Equations over sets of natural numbers

Artur Jeż

Institute of Computer Science
University of Wrocław

May 22, 2008
Joint work with Alexander Okhotin, University of Turku, Academy of Finland.
Equations over languages

\[
\begin{align*}
\varphi_1(X_1, \ldots, X_n) &= \psi_1(X_1, \ldots, X_n) \\
\vdots \\
\varphi_m(X_1, \ldots, X_n) &= \psi_m(X_1, \ldots, X_n)
\end{align*}
\]

\(X_i\): subset of \(\Sigma^*\).

\(\varphi_i\): variables, constants, operations on sets.

Solutions: least, greatest, unique

Example

\(X = XX \cup \{a\} \cup \{b\} \cup \{\epsilon\}\)

Least solution: the Dyck language.

Greatest solution: \(\Sigma^*\).
Equations over languages

\[
\begin{align*}
\varphi_1(X_1, \ldots, X_n) &= \psi_1(X_1, \ldots, X_n) \\
\vdots & \\
\varphi_m(X_1, \ldots, X_n) &= \psi_m(X_1, \ldots, X_n)
\end{align*}
\]

- \(X_i\): subset of \(\Sigma^*\).
Equations over languages

\[
\begin{align*}
\varphi_1(X_1, \ldots, X_n) &= \psi_1(X_1, \ldots, X_n) \\
&\vdots \\
\varphi_m(X_1, \ldots, X_n) &= \psi_m(X_1, \ldots, X_n)
\end{align*}
\]

- \(X_i\): subset of \(\Sigma^*\).
- \(\varphi_i\): variables, constants, operations on sets.
Equations over languages

\[
\begin{align*}
\varphi_1(X_1, \ldots, X_n) &= \psi_1(X_1, \ldots, X_n) \\
&\vdots \\
\varphi_m(X_1, \ldots, X_n) &= \psi_m(X_1, \ldots, X_n)
\end{align*}
\]

- \(X_i\): subset of \(\Sigma^*\).
- \(\varphi_i\): variables, constants, operations on sets.

Solutions: least, greatest, unique
Equations over languages

\[
\begin{align*}
\varphi_1(X_1, \ldots, X_n) &= \psi_1(X_1, \ldots, X_n) \\
\vdots \\
\varphi_m(X_1, \ldots, X_n) &= \psi_m(X_1, \ldots, X_n)
\end{align*}
\]

- $X_i$: subset of $\Sigma^*$.
- $\varphi_i$: variables, constants, operations on sets.

Solutions: least, greatest, unique

Example

\[X = XX \cup \{a\}X\{b\} \cup \{\varepsilon\}\]
Equations over languages

\[
\begin{aligned}
\varphi_1(X_1, \ldots, X_n) &= \psi_1(X_1, \ldots, X_n) \\
\vdots \\
\varphi_m(X_1, \ldots, X_n) &= \psi_m(X_1, \ldots, X_n)
\end{aligned}
\]

- \(X_i\): subset of \(\Sigma^*\).
- \(\varphi_i\): variables, constants, operations on sets.

Solutions: least, greatest, unique

Example

\[X = XX \cup \{a\}X\{b\} \cup \{\varepsilon\}\]

Least solution: the Dyck language.
Equations over languages

\[
\begin{array}{l}
\varphi_1(X_1, \ldots, X_n) = \psi_1(X_1, \ldots, X_n) \\
\vdots \\
\varphi_m(X_1, \ldots, X_n) = \psi_m(X_1, \ldots, X_n)
\end{array}
\]

- \(X_i\): subset of \(\Sigma^*\).
- \(\varphi_i\): variables, constants, operations on sets.

Solutions: least, greatest, unique

Example

\[X = XX \cup \{a\}X\{b\} \cup \{\varepsilon\}\]

Least solution: the Dyck language.
Greatest solution: \(\Sigma^*\).
Interesting special cases

- Resolved equations

\[ X_i = \varphi_i(X_1, \ldots, X_n) \quad \text{for } i = 1, \ldots, n \]
Interesting special cases

- Resolved equations

\[ X_i = \varphi_i(X_1, \ldots, X_n) \quad \text{for } i = 1, \ldots, n \]

- Least solutions
Interesting special cases

- Resolved equations

\[ X_i = \varphi_i(X_1, \ldots, X_n) \quad \text{for} \ i = 1, \ldots, n \]

- Least solutions
- Monotone operations (\(\cap, \cup, \cdot\))
Interesting special cases

- Resolved equations

\[ X_i = \varphi_i(X_1, \ldots, X_n) \quad \text{for } i = 1, \ldots, n \]

