P = NP for Expansions Derived from Some Oracles

Christine Gaßner
Greifswald

P = NP for Expansions Derived from Some Oracles

Our claim:

Any structure can be

- extended to a structure of strings and
  - expanded by a relation R

to

a structure with P = NP.

P = NP for Expansions Derived from Some Oracles

1. The uniform model of computation
2. Oracles implying $P^A = NP^A$
3. Relations derived from oracles with $P = NP$
4. An ordered structure with $P = NP$
5. Summary

Structures (of finite signature):

$$\Sigma = (U\,;\ \underbrace{c_1,\ldots,c_u}_{\text{constants}}\,;\ \underbrace{f_1,\ldots,f_v}_{\text{operations}}\,;\ \underbrace{R_1,\ldots,R_w,=}_{\text{relations}})$$

Examples:

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

$\mathbb{R} = (\mathbb{R}; 0,1,\ldots; +, -, \cdot\ ; \leq)$ (⇨ BSS model)

$\Sigma_{\text{string}} = (U^*; \varepsilon\ a, b; add, sub_l, sub_r; =)$

$\Sigma_{\mathbb{N}} = (\mathbb{N}; 0, 1; shr, shl, inc \circ shl; R, \leq)$

Structures (of finite signature):

$$\Sigma = (U;\ \underbrace{c_1,\ldots,c_u}_{\text{constants}};\ \underbrace{f_1,\ldots,f_v}_{\text{operations}};\ \underbrace{R_1,\ldots,R_w}_{\text{relations}},=)$$

Examples:

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

$\mathbb{R} = (\mathbb{R}; 0,1,\ldots; +, -, \cdot\ ; \leq)$ (⇨ BSS model)

$\Sigma_{\text{string}} = (U^*; \varepsilon, a, b; add, sub_l, sub_r; =)$

$\Sigma_{\mathbb{N}} = (\mathbb{N}; 0, 1; shr, shl, inc \circ shl; R, \leq)$

The computation instructions

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

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

The computation instructions

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

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

The computation instructions

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

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 computation instructions

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

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 computation instructions

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

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 input:

$$(Z_1,\ldots,Z_n) := (x_1,\ldots,x_n) \in \cup_{i\geq 1} U^i$$

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

The input and guessing:

$$(Z_1,\ldots,Z_n,Z_{n+1},\ldots,Z_{n+m}) := (x_1,\ldots,x_n,\underbrace{y_1,\ldots,y_m}) \in \cup_{i\geq 1} U^i$$

Arbitrary elements can be guessed!

The input:

$$(Z_1,\ldots,Z_n) := (x_1,\ldots,x_n) \in \cup_{i\geq 1} U^i$$

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

The input and guessing:

$$(Z_1,\ldots,Z_n,Z_{n+1},\ldots,Z_{n+m}) := (x_1,\ldots,x_n,\underbrace{y_1,\ldots,y_m}) \in \cup_{i\geq 1} U^i$$

Arbitrary elements can be guessed!

The input:

$$(Z_1, \ldots, Z_n) := (x_1, \ldots, x_n) \in \cup_{i \geq 1} U^i$$

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

The input and guessing:

$$(Z_1, \ldots, Z_n, Z_{n+1}, \ldots, Z_{n+m}) := (x_1, \ldots, x_n, \underbrace{y_1, \ldots, y_m}) \in \cup_{i \geq 1} U^i$$

➡ Arbitrary elements can be guessed!

The input:

$$(Z_1, \ldots, Z_n) := (x_1, \ldots, x_n) \in \cup_{i \geq 1} U^i$$

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

The input and guessing:

$$(Z_1, \ldots, Z_n, Z_{n+1}, \ldots, Z_{n+m}) := (x_1, \ldots, x_n, \underbrace{y_1, \ldots, y_m}) \in \cup_{i \geq 1} U^i$$

Arbitrary elements can be guessed!

Computation in polynomial time:

For any machine M there is some polynomial p_M

such that for all $(x_1,\ldots,x_n)$

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

The execution of one operation = one time unit.

$P_\Sigma \subseteq NP_\Sigma$ $P_\Sigma \subseteq DEC_\Sigma$ ⇨ $NP_\Sigma \not\subseteq DEC_\Sigma \Rightarrow P_\Sigma \neq NP_\Sigma$

Oracle machines

Structure: $\Sigma = (U\,;\, a, b, c_3, \ldots, c_u\,;\; f_1, \ldots, f_v\,;\; R_1, \ldots, R_w, =)$

Oracle: $A \subseteq U^\infty =_{\mathrm{df}} \cup_{i \geq 1} U^i$

$\Downarrow$

Oracle query:

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

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

Proposition (Baker, Gill, and Solovay; Emerson; ...):

For any structure Σ, there is some oracle O_Σ such that $P_\Sigma^{O_\Sigma} = NP_\Sigma^{O_\Sigma}$.

Oracle machines

Structure: $\Sigma = (U\,;\, a, b, c_3, \ldots, c_u\,;\; f_1, \ldots, f_v\,;\; R_1, \ldots, R_w, =)$

Oracle: $A \subseteq U^\infty =_{\mathrm{df}} \cup_{i\geq 1} U^i$

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

The length can be computed by $I_1 := 1;\;\; I_1 := I_1 + 1; \ldots$

Proposition (Baker, Gill, and Solovay; Emerson; ...):

For any structure Σ, there is some oracle O_Σ such that $P_\Sigma^{O_\Sigma} = NP_\Sigma^{O_\Sigma}$.

The oracle O_Σ with $P_\Sigma^{O_\Sigma} = NP_\Sigma^{O_\Sigma}$

A universal oracle:

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

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

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

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

$\Rightarrow$ $P_\Sigma^{O_\Sigma} = NP_\Sigma^{O_\Sigma}$

Why do we use strings?

