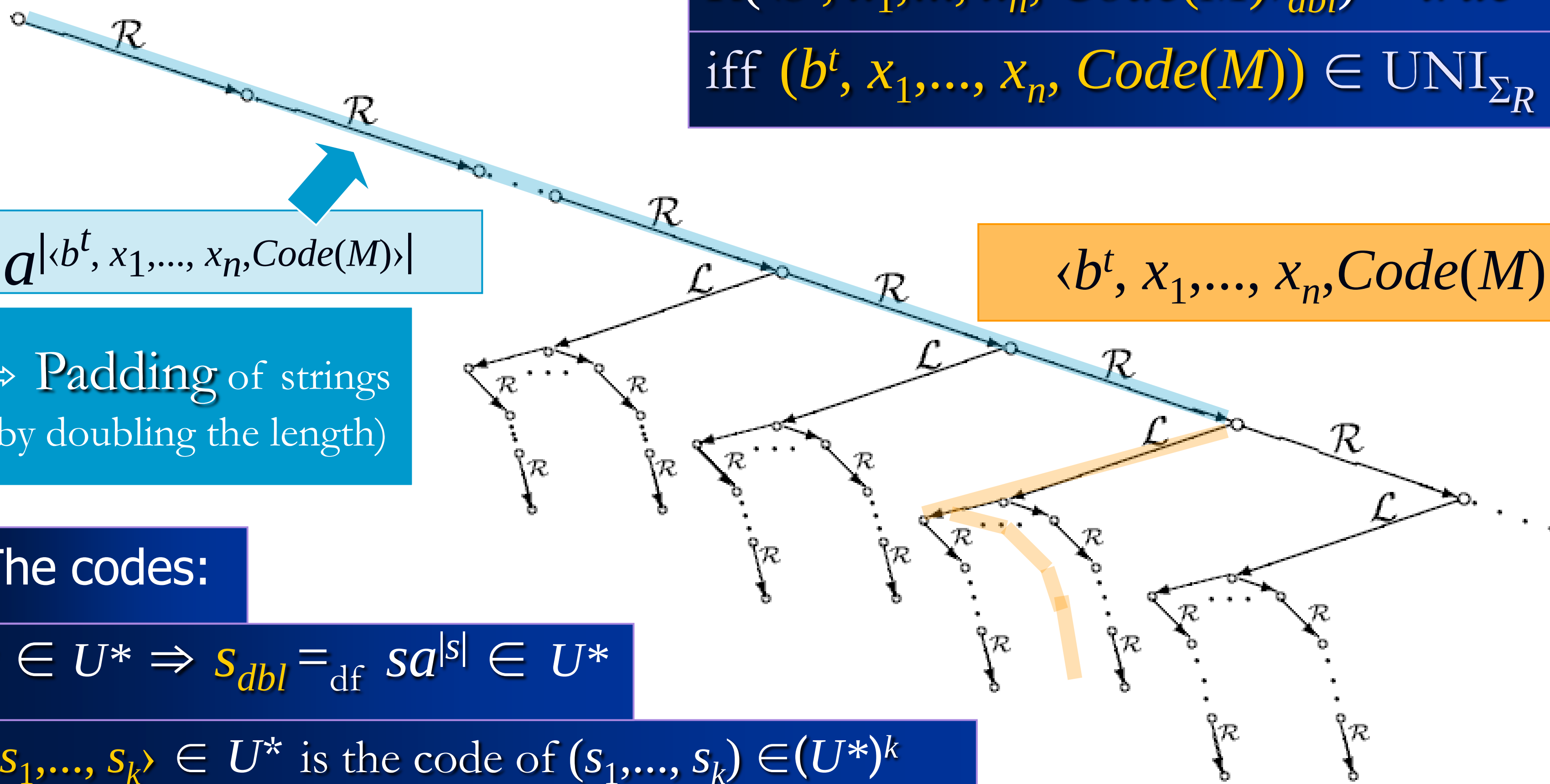
$\langle s_1,\ldots, s_k\rangle \in U^*$ is the code of $(s_1,\ldots, s_k) \in (U^*)^k$

gassnerc@uni-greifswald.de

Some R with $\mathrm{UNI}_{\Sigma_R} \in \mathrm{DEC}_{\Sigma_R}$

$R(\langle b^t, x_1,\ldots, x_n, Code(M)\rangle_{dbl}) = true$

iff $(b^t, x_1,\ldots, x_n, Code(M)) \in \mathrm{UNI}_{\Sigma_R}$



$a^{|\langle b^t, x_1,\ldots, x_n, Code(M)\rangle|}$

$\langle b^t, x_1,\ldots, x_n, Code(M)\rangle$

⇨ Padding of strings (by doubling the length)

The codes:

$s \in U^* \Rightarrow s_{dbl} =_{\mathrm{df}} sa^{|s|} \in U^*$

$\langle s_1,\ldots, s_k\rangle \in U^*$ is the code of $(s_1,\ldots, s_k) \in (U^*)^k$

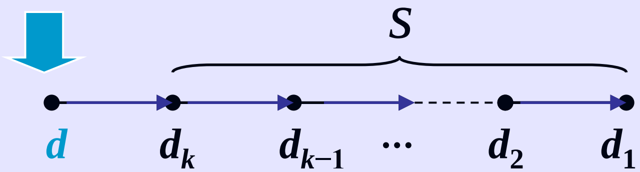
gassnerc@uni-greifswald.de

The new operations for slow (!!!) computation

$\Sigma \quad = (U ; c_1,\ldots, c_u; \; f_1,\ldots, f_v; R_1,\ldots, R_w, =),$

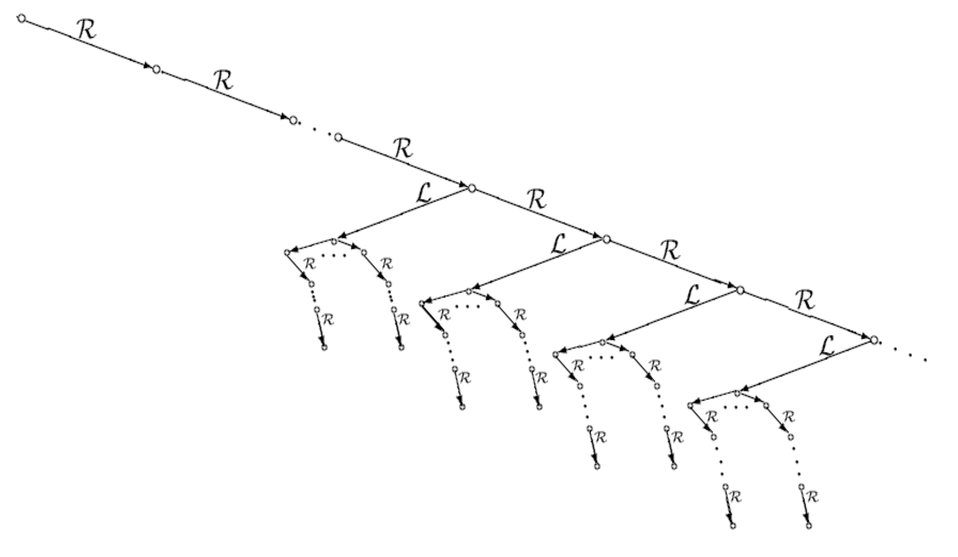
$\Sigma_R = (U^* ; \; \varepsilon, a, b, c_1 ,\ldots, c_u ; \; add, sub_l , sub_r , \; f_1',\ldots, f_v'; R_1',\ldots, R_w', R , =)$

$add(s, d) = sd \qquad sub_l(sd) = s \qquad sub_r(sd) = d \qquad s \in U^*, d \in U$