- Least solutions
- Monotone operations (\(\cap, \cup, \cdot\))
- Connected with grammars (non-terminal \(X \leftrightarrow\) variable \(X\))
Interesting special cases

- Resolved equations
  \[ X_i = \varphi_i(X_1, \ldots, X_n) \quad \text{for } i = 1, \ldots, n \]
  - Least solutions
  - Monotone operations (\(\cap, \cup, \cdot\))
  - Connected with grammars (non-terminal \(X \leftrightarrow\) variable \(X\))

- Unary languages \(\rightarrow\) numbers
  - resolved
  - unresolved
Numbers and the unary alphabet

Unary: $\Sigma = \{a\}$. 

Language $\rightarrow$ set of numbers

Language equations $\rightarrow$ Equations over subsets of $\mathbb{N}$

Remark: Focus: resolved (EQ) and unresolved equations over sets of natural numbers with $\cap$, $\cup$, $+$. 

Artur Jeż (Wrocław)  May 22, 2008 5 / 1
Numbers and the unary alphabet

Unary: $\Sigma = \{a\}$.

$a^n \iff \text{number } n$
Numbers and the unary alphabet

Unary: $\Sigma = \{a\}$.

- $a^n \leftrightarrow$ number $n$
- $a^n \cdot a^m \leftrightarrow n + m$
Numbers and the unary alphabet

Unary: $\Sigma = \{a\}$.

- $a^n \leftrightarrow$ number $n$
- $a^n \cdot a^m \leftrightarrow n + m$
- Language $\leftrightarrow$ set of numbers
Numbers and the unary alphabet

Unary: $\Sigma = \{a\}$.

- $a^n \leftrightarrow$ number $n$
- $a^n \cdot a^m \leftrightarrow n + m$
- Language $\leftrightarrow$ set of numbers
- $K \cdot L \leftrightarrow X + Y = \{x + y \mid x \in X, y \in Y\}$
Numbers and the unary alphabet

Unary: $\Sigma = \{a\}$.

- $a^n \leftrightarrow$ number $n$
- $a^n \cdot a^m \leftrightarrow n + m$
- Language $\leftrightarrow$ set of numbers
- $K \cdot L \leftrightarrow X + Y = \{x + y \mid x \in X, y \in Y\}$
- Language equations $\leftrightarrow$ Equations over subsets of $\mathbb{N}$
Numbers and the unary alphabet

Unary: $\Sigma = \{a\}$.

- $a^n \leftrightarrow$ number $n$
- $a^n \cdot a^m \leftrightarrow n + m$
- Language $\leftrightarrow$ set of numbers
- $K \cdot L \leftrightarrow X + Y = \{x + y \mid x \in X, y \in Y\}$
- Language equations $\leftrightarrow$ Equations over subsets of $\mathbb{N}$

Remark

Focus: resolved (EQ) and unresolved equations over sets of natural numbers with $\cap, \cup, +$. 
Equations over sets of numbers

\[
\begin{align*}
\psi_1(X_1, \ldots, X_n) &= \varphi_1(X_1, \ldots, X_n) \\
&\vdots \\
\psi_m(X_1, \ldots, X_n) &= \varphi_m(X_1, \ldots, X_n)
\end{align*}
\]

- $X_i$: subset of $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$. 

$\varphi_i, \psi_i$: variables, singleton constants, operations on sets. 

For $S, T \subseteq \mathbb{N}_0$,

- $S \cup T$,
- $S \cap T$,
- $S + T = \{x + y | x \in S, y \in T\}$. 

Example $X = (X + \{2\}) \cup \{0\}$.

Unique solution: the even numbers
Equations over sets of numbers

\[
\begin{aligned}
\psi_1(X_1, \ldots, X_n) &= \varphi_1(X_1, \ldots, X_n) \\
&\vdots \\
\psi_m(X_1, \ldots, X_n) &= \varphi_m(X_1, \ldots, X_n)
\end{aligned}
\]

- $X_i$: subset of $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.
- $\varphi_i, \psi_i$: variables, singleton constants, operations on sets.
Equations over sets of numbers

\[
\begin{align*}
\psi_1(X_1, \ldots, X_n) &= \varphi_1(X_1, \ldots, X_n) \\
&\vdots \\
\psi_m(X_1, \ldots, X_n) &= \varphi_m(X_1, \ldots, X_n)
\end{align*}
\]

- \(X_i\): subset of \(\mathbb{N}_0 = \{0, 1, 2, \ldots\}\).
- \(\varphi_i, \psi_i\): variables, singleton constants, operations on sets.
- For \(S, T \subseteq \mathbb{N}_0\),
  - \(S \cup T, S \cap T\).
  - \(S + T = \{x + y \mid x \in S, y \in T\}\).