Our goal:

- A relation R allows to decide whether $z \in O_\Sigma$.
- 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 $z \in O_\Sigma$.
- 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 $z \in O_\Sigma$.
- 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 $z \in O_\Sigma$.
- 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

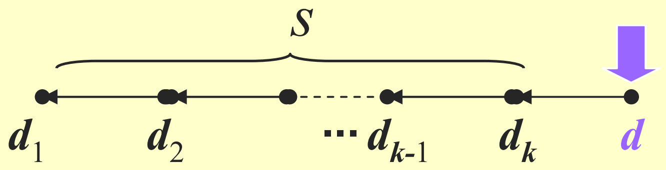
$$\Sigma = (U^* \, ; \varepsilon, a, b, c_3, \ldots, c_u \, ; \; add, sub_l, sub_r, f_1, \ldots, f_v \, ; R_1, \ldots, R_w, R, =)$$

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

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

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

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



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

$$sd = d_1 \cdots d_k d$$

$R_i \subseteq U^{n_i}$

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

Structures over strings

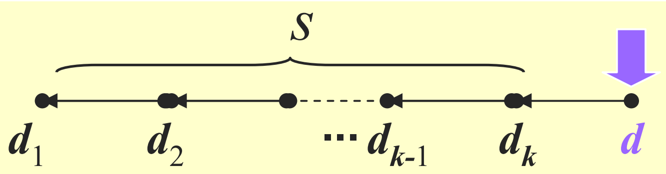
$$\Sigma = (U^* \,;\, \varepsilon, a, b, c_3, \ldots, c_u \,;\, add, sub_l, sub_r, f_1, \ldots, f_v \,;\, R_1, \ldots, R_w, R, =)$$

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

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

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

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



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

$sd = d_1 \cdots d_k d$

$R_i \subseteq U^{n_i}$

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

Structures over strings

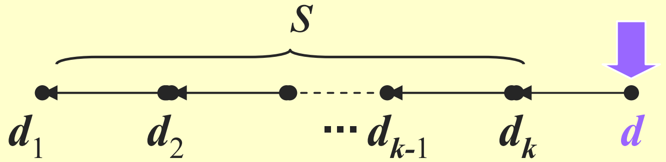
$$\Sigma = (U^* \,;\, \varepsilon, a, b, c_3, \ldots, c_u \,;\, add, sub_l, sub_r, f_1, \ldots, f_v \,;\, R_1, \ldots, R_w, R, =)$$

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

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

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

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



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

$sd = d_1 \cdots d_k d$

$R_i \subseteq U^{n_i}$

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

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$

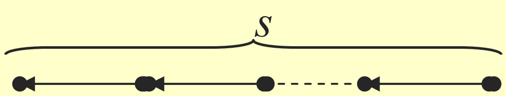
Definition: $s \subset_1 r \Leftrightarrow sub_1(r) = s.$

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.

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$



Definition: $s \subset_1 r \Leftrightarrow sub_1(r) = s.$

Lemma. For t steps of a machine holds:

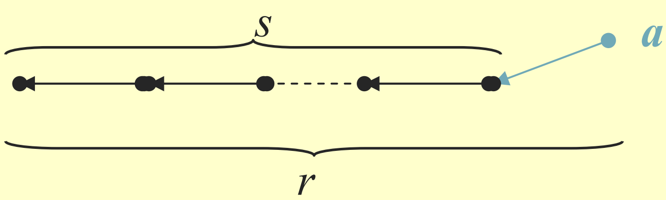
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

Corollary:

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

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$



1) $r = sa$ $\quad add(s, a) = r$ $\quad sub_1(r) = s$

Definition: $s \subset_1 r \Leftrightarrow sub_1(r) = s.$

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.

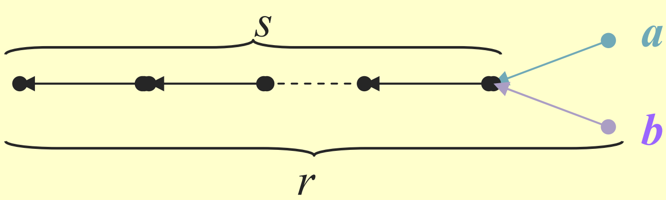
Corollary:

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

Christine Gaßner gassnerc@uni-greifswald.de

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$



1) $r = sa \quad add(s, a) = r \quad sub_1(r) = s$

2) $r = sb \quad add(s, b) = r \quad sub_1(r) = s$

Definition: $s \subset_1 r \Leftrightarrow sub_1(r) = s.$

Lemma. For t steps of a machine holds:

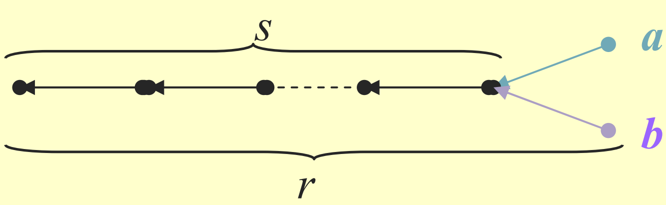
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

Corollary:

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

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$



1) $r = sa$ $\quad add(s, a) = r$ $\quad sub_1(r) = s$

2) $r = sb$ $\quad add(s, b) = r$ $\quad sub_1(r) = s$

Definition: $s \subset_1 r \;\Leftrightarrow\; sub_1(r) = s$.

Lemma. For t steps of a machine holds:

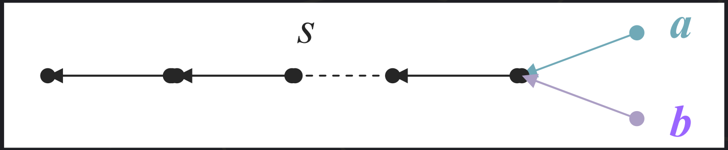
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

Corollary:

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

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$



Definition: $s \subset_1 r \Leftrightarrow sub_1(r) = s$.

Lemma. For t steps of a machine holds:

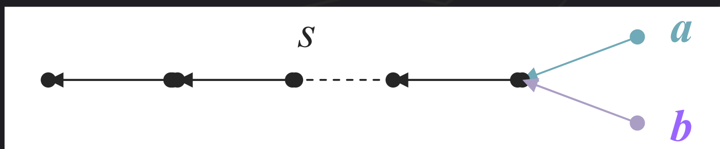
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

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

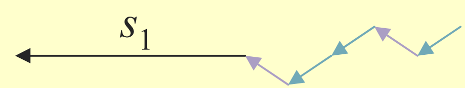
Christine Gaßner gassnerc@uni-greifswald.de

Computation over strings

Example: $\Sigma = (\{a, b\}^*\,; \varepsilon, a, b; add, sub_1; =)$



1.