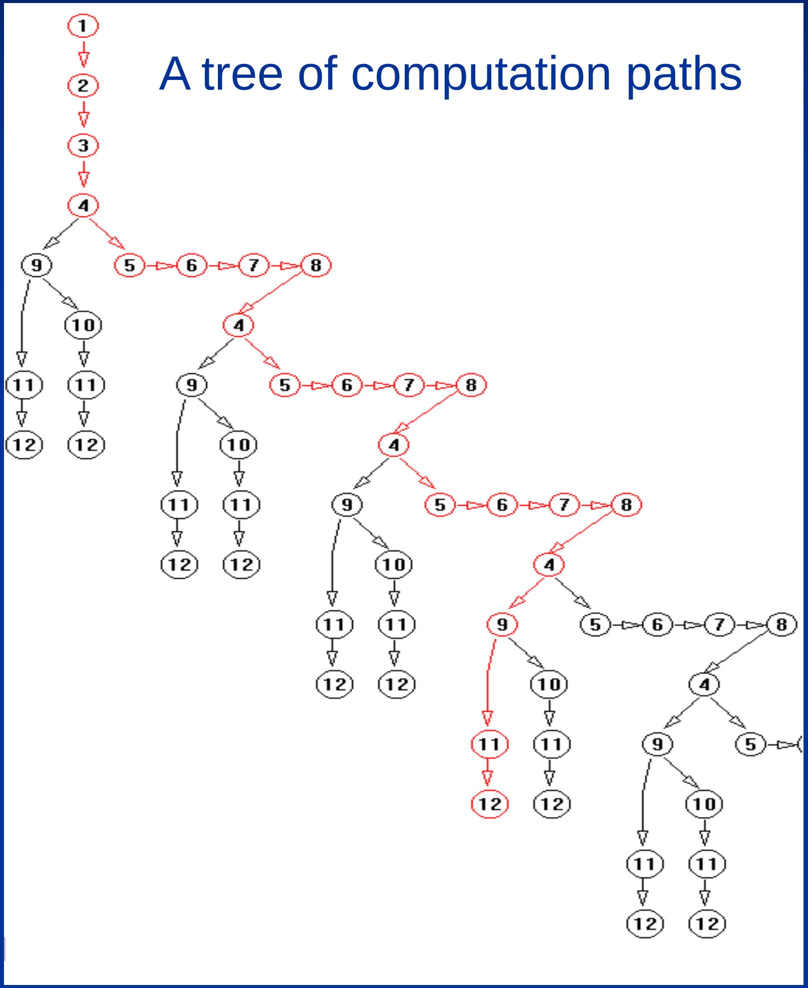
$s = d_1 \cdots d_{k-1} d_k$

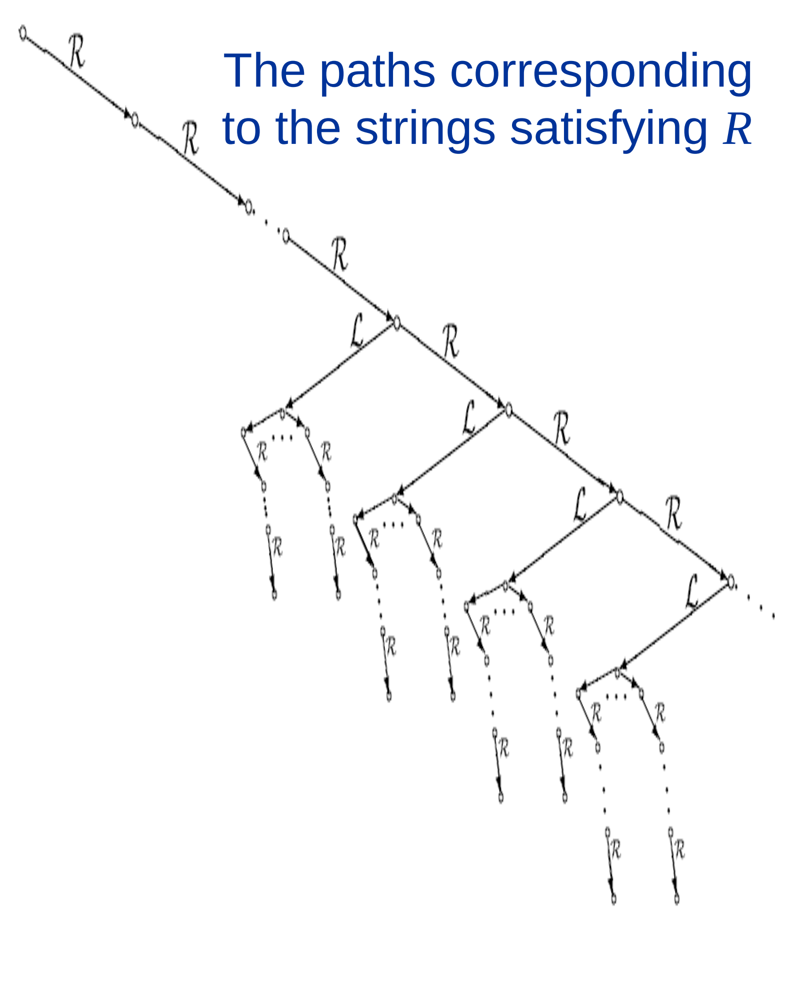
$add(s, d) = d_1 \cdots d_k d$



gassnerc@uni-greifswald.de

Similar trees

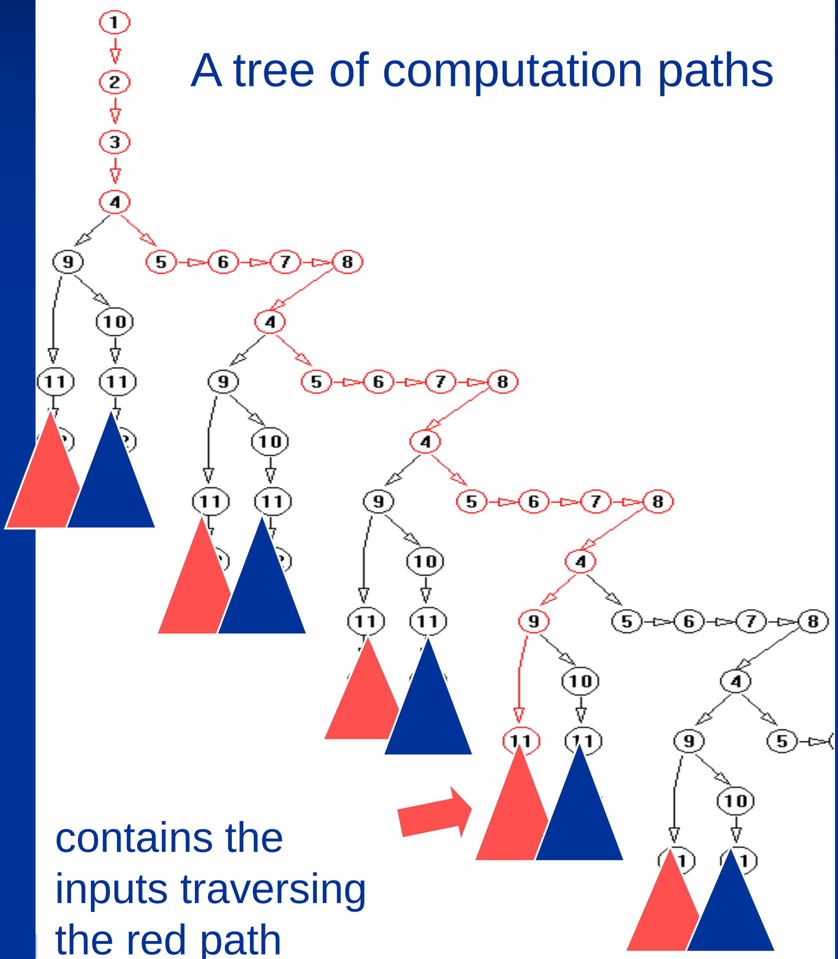




gassnerc@uni-greifswald.de

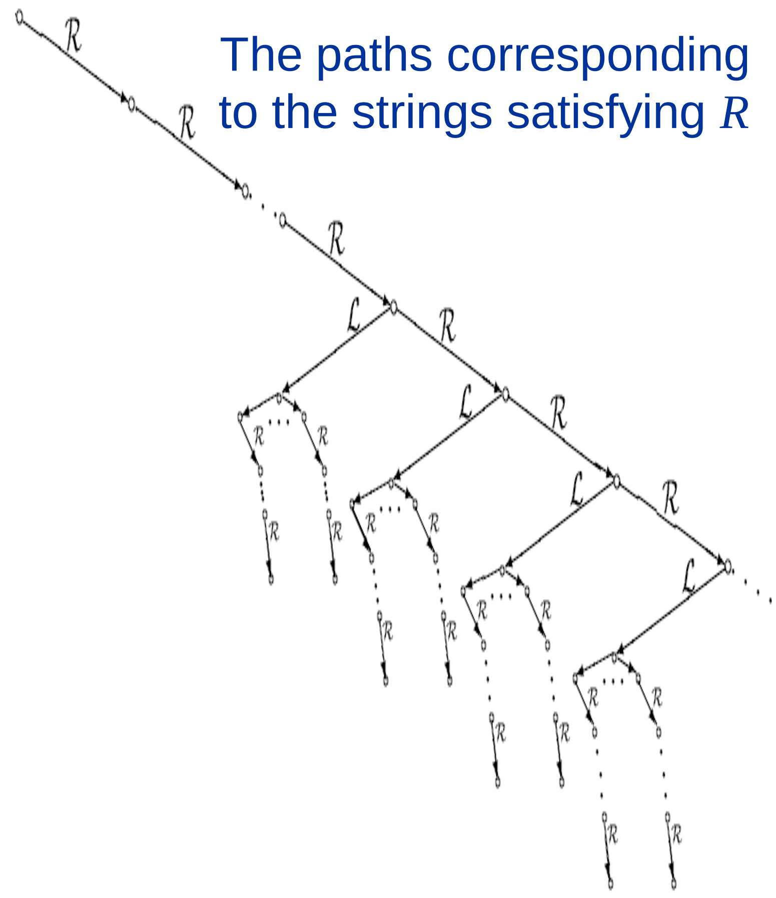
Similar trees

A tree of computation paths



contains the inputs traversing the red path

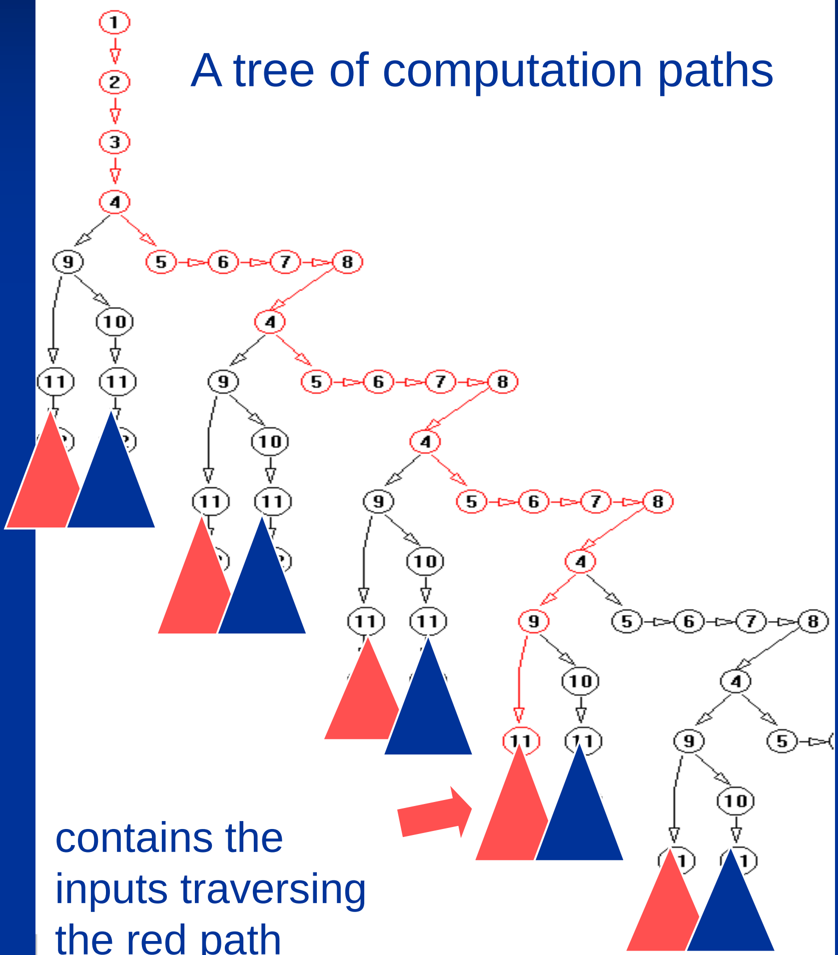
The paths corresponding to the strings satisfying R