Example: \(X = (X + \{2\}) \cup \{0\}\). Unique solution: the even numbers.
Equations over sets of numbers

\[
\begin{align*}
\psi_1(X_1, \ldots, X_n) &= \varphi_1(X_1, \ldots, X_n) \\
\vdots \\
\psi_m(X_1, \ldots, X_n) &= \varphi_m(X_1, \ldots, X_n)
\end{align*}
\]

- $X_i$: subset of $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.
- $\varphi_i, \psi_i$: variables, singleton constants, operations on sets.

For $S, T \subseteq \mathbb{N}_0$,
- $S \cup T, S \cap T$.
- $S + T = \{x + y \mid x \in S, y \in T\}$.

Example

$X = (X + \{2\}) \cup \{0\}$
Equations over sets of numbers

\[
\begin{cases}
\psi_1(X_1, \ldots, X_n) = \varphi_1(X_1, \ldots, X_n) \\
\vdots \\
\psi_m(X_1, \ldots, X_n) = \varphi_m(X_1, \ldots, X_n)
\end{cases}
\]

- \(X_i\): subset of \(\mathbb{N}_0 = \{0, 1, 2, \ldots\}\).
- \(\varphi_i, \psi_i\): variables, singleton constants, operations on sets.
- For \(S, T \subseteq \mathbb{N}_0\),
  - \(S \cup T, S \cap T\).
  - \(S + T = \{x + y \mid x \in S, y \in T\}\).

**Example**

\[X = (X + \{2\}) \cup \{0\}\]  
Unique solution: the even numbers
Outline of the results

1. Resolved—expressive power
   How complicated the sets can be?
   - with regular notation
   - much more
Outline of the results

1. Resolved—expressive power
   How complicated the sets can be?
     - with regular notation
     - much more

2. Resolved: complexity
   How many resources are needed to recognise? EXPTIME
Outline of the results

1. Resolved—expressive power
   How complicated the sets can be?
   ▶ with regular notation
   ▶ much more

2. Resolved: complexity
   How many resources are needed to recognise?
   EXPTIME

3. General: universality
   $\cap, \cdot$ and $\cup, \cdot$ are computationally universal
Outline of the results

1. Resolved—expressive power
   How complicated the sets can be?
   ▶ with regular notation
   ▶ much more

2. Resolved: complexity
   How many resources are needed to recognise?
   \textsc{EXPTIME}

3. General: universality
   \(\cap, \cdot\) and \(\cup, \cdot\) are computationally universal

4. One variable
   How to encode results in one variable?
Outline of the results

1 Resolved—expressive power
   How complicated the sets can be?
   ▶ with regular notation
   ▶ much more

2 Resolved: complexity
   How many resources are needed to recognise?
   EXPTIME

3 General: universality
   $\cap, \cdot$ and $\cup, \cdot$ are computationally universal

4 One variable
   How to encode results in one variable?

5 General: addition only
   Can we use only addition?
Positional notation

- Base $k$. 

\[ \sum_{k} = \{0, 1, \ldots, k-1\}. \]

Numbers $\leftrightarrow$ strings in $\Sigma^*$. 

Sets of numbers $\leftrightarrow$ languages over $\Sigma^k$.

Example $\left(\Sigma^*\right)_4^4 = \{4^n | n \geq 0\}$.
Positional notation

- Base $k$.
- $\Sigma_k = \{0, 1, \ldots, k - 1\}$.

Example: $(10^*)_4 = \{4^n | n \geq 0\}$
Positional notation

- Base $k$.
- $\Sigma_k = \{0, 1, \ldots, k - 1\}$.
- Numbers $\leftrightarrow$ strings in $\Sigma_k^\ast \setminus 0\Sigma_k^\ast$.
Positional notation

- Base $k$.
- $\Sigma_k = \{0, 1, \ldots, k - 1\}$.
- Numbers $\leftrightarrow$ strings in $\Sigma_k^* \setminus 0\Sigma_k^*$.
- Sets of numbers $\leftrightarrow$ languages over $\Sigma_k$. 

Example $(10^*)^4 = \{4^n | n \geq 0\}$

We focus on properties in base-$k$ notation.
Positional notation

- Base $k$. 
  - $\Sigma_k = \{0, 1, \ldots, k - 1\}$. 
- Numbers $\leftrightarrow$ strings in $\Sigma_k^* \setminus 0\Sigma_k^*$. 
- Sets of numbers $\leftrightarrow$ languages over $\Sigma_k$.