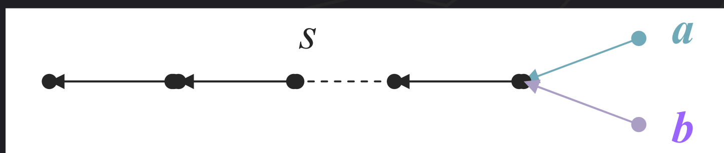
Definition: $s \subset_1 r \iff sub_1(r) = s.$

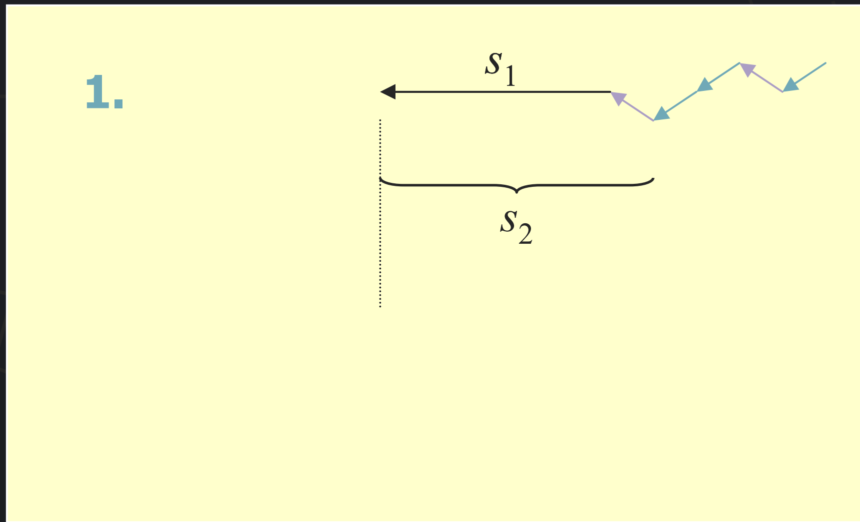
Lemma. For t steps of a machine holds:

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.

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$





Definition: $s \subset_1 r \Leftrightarrow sub_1(r) = s$.

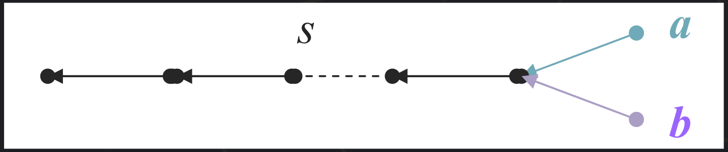
Lemma. For t steps of a machine holds:

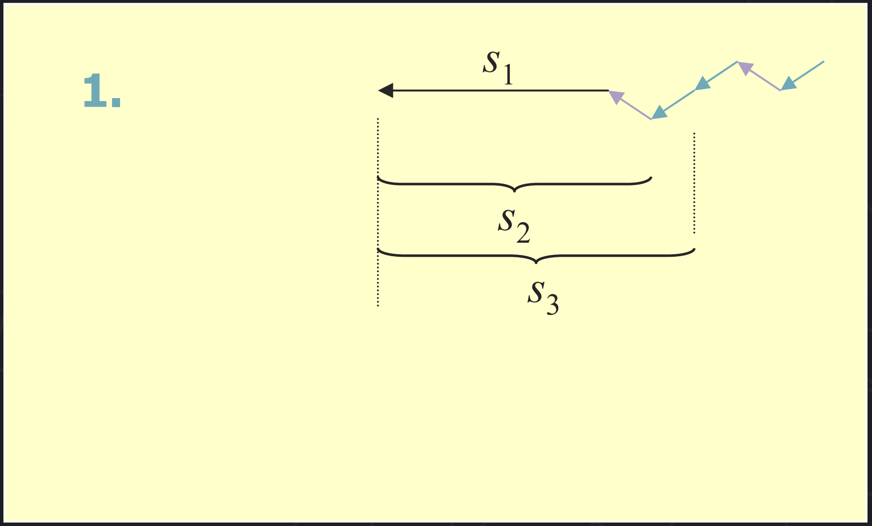
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

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

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$





Definition: $s \subset_1 r \iff sub_1(r) = s$.

Lemma. For t steps of a machine holds:

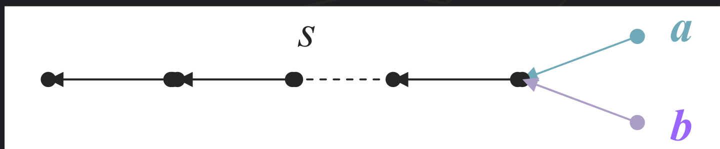
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

Corollary:

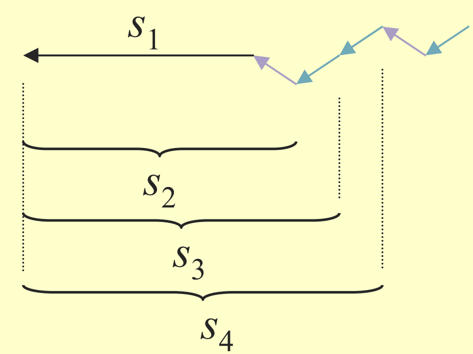
The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^*; \varepsilon, a, b; add, sub_1; =)$



1.



Definition: $s \subset_1 r \iff sub_1(r) = s$.

Lemma. For t steps of a machine holds:

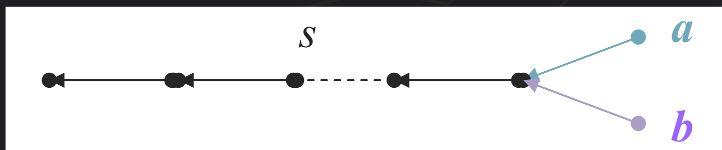
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

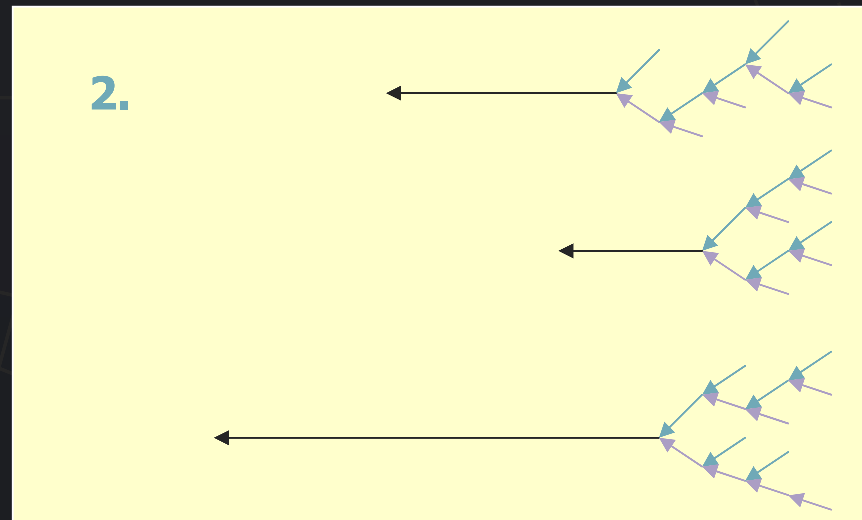
Corollary:

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

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$





Definition: $s \subset_1 r \Leftrightarrow sub_1(r) = s$.