gassnerc@uni-greifswald.de

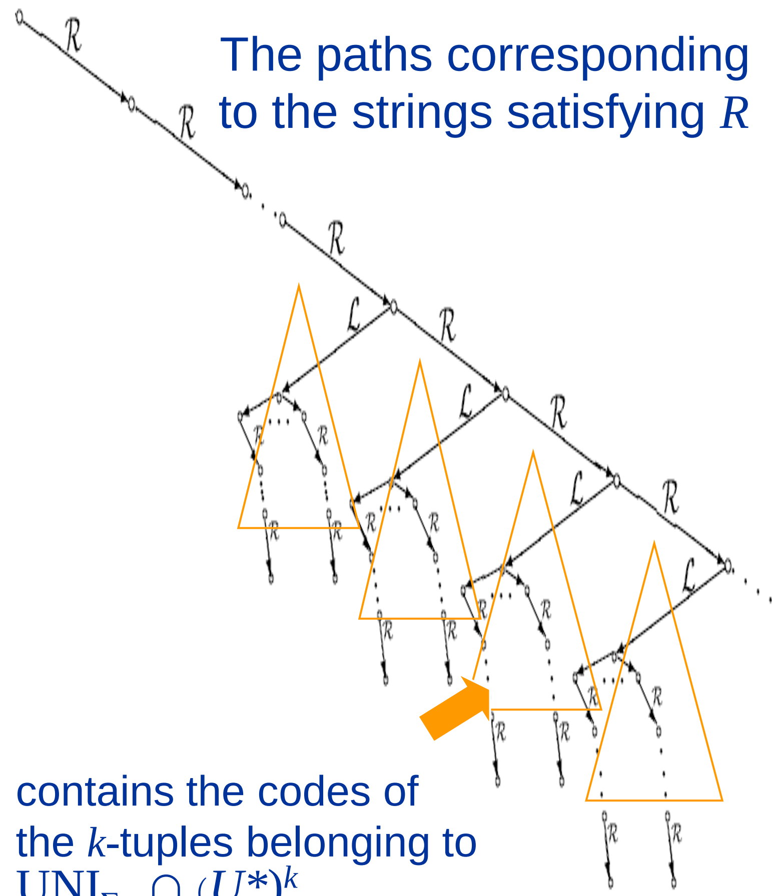
Similar trees

A tree of computation paths



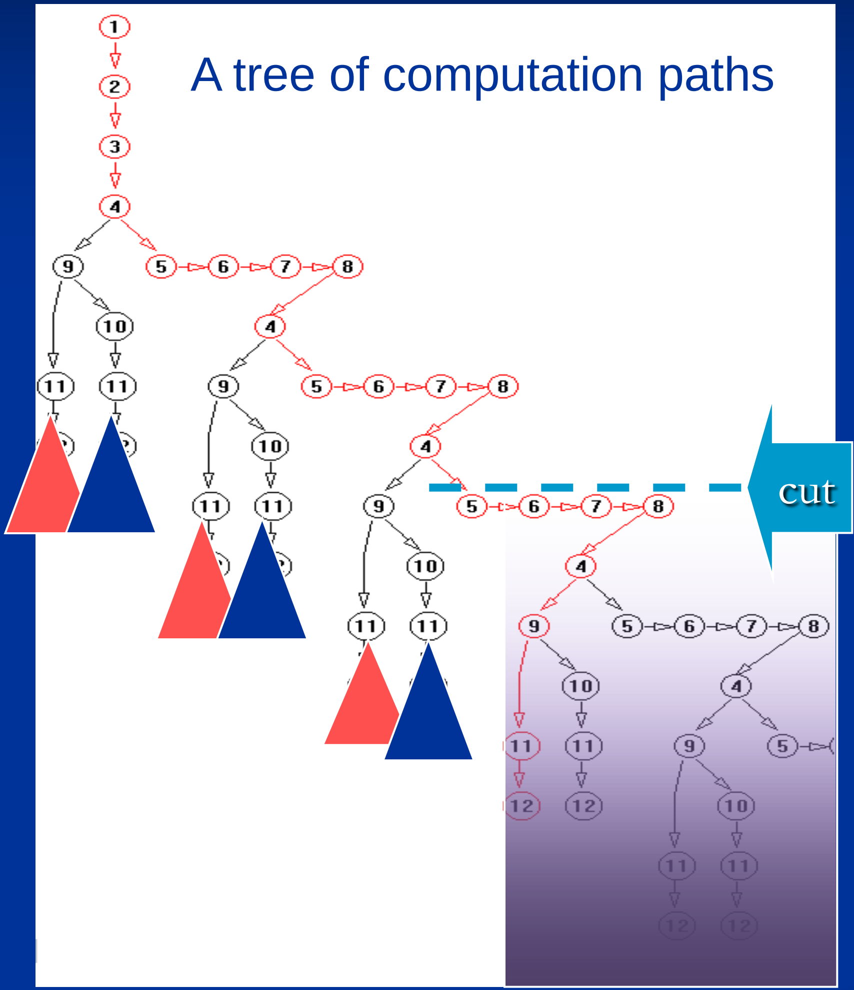
contains the inputs traversing the red path

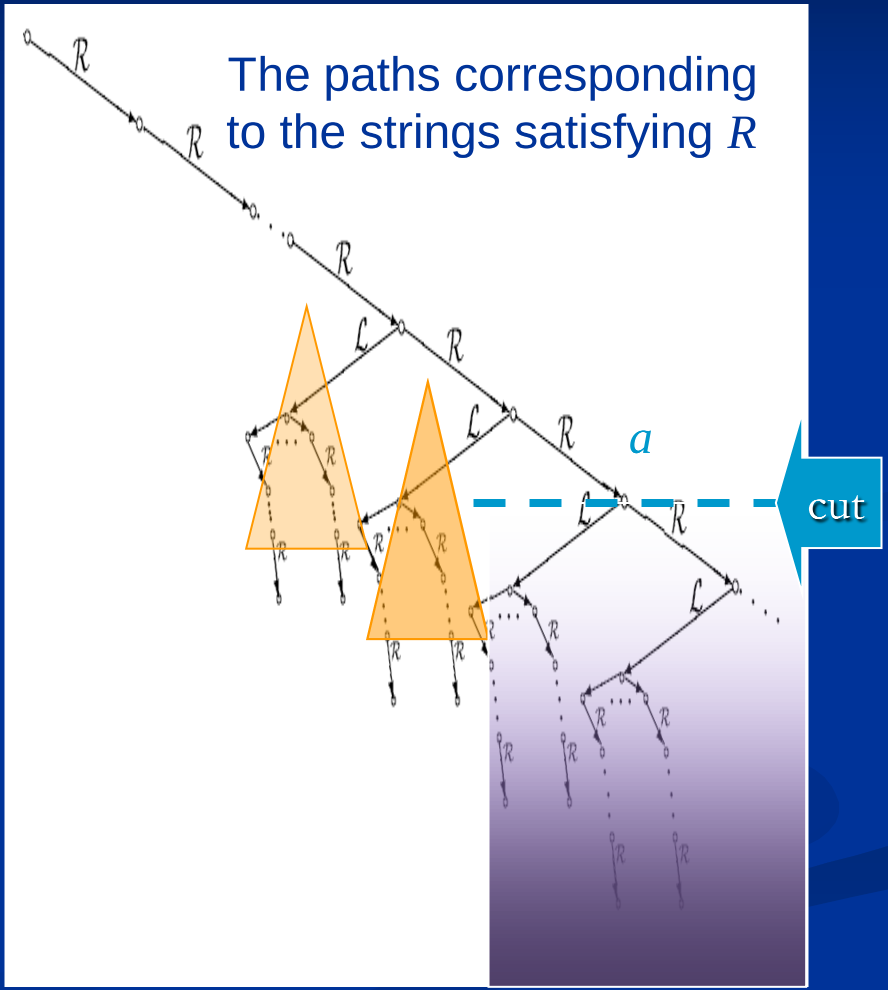
The paths corresponding to the strings satisfying R



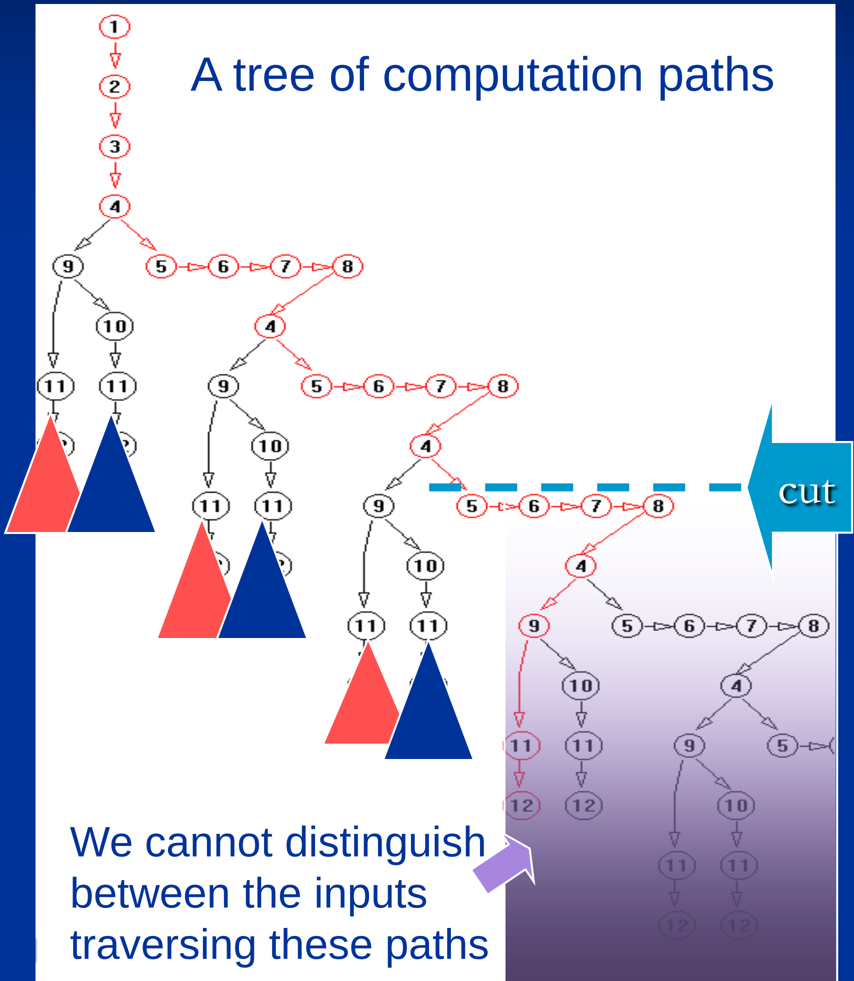
contains the codes of the k-tuples belonging to $\mathrm{UNI}_{\Sigma_R} \cap (U^*)^k$