Example

$$(10^*)_4 = \{4^n \mid n \geq 0\}$$
Positional notation

- Base $k$.
- $\Sigma_k = \{0, 1, \ldots, k - 1\}$.
- Numbers $\leftrightarrow$ strings in $\Sigma_k^* \setminus 0\Sigma_k^*$.
- Sets of numbers $\leftrightarrow$ languages over $\Sigma_k$.

Example

$$(10^*)_4 = \{4^n \mid n \geq 0\}$$

- We focus on properties in base-$k$ notation
Important example—$(10^*)_4$

Example

\[ X_1 = (X_2 + X_2 \cap X_1 + X_3) \cup \{1\} \]
\[ X_2 = (X_{12} + X_2 \cap X_1 + X_1) \cup \{2\} \]
\[ X_3 = (X_{12} + X_{12} \cap X_1 + X_2) \cup \{3\} \]
\[ X_{12} = X_3 + X_3 \cap X_1 + X_2 \]
Important example—$(10^*)_4$

Example

\[
X_1 = (X_2 + X_2 \cap X_1 + X_3) \cup \{1\}
\]

\[
X_2 = (X_{12} + X_2 \cap X_1 + X_1) \cup \{2\}
\]

\[
X_3 = (X_{12} + X_{12} \cap X_1 + X_2) \cup \{3\}
\]

\[
X_{12} = X_3 + X_3 \cap X_1 + X_2
\]
Important example—\((10^*)_4\)

Example

\[
X_1 = (X_2 + X_2 \cap X_1 + X_3) \cup \{1\}
\]
\[
X_2 = (X_{12} + X_2 \cap X_1 + X_1) \cup \{2\}
\]
\[
X_3 = (X_{12} + X_{12} \cap X_1 + X_2) \cup \{3\}
\]
\[
X_{12} = X_3 + X_3 \cap X_1 + X_2
\]

Least solution:

\[
((10^*)_4, (20^*)_4, (30^*)_4, (120^*)_4)
\]
Important example—\((10^*)_4\)

Example

\[
\begin{align*}
X_1 &= (X_2 + X_2 \cap X_1 + X_3) \cup \{1\} \\
X_2 &= (X_{12} + X_2 \cap X_1 + X_2) \cup \{2\} \\
X_3 &= (X_{12} + X_{12} \cap X_1 + X_2) \cup \{3\} \\
X_{12} &= X_3 + X_3 \cap X_1 + X_2
\end{align*}
\]

Least solution:

\[((10^*)_4, (20^*)_4, (30^*)_4, (120^*)_4)\]

Checking:

- \(X_2 + X_2 = 20^* + 20^* = 10^+ \cup 20^*20^*\)

Remark: Resolved equations with \(\cap\), \(+\) or \(\cup\), \(+\) specify only ultimately periodic sets.
Important example—\((10^*)_4\)

Example

\[
X_1 = (X_2 + X_2 \cap X_1 + X_3) \cup \{1\}
\]
\[
X_2 = (X_{12} + X_2 \cap X_1 + X_1) \cup \{2\}
\]
\[
X_3 = (X_{12} + X_{12} \cap X_1 + X_2) \cup \{3\}
\]
\[
X_{12} = X_3 + X_3 \cap X_1 + X_2
\]

Least solution:

\[((10^*)_4, (20^*)_4, (30^*)_4, (120^*)_4)\]

Checking:

- \(X_2 + X_2 = 20^* + 20^* = 10^+ \cup 20^*20^*\)
- \(X_1 + X_3 = 10^* + 30^* = 10^+ \cup 10^*30^* \cup 30^*10^*\)
Important example—$(10^*)_4$

Example

\[
\begin{align*}
X_1 &= (X_2 + X_2 \cap X_1 + X_3) \cup \{1\} \\
X_2 &= (X_{12} + X_2 \cap X_1 + X_1) \cup \{2\} \\
X_3 &= (X_{12} + X_{12} \cap X_1 + X_2) \cup \{3\} \\
X_{12} &= X_3 + X_3 \cap X_1 + X_2
\end{align*}
\]

Least solution:

\[((10^*)_4, (20^*)_4, (30^*)_4, (120^*)_4)\]

Checking:

- $X_2 + X_2 = 20^* + 20^* = 10^+ \cup 20^* 20^*$
- $X_1 + X_3 = 10^* + 30^* = 10^+ \cup 10^* 30^* \cup 30^* 10^*$,
- $(X_2 + X_2) \cap (X_1 + X_3) = 10^+.$
Important example—$(10^*)_4$