Lemma. For t steps of a machine holds:

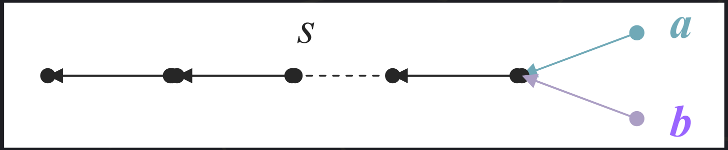
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

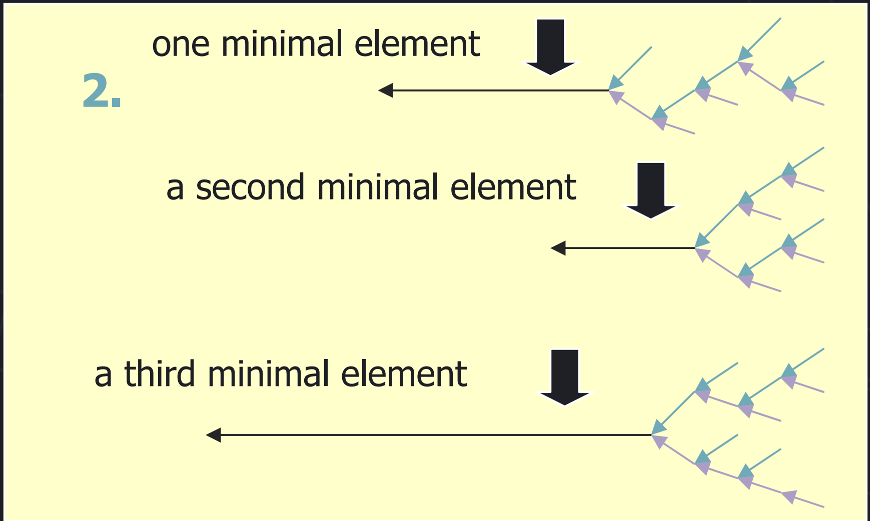
Corollary:

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

Computation over strings

Example: $\Sigma = (\{a, b\}^* ; \varepsilon, a, b; add, sub_1; =)$





Definition: $s \subset_1 r \;\Leftrightarrow\; sub_1(r) = s.$

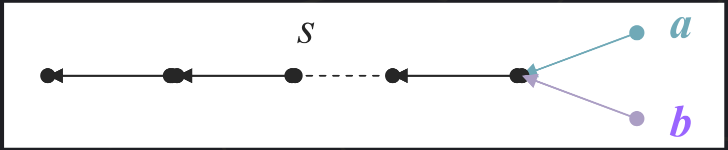
Lemma. For t steps of a machine holds:

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.

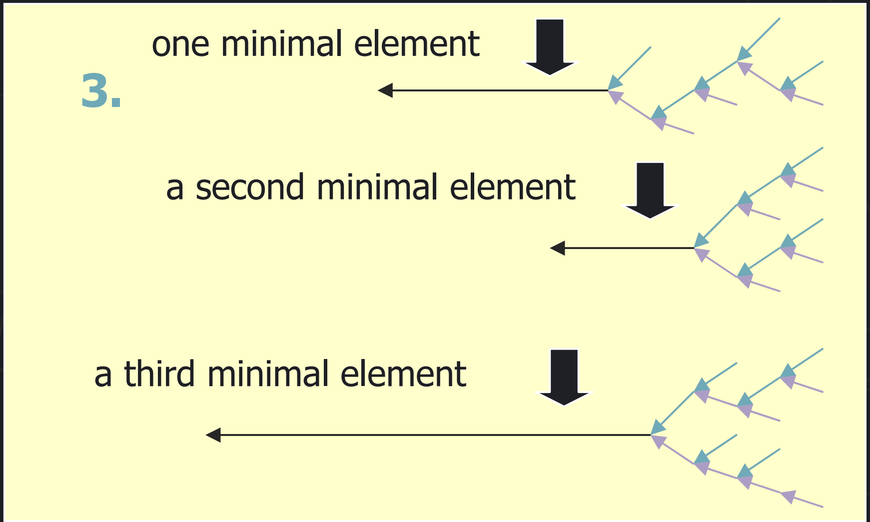
Corollary:
The minimal elements can be replaced without changing the computation path.

Computation over strings

Example: $\Sigma = (\{a, b\}^*; \varepsilon, a, b; add, sub_1; =)$



Definition: $s \subset_1 r \Leftrightarrow sub_1(r) = s.$



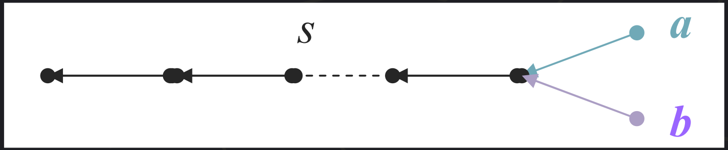
Lemma. For t steps of a machine holds:

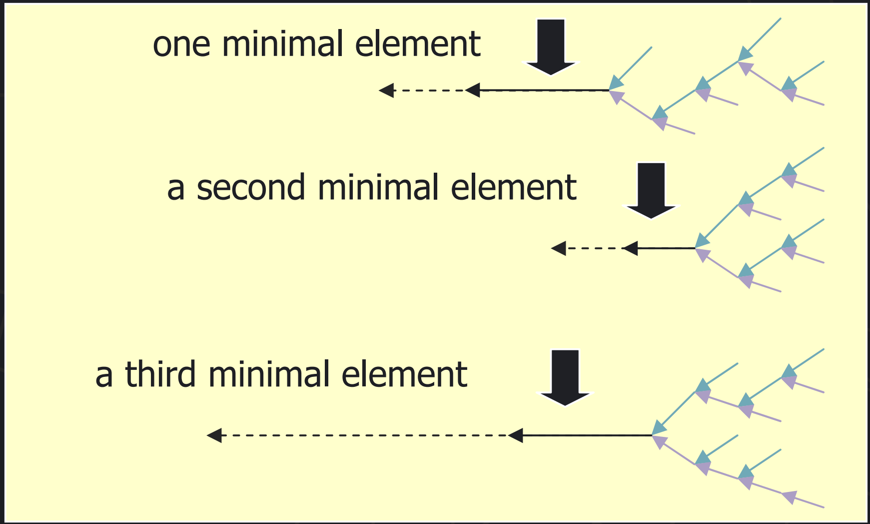
1. The input values, the guesses, and the new computed values form maximal chains $s_1 \subset_1 \cdots \subset_1 s_k$.
2. The maximal chains form trees. Every tree has only one minimal element.
3. The predecessors $r \subset_1 s_1$ of the minimal elements s_1 are not computed.

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

Computation over strings

Example: $\Sigma = (\{a, b\}^*;\ \varepsilon, a, b;\ add, sub_1;\ =)$





Definition: $s \subset_1 r \iff sub_1(r) = s.$

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.

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 $\{a, b\}$.

Why do we pad the codes?

Our goal:

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

Problems:

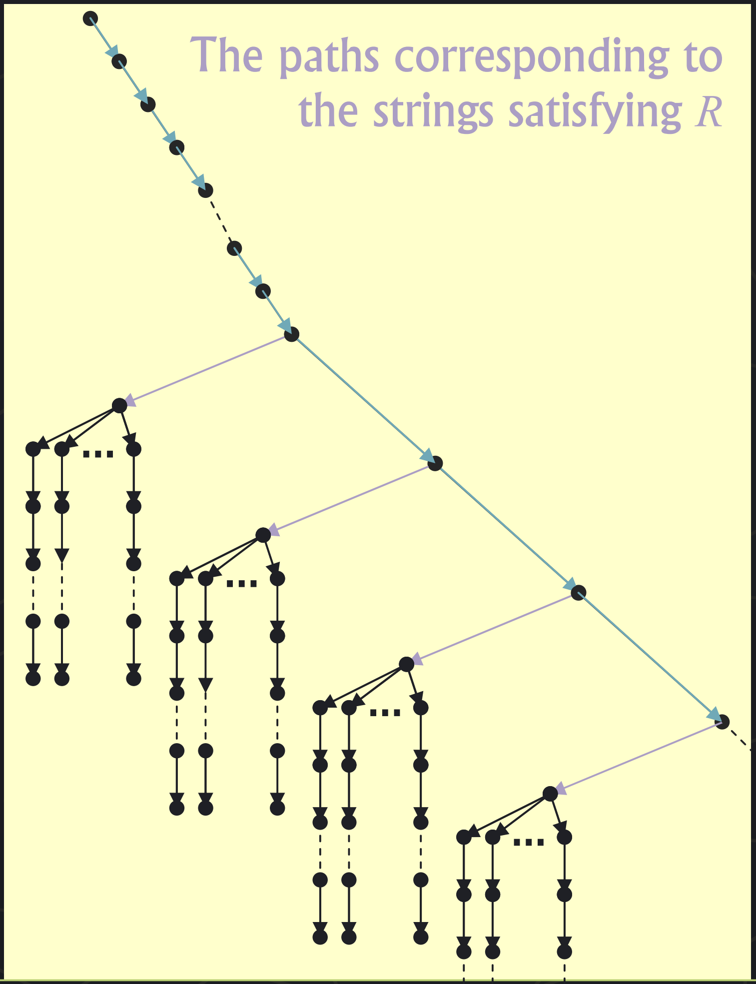
- Arbitrary strings can be guessed.
- A new R could imply $H_{\Sigma_R} \in NP_{\Sigma_R} \setminus DEC_{\Sigma_R}$ for the halting problem H_{Σ_R}.

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}.

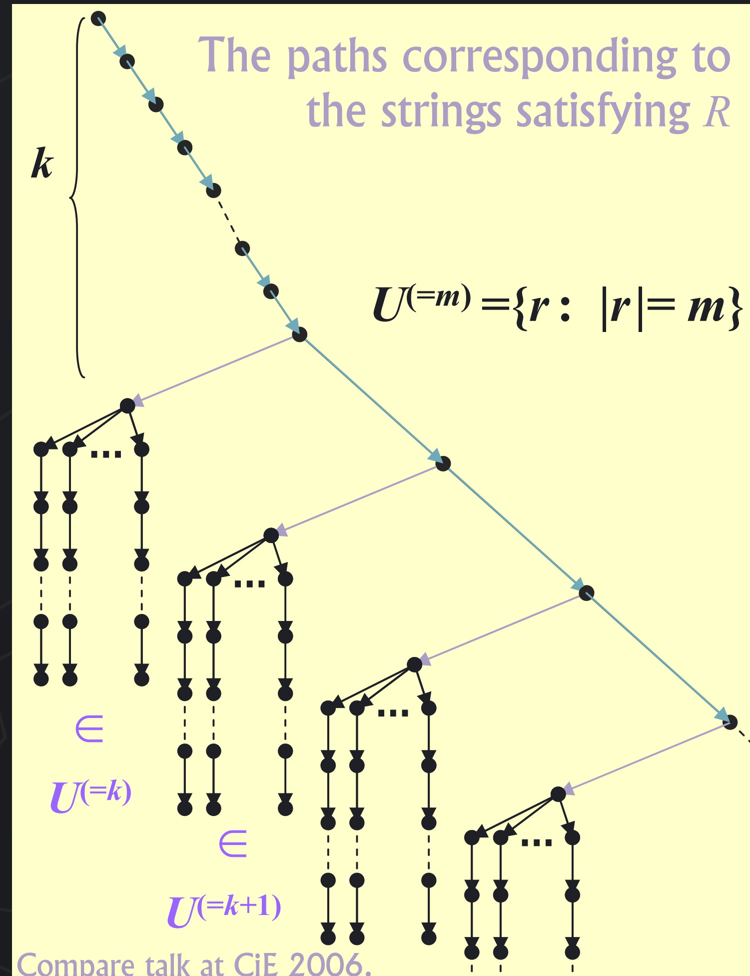
Solution:

- Padding strings:

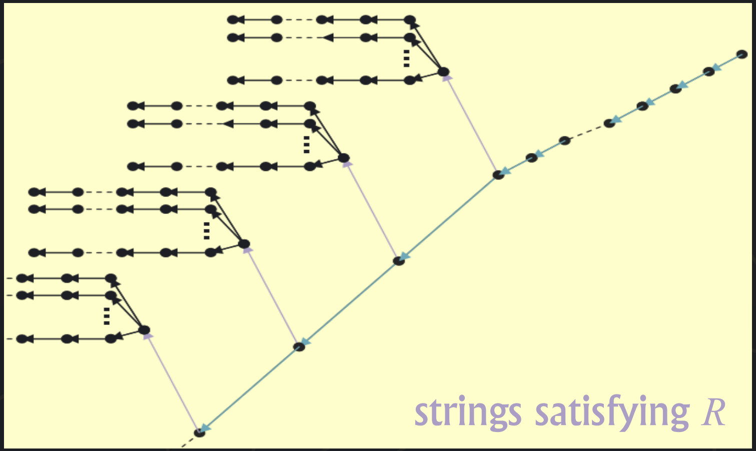
$$R(s) \Rightarrow (\exists r \in U^*)(\, s = ra^{|r|}\,).$$

It allows to replace

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



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}.

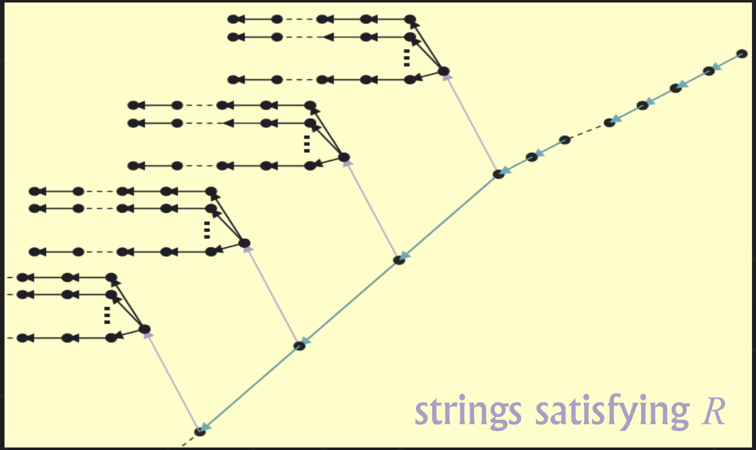
Solution:

- 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}.

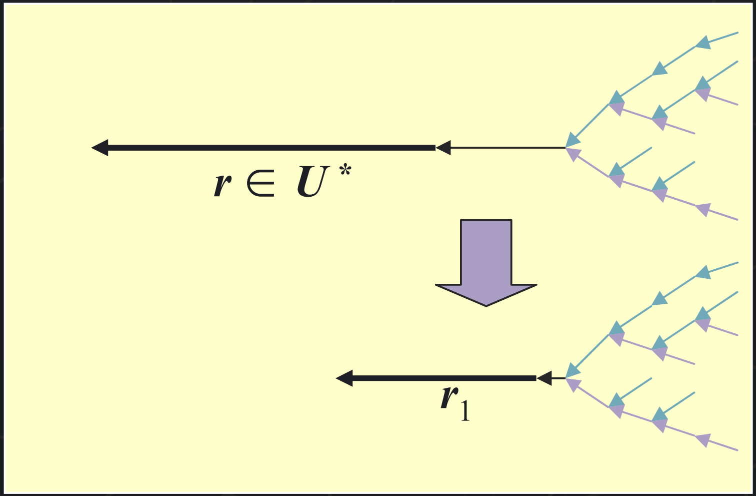
Solution:

- Padding strings:

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

It allows to replace

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



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 $W_\Sigma \subset U^\infty$ 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:

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

Christine Gaßner 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 $W_\Sigma \subset U^\infty$ with $P_\Sigma^{W_\Sigma} = NP_\Sigma^{W_\Sigma}$ (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:

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

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 $W_\Sigma \subset U^\infty$ with $P_\Sigma^{W_\Sigma} = NP_\Sigma^{W_\Sigma}$ (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:

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

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

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

UNI $= \{(x_1,\ldots, x_n, Code(M), b,\ldots, b) \mid \boldsymbol{x} \in (U^*)^\infty \;\&\; M$ 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

Christine Gaßner gassnerc@uni-greifswald.de

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

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

UNI $= \{(x_1,\ldots, x_n, Code(M\,), b,\ldots, b) \mid \boldsymbol{x} \in (U^*)^\infty \;\;\&\;\; M$ 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

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

UNI $= \{(x_1,\ldots, x_n, Code(M), b,\ldots, b) \mid \boldsymbol{x} \in (U^*)^\infty$ & 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

Christine Gaßner gassnerc@uni-greifswald.de

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

UNI $= \{(x_1,\ldots, x_n, Code(M), b,\ldots, b) \mid \boldsymbol{x} \in (U^*)^\infty \;\&\; M$ 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

Christine Gaßner gassnerc@uni-greifswald.de

The replacements for an ordered structure (1)

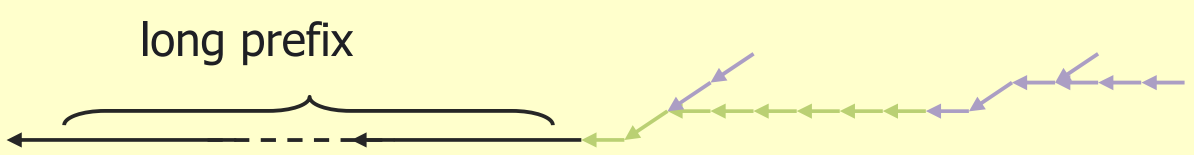
$(\mathbb{N}; 0, 1; shr, shl, inc \circ shl; R, \leq)$

The binary code of an input: $11000 \cdots 11010101101111101111111$

Results after 9 steps:

$11000 \cdots 11010101101111101111111$
$11000 \cdots 110101011011111000$
$11000 \cdots 110101011011111011111111010$
$$\underbrace{\underbrace{11000 \cdots 1101010110111110}_{\text{minimal element}}}_{\text{long prefix}}$$

Description by a tree:



The replacements for an ordered structure (2)

$(\mathbb{N}; 0, 1; shr, shl, inc \circ shl; R, \leq)$

If s satisfies R, then s can be replaced by some s_0

$bin(s) = r1^{|r|}$ $\quad$ $r = \langle bin(x_1),\ldots, bin(x_n)\rangle\, Code^*(M)\, 0^t$

⇓

$bin(s_0) = r_0 1^{|r_0|}$ $\quad$ $r_0 = \underbrace{\langle 10\cdots\cdots\rangle}_{\text{any length}}\ \underbrace{Code^*(M_0)}_{\text{where } \forall x\, M_0(x)\downarrow^4}\ 0000$

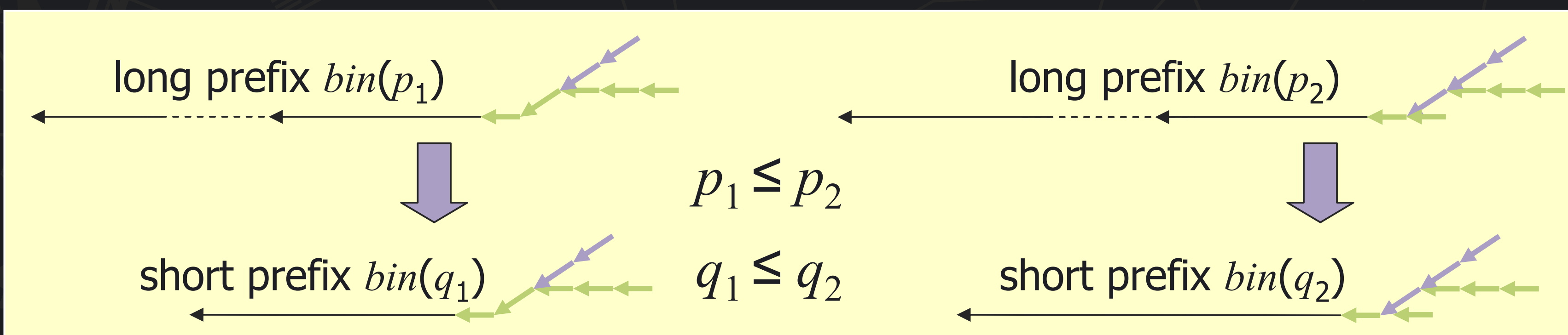
The replacements for an ordered structure (3)

Long prefixes in the binary code of

- large guesses
- large inputs

can be replaced by short prefixes

- without changing the computation path
- such that the order remains valid.



Christine Gaßner gassnerc@uni-greifswald.de

The reduction (Proof of $P_\Sigma = NP_\Sigma$):

$UNI_\Sigma = RES\text{-}UNI_\Sigma$ (the guesses can be restricted such that the order remains valid)

↓ Replace inputs such that the order remains valid.

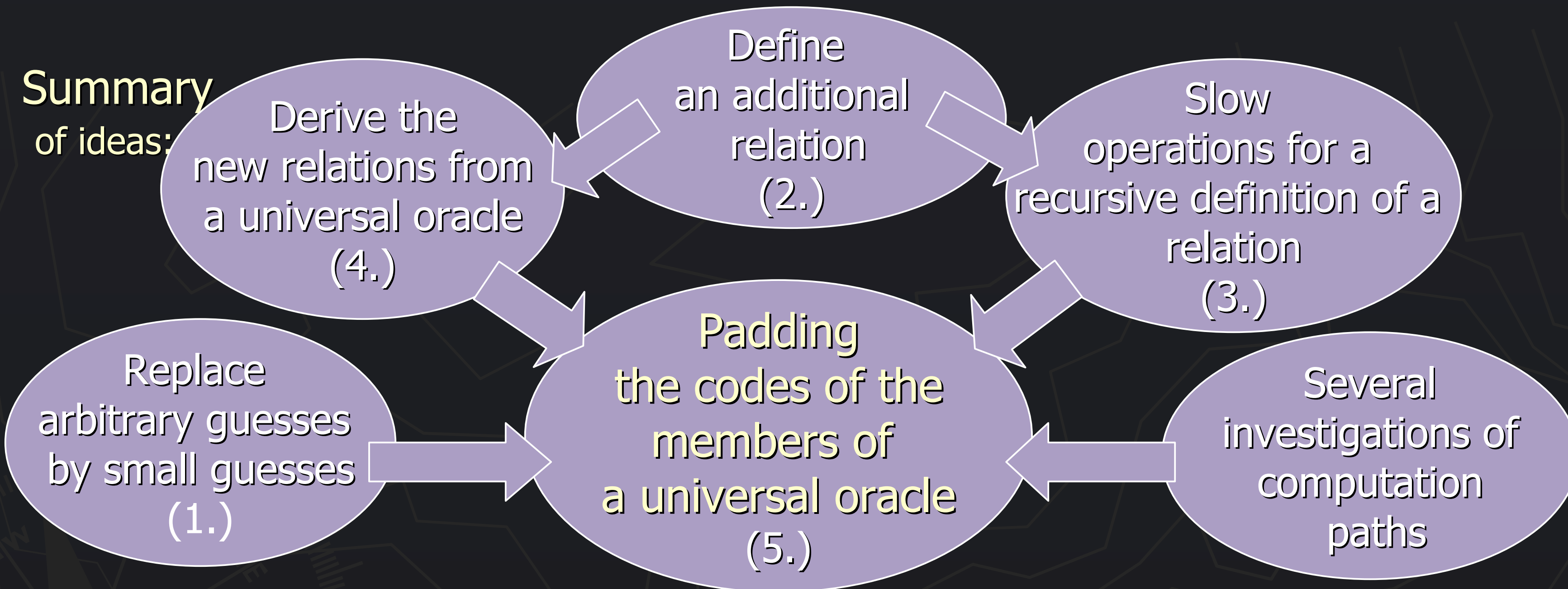
$SUB\text{-}UNI_\Sigma$ (small inputs) $SUB\text{-}UNI_\Sigma \subset RES\text{-}UNI_\Sigma$