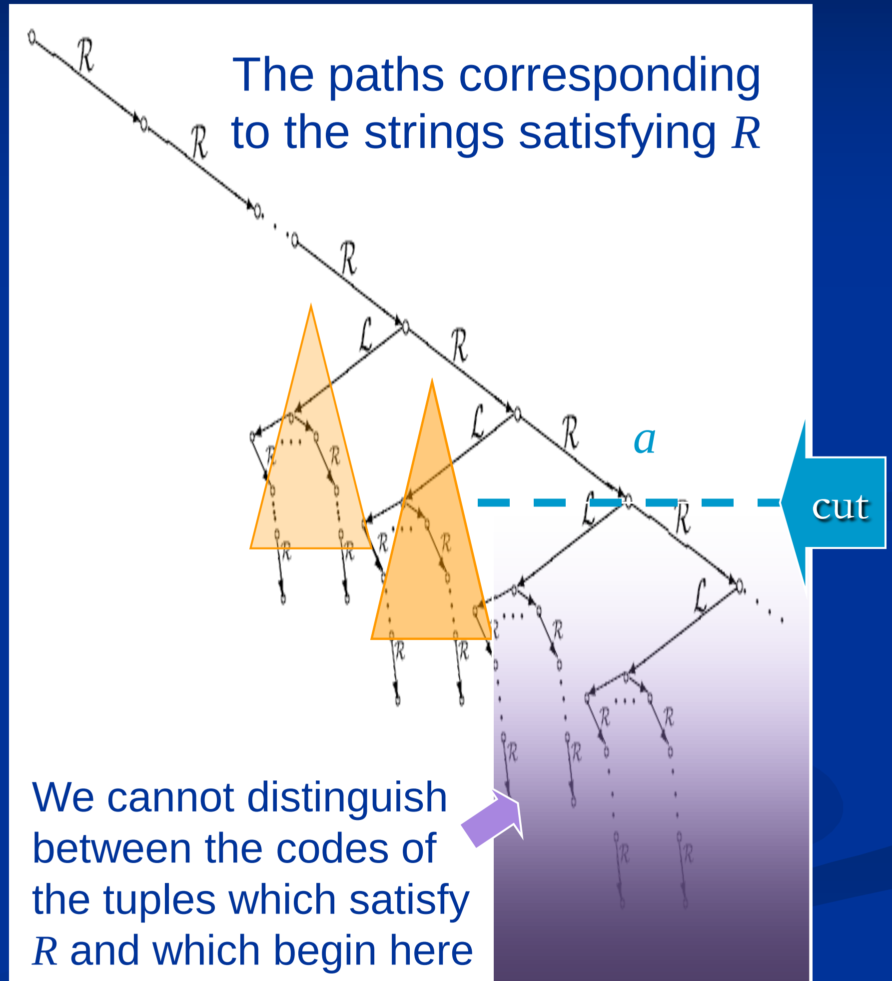
Trees and polynomial time





Trees and polynomial time





gassnerc@uni-greifswald.de

$P_{\Sigma_R} = NP_{\Sigma_R}$

Proof of $P_{\Sigma_R} = NP_{\Sigma_R}$ by a reduction.

UNI $=\{(b,\ldots, b, x_1,\ldots, x_n, Code(M)) \mid \mathbf{x} \in (U^*)^\infty \ \& \ M$ is NP_{Σ_R}-mach. $\& \ M(\mathbf{x})\downarrow^t\}$

UNI = RES-UNI (the length of guesses can be restricted)

- Decompose $\{x_1,\ldots, x_n\}$ into equivalence classes,
- replace $x_1,\ldots, x_n$ by suitable short strings

such that possible chains are not destroyed.

SUB-UNI (short input strings) SUB-UNI ⊂ RES-UNI

- Transform the input tuple into a string,
- double the length,
  check the new string by means of R.

Output: a / b

gassnerc@uni-greifswald.de

$P_{\Sigma_R} = NP_{\Sigma_R}$

Proof of $P_{\Sigma_R} = NP_{\Sigma_R}$ by a reduction.

UNI $=\{(b,\ldots, b, x_1,\ldots, x_n, Code(M)) \mid \boldsymbol{x} \in (U^*)^\infty$ & M is NP_{Σ_R}-mach. & $M(\boldsymbol{x})\downarrow^t\}$

UNI = RES-UNI (the length of guesses can be restricted)

- Decompose $\{x_1,\ldots, x_n\}$ into equivalence classes,
- replace $x_1,\ldots, x_n$ by suitable short strings

such that possible chains are not destroyed.

SUB-UNI (short input strings) SUB-UNI $\subset$ RES-UNI

- Transform the input tuple into a string,
- double the length,
  check the new string by means of R.

Output: a / b

gassnerc@uni-greifswald.de

$P_{\Sigma_R} = NP_{\Sigma_R}$

Proof of $P_{\Sigma_R} = NP_{\Sigma_R}$ by a reduction.

UNI $=\{(b,\ldots, b, x_1,\ldots, x_n, Code(M)) \mid \boldsymbol{x} \in (U^*)^\infty$ & M is NP_{Σ_R}-mach. & $M(\boldsymbol{x})\downarrow^t\}$

UNI = RES-UNI (the length of guesses can be restricted)

- Decompose $\{x_1,\ldots, x_n\}$ into equivalence classes,
- replace $x_1,\ldots, x_n$ by suitable short strings

such that possible chains are not destroyed.

SUB-UNI (short input strings) SUB-UNI ⊂ RES-UNI

- Transform the input tuple into a string,
- double the length,
  check the new string by means of R.

Output: a / b

gassnerc@uni-greifswald.de

$P_{\Sigma_R} = NP_{\Sigma_R}$

Proof of $P_{\Sigma_R} = NP_{\Sigma_R}$ by a reduction.

UNI $=\{(b,\ldots, b, x_1,\ldots, x_n, Code(M)) \mid \boldsymbol{x} \in (U^*)^\infty$ & M is NP_{Σ_R}-mach. & $M(\boldsymbol{x})\downarrow^t\}$

UNI = RES-UNI (the length of guesses can be restricted)

- Decompose $\{x_1,\ldots, x_n\}$ into equivalence classes,
- replace $x_1,\ldots, x_n$ by suitable short strings

such that possible chains are not destroyed.

SUB-UNI (short input strings) SUB-UNI $\subset$ RES-UNI

- Transform the input tuple into a string,
- double the length,
- check the new string by means of R.

Output: a / b

Why do we use strings?

Our goal:

- A relation R allows to decide whether $\boldsymbol{z} \in \mathcal{O}_\Sigma$.
- R can be defined recursively.

Problems:

- Each relation has a fixed arity.
- $\mathcal{O}_\Sigma$ contains tuples of any length.

- For many structures:
  The tuples of arbitrary length cannot be encoded by tuples of fixed length.

A solution:

- Strings.

Why do we use strings?

Our goal:

- A relation R allows to decide whether $\boldsymbol{z} \in O_\Sigma$.
- R can be defined recursively.

Problems:

- Each relation has a fixed arity.
- O_Σ contains tuples of any length.

- For many structures:
  The tuples of arbitrary length cannot be encoded by tuples of fixed length.

A solution:

- Strings.

Why do we use strings?

Our goal:

- A relation R allows to decide whether $\boldsymbol{z} \in O_{\Sigma}$.
- R can be defined recursively.

Problems:

- Each relation has a fixed arity.
- O_{Σ} contains tuples of any length.

- For many structures:
  The tuples of arbitrary length cannot be encoded by tuples of fixed length.

A solution:

- Strings.

Why do we use strings?
Appendix

Our goal:

- A relation R allows to decide whether $\boldsymbol{z} \in O_\Sigma$.
- R can be defined recursively.

Problems:

- Each relation has a fixed arity.
- O_Σ contains tuples of any length.

- For many structures:
  The tuples of arbitrary length cannot be encoded by tuples of fixed length.

A solution:

- Strings.

Structures over strings

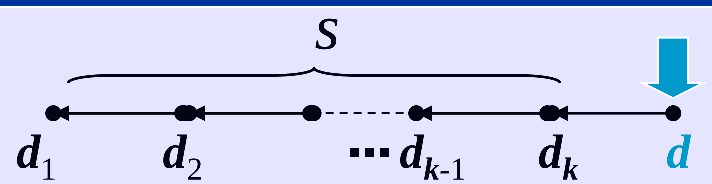
$$\Sigma = (U^* \,;\, \varepsilon, a, b, c_3, \ldots, c_u \,;\, add, sub_l, sub_r, f_1, \ldots, f_v \,;\, R_1, \ldots, R_w, R, =)$$

$s = d_1 \cdots d_k \in U^*$ stored in one register

$(d_1, \ldots, d_k) \in U^k \subset U^\infty$ stored in k registers

$add(s, d) = sd$ $\qquad sub_l(sd) = s$ $\qquad sub_r(sd) = d$

$s \in U^*, d \in U$



$s = d_1 \cdots d_{k-1} d_k$

$sd = d_1 \cdots d_k d$

$R_i \subseteq U^{n_i} \subseteq (U^*)^{n_i}$ and $R \subseteq U^*$

$f_i(s_1, \ldots, s_{m_i}) = \varepsilon$ if $|s_j| > 1$ for some j

gassnerc@uni-greifswald.de

Structures over strings

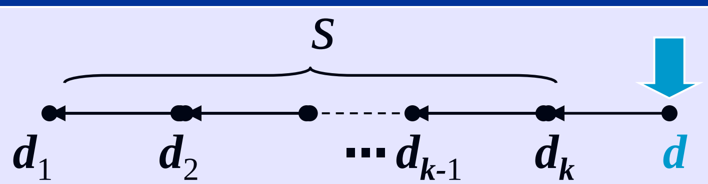
$$\Sigma = (U^* \,;\, \varepsilon, a, b, c_3,\ldots, c_u \,;\; add, sub_{\mathrm{l}}, sub_{\mathrm{r}}, f_1,\ldots, f_v \,;\, R_1,\ldots,R_w, R, =)$$

$s = d_1 \cdots d_k \in U^*$ stored in one register

$(d_1,\ldots, d_k) \in U^k \subset U^\infty$ stored in k registers

$add(s, d) = sd$ $\qquad sub_{\mathrm{l}}(sd) = s$ $\qquad sub_{\mathrm{r}}(sd) = d$

$s \in U^*, d \in U$



$s = d_1 \cdots d_{k-1} d_k$

$sd = d_1 \cdots d_k d$

$R_i \subseteq U^{n_i} \subseteq (U^*)^{n_i}$ and $R \subseteq U^*$

$f_i(s_1, \ldots, s_{m_i}) = \varepsilon$ if $|s_j| > 1$ for some j

Structures over strings

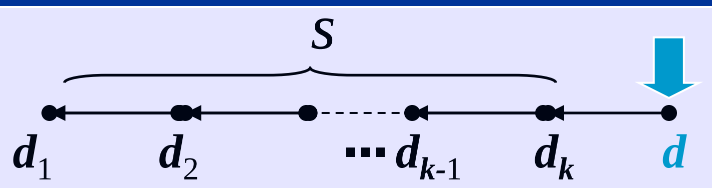
$\Sigma = (U^* \,;\, \varepsilon, a, b, c_3, \ldots, c_u \,;\; add, sub_{\mathrm{l}}, sub_{\mathrm{r}}, f_1, \ldots, f_v \,;\, R_1, \ldots, R_w, R, =)$

$s = d_1 \cdots d_k \in U^*$ stored in one register

$(d_1, \ldots, d_k) \in U^k \subset U^\infty$ stored in k registers

$add(s, d) = sd$ $\qquad sub_{\mathrm{l}}(sd) = s$ $\qquad sub_{\mathrm{r}}(sd) = d$

$s \in U^*, d \in U$



$s = d_1 \cdots d_{k-1} d_k$

$sd = d_1 \cdots d_k d$

$R_i \subseteq U^{n_i} \subseteq (U^*)^{n_i}$ and $R \subseteq U^*$

$f_i(s_1, \ldots, s_{m_i}) = \varepsilon$ if $|s_j| > 1$ for some j

gassnerc@uni-greifswald.de

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$

Definition: $s \subset_1 r \iff sub_l(r) = s$.

Lemma. For t steps of a machine holds:

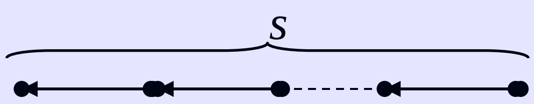
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

Corollary:

The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$



Definition: $s \subset_1 r \Leftrightarrow sub_l(r) = s$.

Lemma. For t steps of a machine holds:

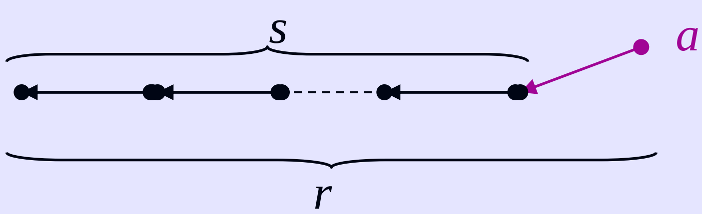
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

Corollary:

The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$



1) $r = sa$ $\quad add(s, a) = r \quad sub_l(r) = s$

Definition: $s \subset_1 r \iff sub_l(r) = s$.

Lemma. For t steps of a machine holds:

1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

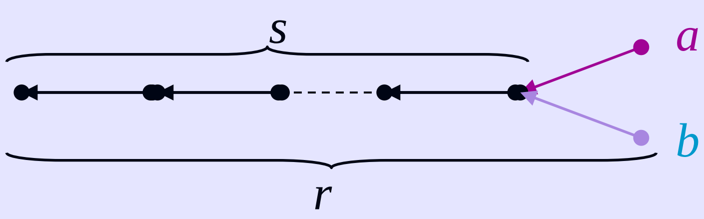
Corollary:

The minimal elements can be replaced without changing the computation path.

gassnerc@uni-greifswald.de

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$



1) $r = sa$ $\quad add(s, a) = r$ $\quad sub_l(r) = s$

2) $r = sb$ $\quad add(s, b) = r$ $\quad sub_l(r) = s$

Definition: $s \subset_1 r \Leftrightarrow sub_l(r) = s$.

Lemma. For t steps of a machine holds:

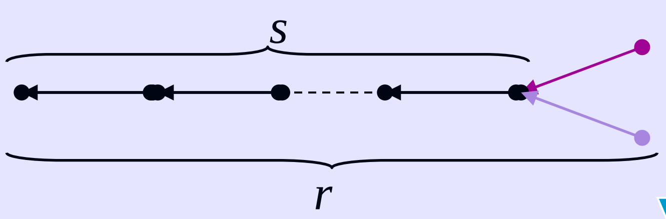
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

Corollary:

The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$



1) $r = sa$ $add(s, a) = r$ $sub_l(r) = s$

2) $r = sb$ $add(s, b) = r$ $sub_l(r) = s$

Definition: $s \subset_1 r \Leftrightarrow sub_l(r) = s.$

Lemma. For t steps of a machine holds:

1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

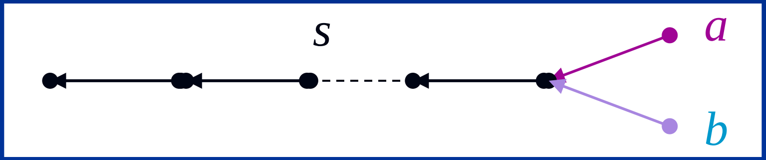
Corollary:

The minimal elements can be replaced without changing the computation path.

gassnerc@uni-greifswald.de

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$



Definition: $s \subset_1 r \iff sub_l(r) = s.$

Lemma. For t steps of a machine holds:

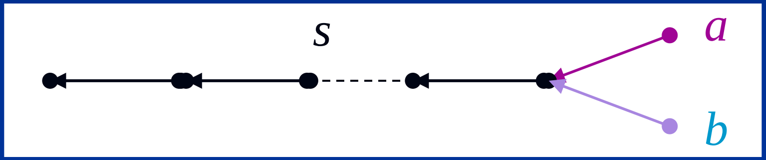
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

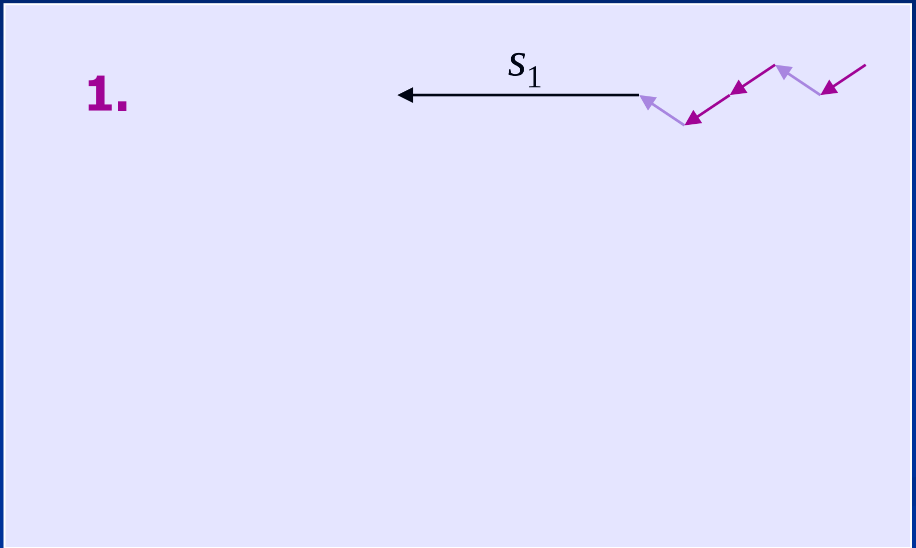
Corollary:

The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$





Definition: $s \subset_1 r \Leftrightarrow sub_l(r) = s.$

Lemma. For t steps of a machine holds:

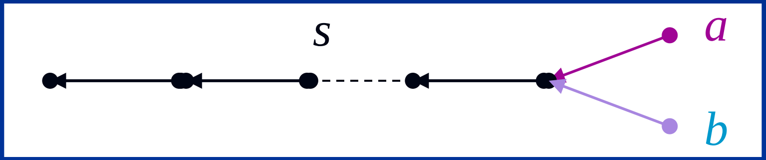
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

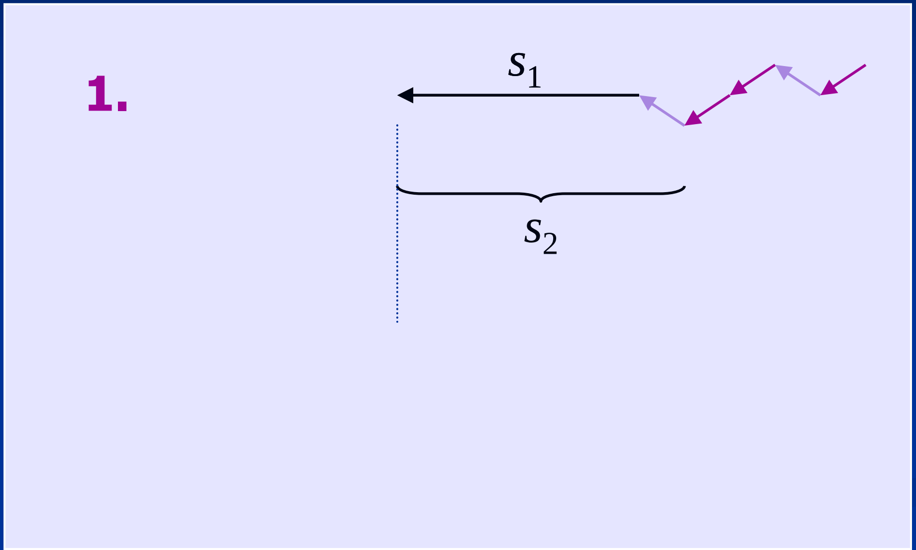
Corollary:

The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$





Definition: $s \subset_1 r \Leftrightarrow sub_l(r) = s.$

Lemma. For t steps of a machine holds:

1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

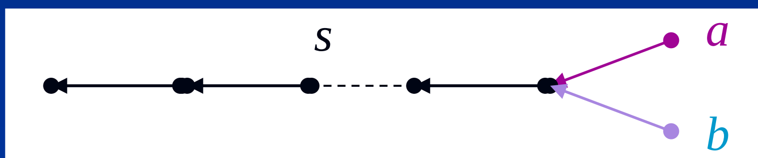
Corollary:

The minimal elements can be replaced without changing the computation path.