↓

Output: a / b

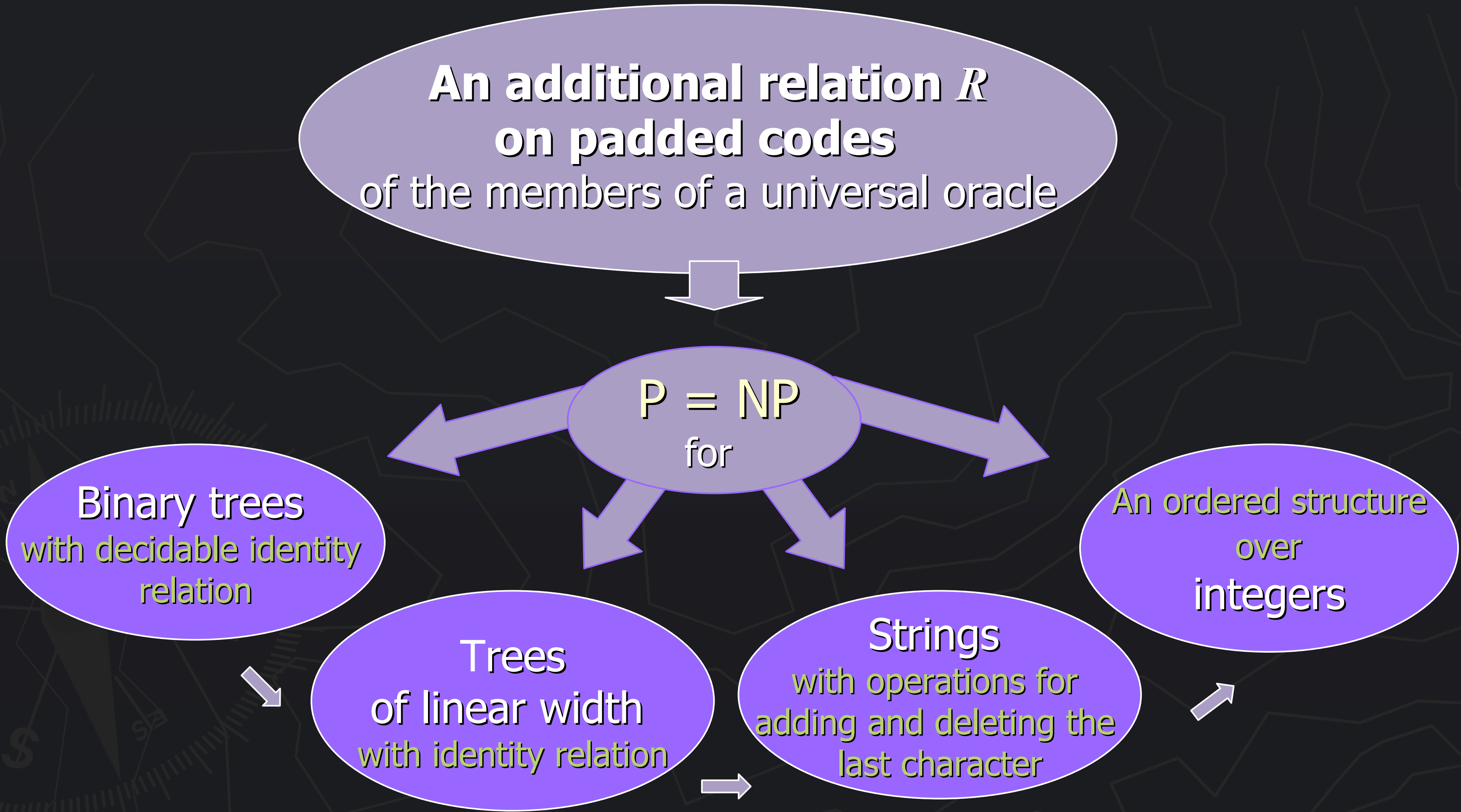
A summary of the ideas for the construction of structures of finite signature with P = NP

Summary of ideas:



1. Koiran (1994): DNP = NP for ($\mathbb{R}$; 0, 1; +, − ; ≤)
2. Poizat (1995): Is there a structure of finite signature with P = NP?
3. Prunescu (2001): Talk on an idea to define an additional relation over a term algebra which implies P = NP
4. Mainhardt (2001): P = NP for a structure of infinite signature
5. Gaßner (2004): P = NP for structures over trees with decidable identity

Several structures of finite signature with P = NP



Christine Gaßner gassnerc@uni-greifswald.de

In order to get $\mathrm{P}_{\Sigma_R} = \mathrm{NP}_{\Sigma_R}$ we can choose:

- Using some oracle as O_Σ and padding the elements.
  The oracle can be derived from:
  - UNI_Σ
  - SAT_Σ
- A directly recursive definition of the relation R
  (analogously to an oracle).

- Does there hold P = NP for some known structures of finite signature?
- Are there other constructions of structures with P = NP without using padding?

- Does there hold P = NP for some known structures of finite signature?
- Are there other constructions of structures with P = NP without using padding?

Thank you for your attention!

Christine Gaßner
Greifswald.

Thanks also to

Volkmar Liebscher,
Rainer Schimming.

Appendix: The new relation R (some details)

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

Σ_R = Expansion of Σ by R

$$W_\Sigma^{(0)} = \emptyset$$

$$W_\Sigma^{(i+1)} = \{([\langle \boldsymbol{x} \rangle], Code(M), [b^t]) \in U^i \mid M \text{ is non-det. } W_\Sigma^{(i)}\text{-mach. } \& \ M(\boldsymbol{x})\downarrow^t \}$$

$$W_\Sigma = \cup_{i \geq 1} W_\Sigma^{(i)} \quad \Rightarrow \quad P_\Sigma^{W_\Sigma} = NP_\Sigma^{W_\Sigma}$$

$s = d_1 \cdots d_k \in U^*$	string over U stored in one register
$[d_1 \cdots d_k] = (d_1, \ldots, d_k) \in U^k$	k-tuple over U stored in k registers
$\boldsymbol{x} \in (U^*)^\infty$	tuple of strings over U
$\langle \boldsymbol{x} \rangle \in U^*$	code the tuple $\boldsymbol{x}$

The relation R:

$$R(s) \quad \Leftrightarrow (\exists r \in U^*)([r] \in W_\Sigma \ \& \ s = ra^{|r|})$$

Theorem:

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