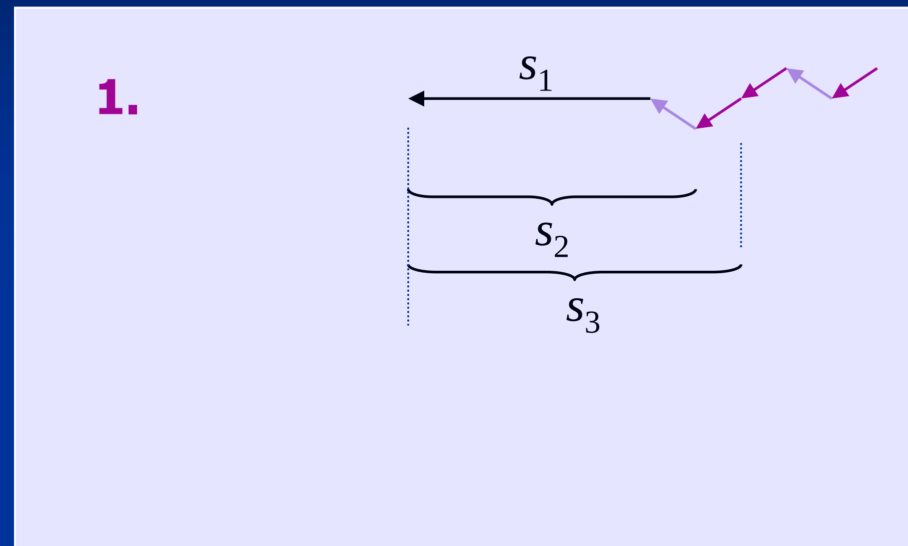
gassnerc@uni-greifswald.de

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$



Definition: $s \subset_1 r \Leftrightarrow sub_l(r) = s.$



Lemma. For t steps of a machine holds:

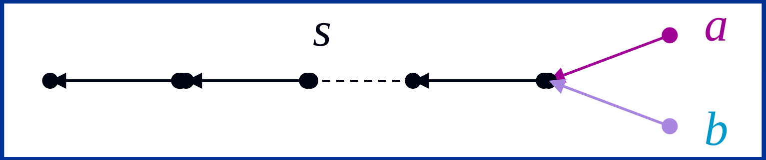
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

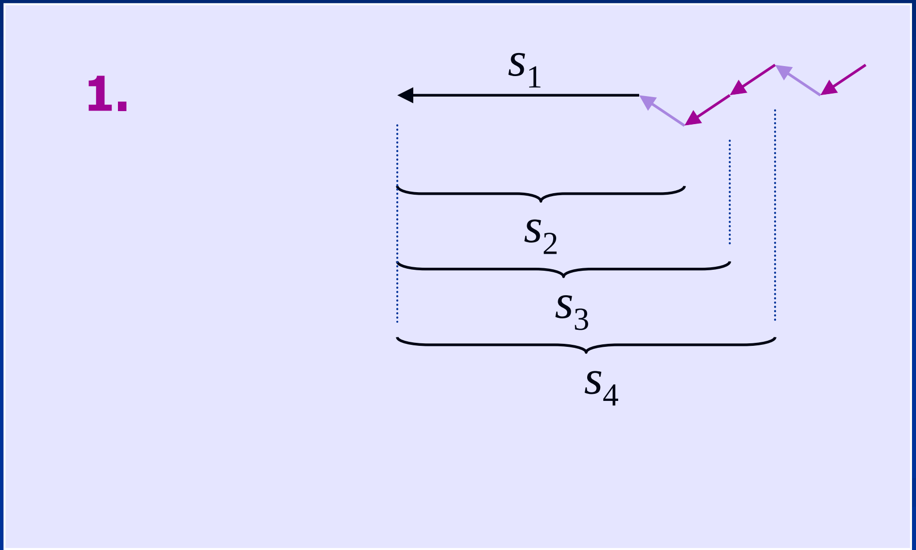
Corollary:

The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$





Definition: $s \subset_1 r \Leftrightarrow sub_l(r) = s$.

Lemma. For t steps of a machine holds:

1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

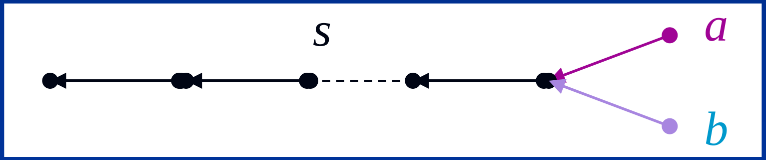
Corollary:

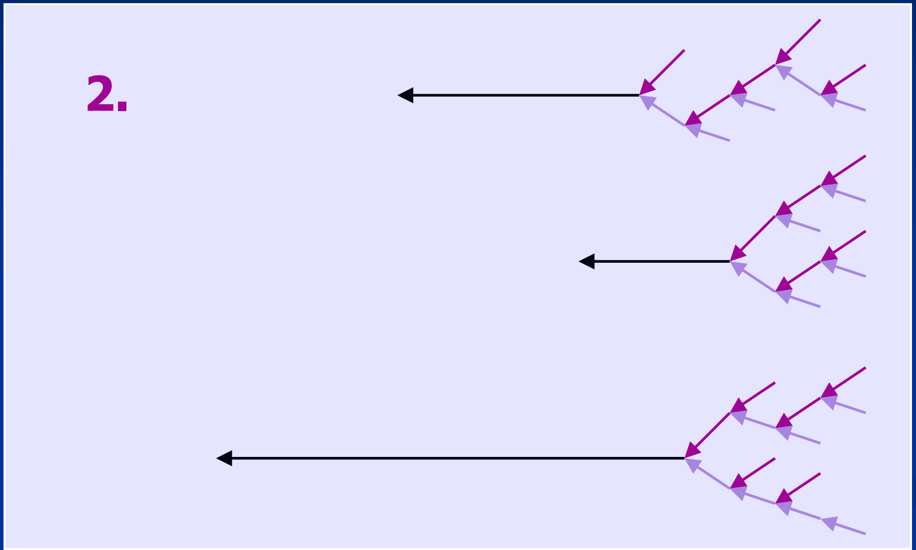
The minimal elements can be replaced without changing the computation path.

gassnerc@uni-greifswald.de

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$





Definition: $s \subset_1 r \Leftrightarrow sub_l(r) = s.$

Lemma. For t steps of a machine holds:

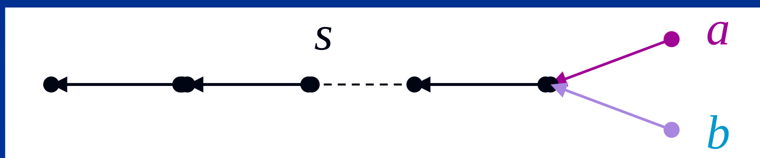
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

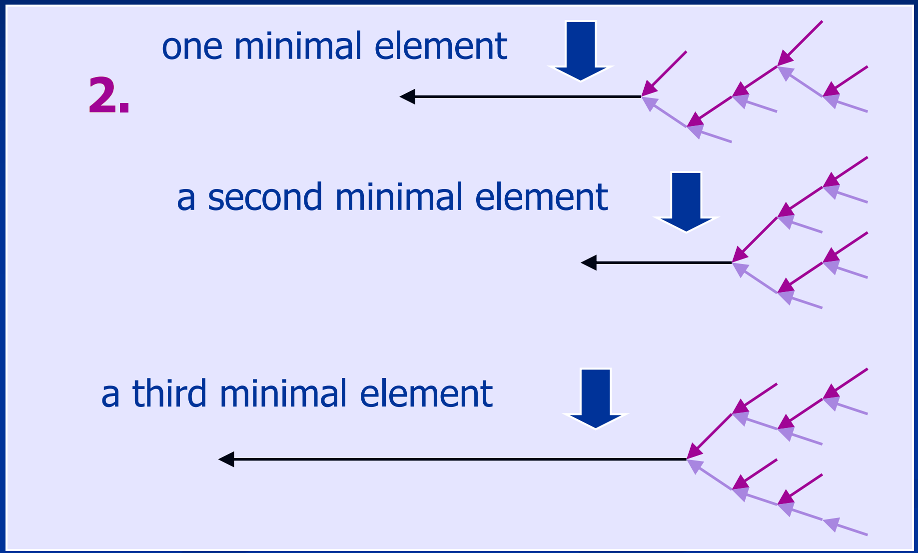
Corollary:

The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$





Definition: $s \subset_1 r \iff sub_l(r) = s.$

Lemma. For t steps of a machine holds:

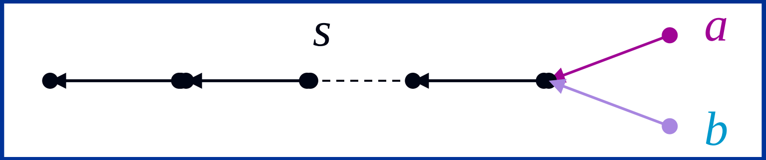
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

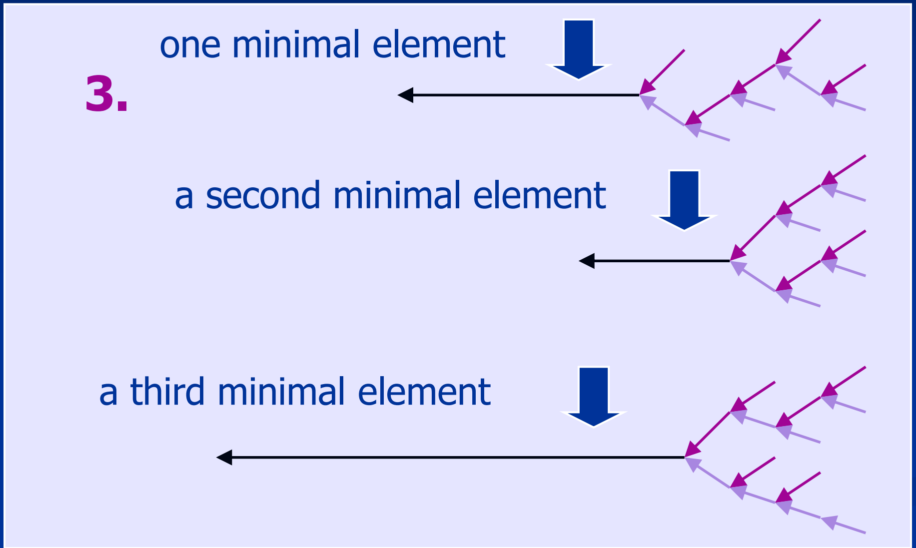
Corollary:

The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_l; =)$





Definition: $s \subset_1 r \Leftrightarrow sub_l(r) = s.$

Lemma. For t steps of a machine holds:

1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

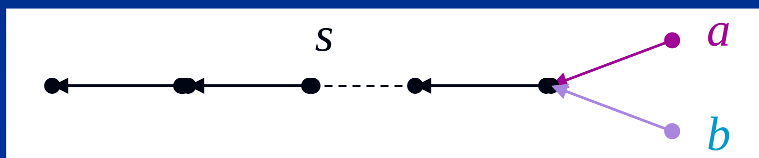
Corollary:

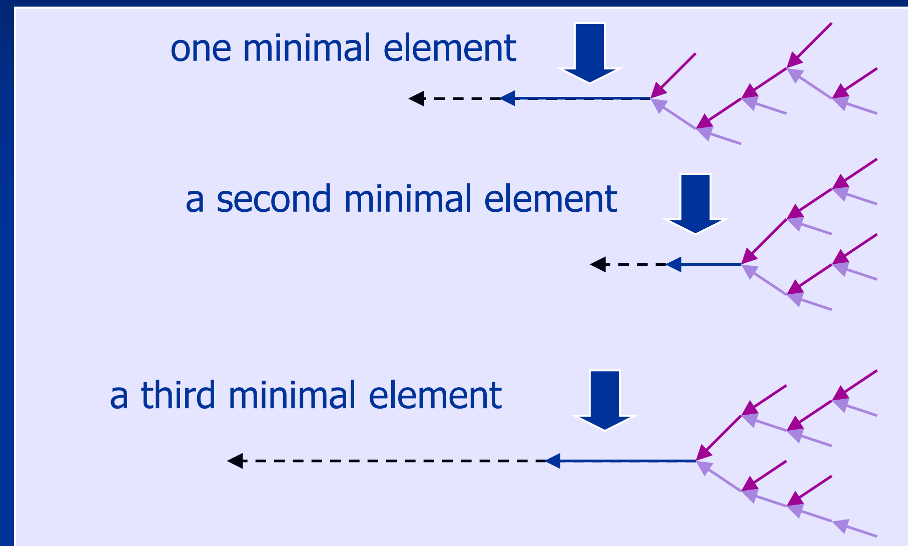
The minimal elements can be replaced without changing the computation path.

gassnerc@uni-greifswald.de

Computation over strings

Example: $\Sigma = (\{a, b\}^* \,;\, \varepsilon, a, b;\, add, sub_l;\, =)$





Definition: $s \subset_1 r \;\Leftrightarrow\; sub_l(r) = s.$

Lemma. For t steps of a machine holds:

1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

Corollary:

The minimal elements can be replaced without changing the computation path.

gassnerc@uni-greifswald.de

Why do we pad the codes?

Our goal:

- Structures Σ with $\mathrm{NP}_\Sigma \subseteq \mathrm{DEC}_\Sigma$.

Problems:

- Arbitrary strings can be guessed.
- A new R could imply $\mathrm{H}_{\Sigma_R} \in \mathrm{NP}_{\Sigma_R} \setminus \mathrm{DEC}_{\Sigma_R}$ for the halting problem H_{Σ_R} .

Solution:

- Padding strings:

$$R(s) \Rightarrow (\exists r \in U^*)\,(\, s = ra^{|r|}\,).$$

It allows to replace

- long inputs and guesses
- by short strings over $\{a, b\}$.

Why do we pad the codes?

Our goal:

- Structures Σ with $\mathrm{NP}_\Sigma \subseteq \mathrm{DEC}_\Sigma$.

Problems:

- Arbitrary strings can be guessed.
- A new R could imply $\mathrm{H}_{\Sigma_R} \in \mathrm{NP}_{\Sigma_R} \setminus \mathrm{DEC}_{\Sigma_R}$ for the halting problem H_{Σ_R}.

Solution:

- Padding strings:

$$R(s) \Rightarrow (\exists r \in U^*)(\ s = ra^{|r|}\ ).$$

It allows to replace

- long inputs and guesses
- by short strings over $\{a, b\}$.

gassnerc@uni-greifswald.de

Why do we pad the codes?

Our goal:

- Structures Σ with $NP_\Sigma \subseteq DEC_\Sigma$.

Problems:

- Arbitrary strings can be guessed.
- A new R could imply $H_{\Sigma_R} \in NP_{\Sigma_R} \setminus DEC_{\Sigma_R}$ for the halting problem H_{Σ_R}.

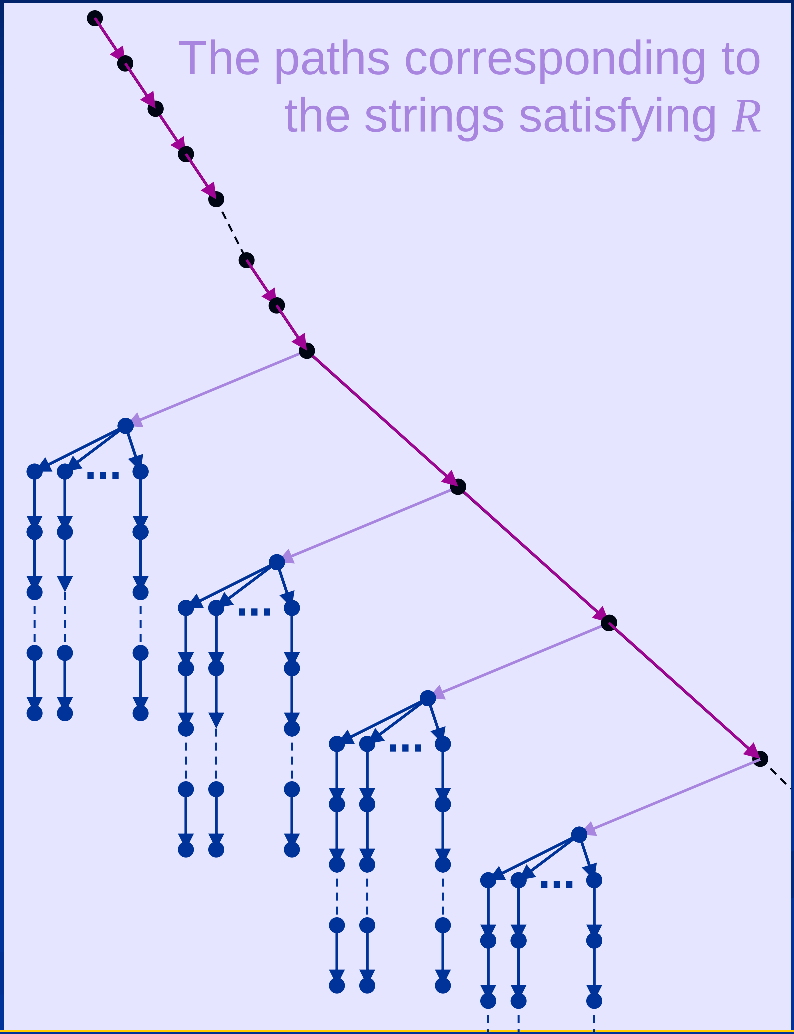
Solution:

- Padding strings:

$$R(s) \Rightarrow (\exists r \in U^*)(\, s = ra^{|r|}\,).$$

It allows to replace

- long inputs and guesses
- by short strings over $\{a, b\}$.



Why do we pad the codes?

Our goal:

- Structures Σ with $NP_\Sigma \subseteq DEC_\Sigma$.

Problems:

- Arbitrary strings can be guessed.
- A new R could imply $H_{\Sigma_R} \in NP_{\Sigma_R} \setminus DEC_{\Sigma_R}$ for the halting problem H_{Σ_R}.

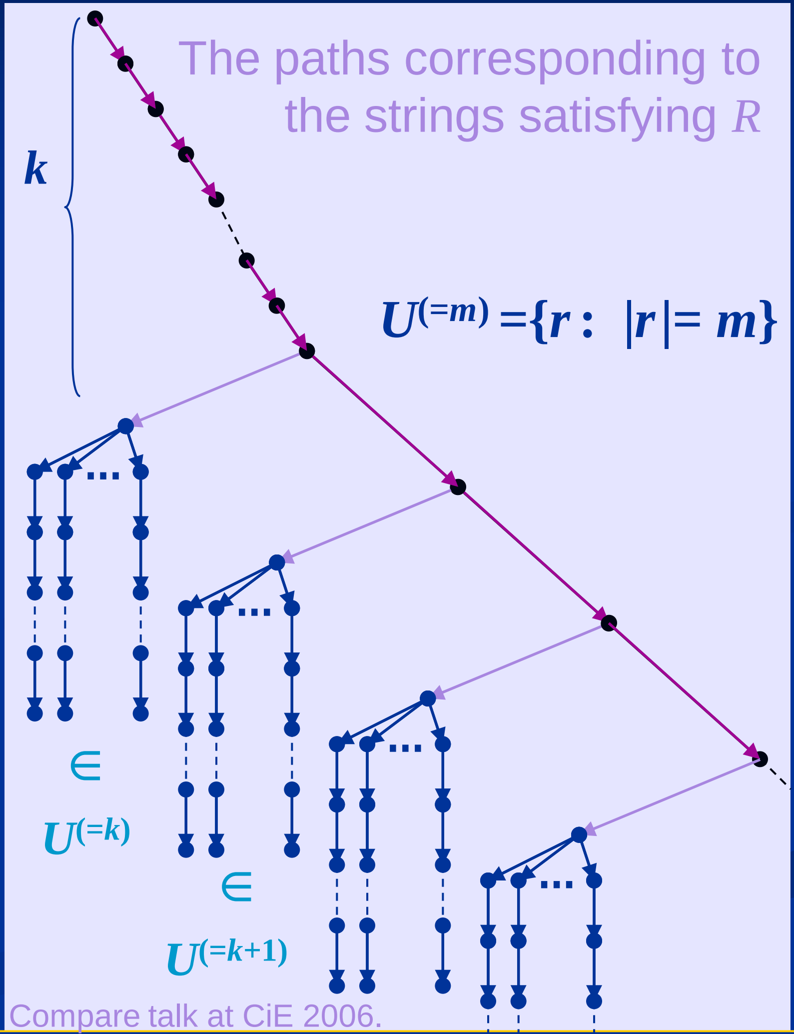
Solution:

- Padding strings:

$$R(s) \Rightarrow (\exists r \in U^*)(\, s = ra^{|r|}\,).$$

It allows to replace

- long inputs and guesses
- by short strings over $\{a, b\}$.

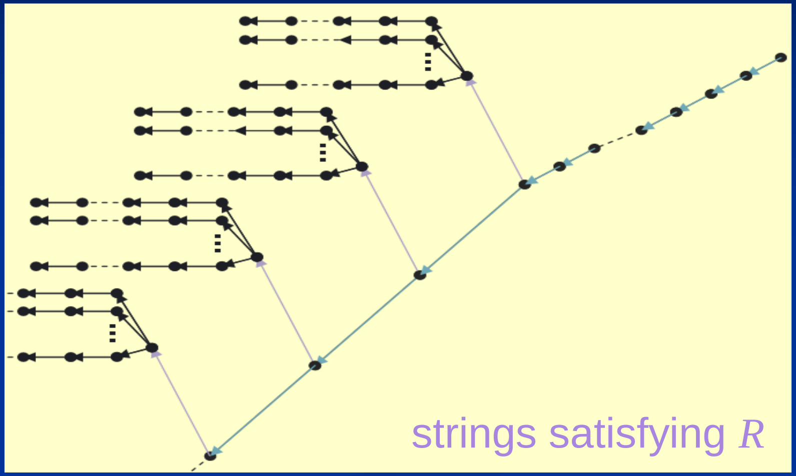


Compare talk at CiE 2006.

Why do we pad the codes?

Our goal:

- Structures Σ with $\mathrm{NP}_{\Sigma} \subseteq \mathrm{DEC}_{\Sigma}$.



Problems:

- Arbitrary strings can be guessed.
- A new R could imply $\mathrm{H}_{\Sigma_R} \in \mathrm{NP}_{\Sigma_R} \setminus \mathrm{DEC}_{\Sigma_R}$ for the halting problem H_{Σ_R}.

Solution:

- Padding strings:

$$R(s) \Rightarrow (\exists r \in U^*)(\, s = r a^{|r|}\,).$$

It allows to replace

- long inputs and guesses
- by short strings over $\{a, b\}$.

gassnerc@uni-greifswald.de

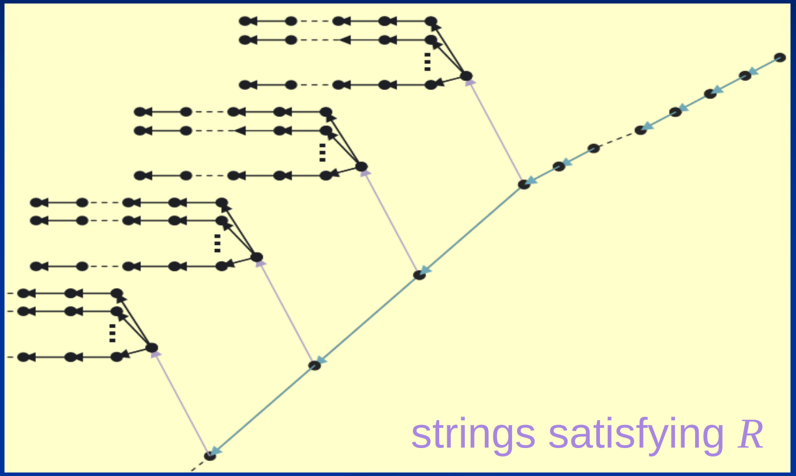
Why do we pad the codes?

Our goal:

- Structures Σ with $NP_\Sigma \subseteq DEC_\Sigma$.

Problems:

- Arbitrary strings can be guessed.
- A new R could imply $H_{\Sigma_R} \in NP_{\Sigma_R} \setminus DEC_{\Sigma_R}$ for the halting problem H_{Σ_R}.



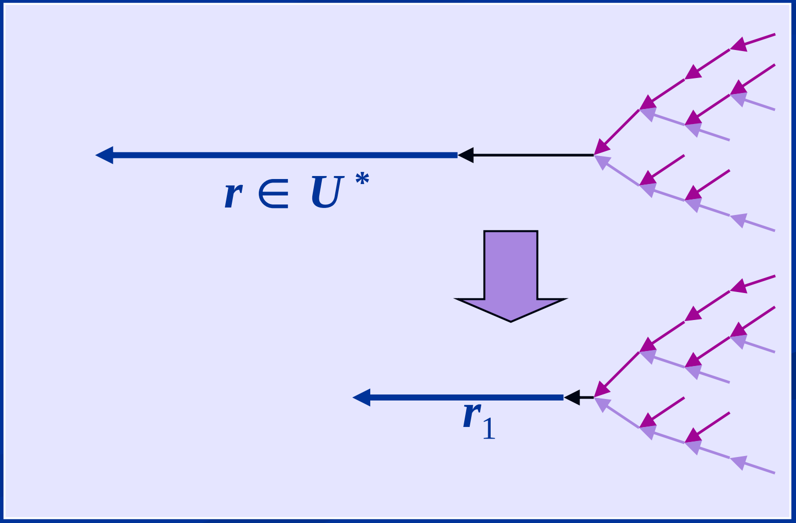
Solution:

- Padding strings:

$$R(s) \Rightarrow (\exists r \in U^*)(\, s = ra^{|r|}\,).$$

It allows to replace

- long inputs and guesses
- by short strings over $\{a, b\}$.



gassnerc@uni-greifswald.de

The new relation R

$\Sigma = (U^* ; a, b, c_3, \ldots, c_u ; f_1, \ldots, f_v ; R_1, \ldots, R_w, =)$
Σ_R = Expansion of Σ by R

A universal oracle:

Let $\mathcal{W}_\Sigma \subset (U^*)^\infty$ with $P_\Sigma^{\mathcal{W}_\Sigma} = NP_\Sigma^{\mathcal{W}_\Sigma}$ (derived from $\mathcal{O}_\Sigma$).

The relation R:

$$r_1 \cdots r_k a^{|r_1 \cdots r_k|} \in R \iff (r_1, \ldots, r_k) \in \mathcal{W}_\Sigma$$

Theorem:

$P_{\Sigma_R} = NP_{\Sigma_R}$.

gassnerc@uni-greifswald.de

The new relation R

$\mathbf{\Sigma} = (U^*\,;\, a, b, c_3, \ldots, c_u\,;\; f_1, \ldots, f_v\,;\; R_1, \ldots, R_w, =)$
$\mathbf{\Sigma}_{\boldsymbol{R}}$ = Expansion of $\mathbf{\Sigma}$ by R

A universal oracle:

$$\text{Let } W_\Sigma \subset (U^*)^\infty \text{ with } \mathrm{P}_\Sigma^{W_\Sigma} = \mathrm{NP}_\Sigma^{W_\Sigma}$$ (derived from O_Σ).

The relation R:

$$r_1 \cdots r_k\, a^{|r_1 \cdots r_k|} \in R \quad \Leftrightarrow \quad (r_1, \ldots, r_k) \in W_\Sigma$$

Theorem:

$$\mathrm{P}_{\Sigma_R} = \mathrm{NP}_{\Sigma_R}\,.$$

gassnerc@uni-greifswald.de

The new relation *R*

$\Sigma = (U^*; a, b, c_3, \ldots, c_u; f_1, \ldots, f_v; R_1, \ldots, R_w, =)$
Σ_R = Expansion of Σ by R

A universal oracle:

Let $\mathcal{W}_\Sigma \subset (U^*)^\infty$ with $P_\Sigma^{\mathcal{W}_\Sigma} = NP_\Sigma^{\mathcal{W}_\Sigma}$ (derived from $\mathcal{O}_\Sigma$).
(CiE 2007)

The relation *R*:

$$r_1 \cdots r_k a^{|r_1 \cdots r_k|} \in R \iff (r_1, \ldots, r_k) \in \mathcal{W}_\Sigma$$

Theorem (CiE 2007):

$$P_{\Sigma_R} = NP_{\Sigma_R}.$$

Thank you very much!

Christine Gaßner

Vielen Dank auch

- für die Unterstützung bei der Vorbereitung von Präsentationen an

  Gerald van den Boogaart, Volkmar Liebscher, Rainer Schimming, Michael Schürmann u.v.a.

- für frühere Diskussionen an

  Mihai Prunescu, Günter Asser, Pascal Koiran (über M. P.), Dieter Spreen u.v.a.

gassnerc@uni-greifswald.de