Example

$$X_1 = (X_2 + X_2 \cap X_1 + X_3) \cup \{1\}$$
$$X_2 = (X_{12} + X_2 \cap X_1 + X_1) \cup \{2\}$$
$$X_3 = (X_{12} + X_{12} \cap X_1 + X_2) \cup \{3\}$$
$$X_{12} = X_3 + X_3 \cap X_1 + X_2$$

Least solution:

$$((10^*)_4, (20^*)_4, (30^*)_4, (120^*)_4)$$

Checking:

- $X_2 + X_2 = 20^* + 20^* = 10^+ \cup 20^*20^*$
- $X_1 + X_3 = 10^* + 30^* = 10^+ \cup 10^*30^* \cup 30^*10^*$,
- $(X_2 + X_2) \cap (X_1 + X_3) = 10^+$.

Remark

Resolved equations with $\cap$, $+$ or $\cup$, $+$ specify only ultimately periodic sets.
Generalisation and how to apply it

Idea

*We append digits from the left, controlling the sets of digits.*
Generalisation and how to apply it

**Idea**

*We append digits from the left, controlling the sets of digits.*

Using the idea

Theorem

For every $k$ and $R \subset \{0, \ldots, k-1\}$ if $R$ is regular then $(R \cdot \Sigma_0)^k \in \text{EQ}.$

Example (Application)

Let $S \subseteq (10^* \Sigma_0^k)^k.$ How to obtain $S' = \left\{ (10^n (d+1)^0)_k : (10^n d^0)_k \in S \right\} ?$

$S' = \bigcup_{d \in \Sigma^k} (S \cap (10^* d^0^k) + (10^* \Sigma^k)) \cap (10^* (d+1)^0^k).$

Artur Jeż (Wrocław)
Generalisation and how to apply it

Idea

*We append digits from the left, controlling the sets of digits.*

Using the idea

- \((ij0^*)_k\) for every \(i, j, k\)
Generalisation and how to apply it

Idea

*We append digits from the left, controlling the sets of digits.*

Using the idea

- \((ij0^*)_k\) for every \(i, j, k\)

Theorem

*For every \(k\) and \(R \subset \{0, \ldots, k - 1\}^*\) if \(R\) is regular then \((R)_k \in EQ.*
Generalisation and how to apply it

Idea

*We append digits from the left, controlling the sets of digits.*

Using the idea

- \((ij0^*)_k\) for every \(i, j, k\)

Theorem

*For every \(k\) and \(R \subset \{0, \ldots, k-1\}^*\) if \(R\) is regular then \((R)_k \in \mathcal{EQ}.*
Generalisation and how to apply it

**Idea**

We append digits from the left, controlling the sets of digits.

Using the idea

- \((ij0^*)_k\) for every \(i, j, k\)

**Theorem**

For every \(k\) and \(R \subset \{0, \ldots, k - 1\}^*\) if \(R\) is regular then \((R)_k \in EQ\).

**Example (Application)**

Let \(S \subseteq (10^* \Sigma_k 0^*)_k\). How to obtain \(S' = \{(10^n(d + 1)0^m)_k : (10^n d 0^m)_k \in S\}\)?

\[
S' = \bigcup_{d \in \Sigma_k} \left( \left( S \cap (10^*d0^*)_k \right) + (10^*)_k \right) \cap (10^*(d + 1)0^*)_k
\]
Application: complexity

Definition

Complexity theory (of a set S)—how many resources are needed to answer a question?
"Given $n$, does $n \in S$"
Application: complexity

Definition

Complexity theory (of a set $S$)—how many resources are needed to answer a question?

"Given $n$, does $n \in S$"

Resources:
- space
- time
- non-determinism
Application: complexity

**Definition**

*Complexity theory* (of a set $S$)—how many resources are needed to answer a question?

"Given $n$, does $n \in S$"

Resources:
- space
- time
- non-determinism

For example EXPTIME.
### Application: complexity

#### Definition

**Complexity theory** (of a set $S$)—how many resources are needed to answer a question?

"Given $n$, does $n \in S$"

**Resources:**

- space
- time
- non-determinism

For example EXPTIME.

#### Definition

**Reduction:** Problem $P \geq P'$ if we can answer $P$ (fast) then we can answer $P'$ (fast).
Problem

Given a resolved system with $\cap, \cup, +$ and a number $n$, does $n \in S_1$. 

EXPTIME-complete

Idea

state of the machine is a string—encode as a number

easy to define final accepting computation

recurse back

transition is a local change

easily encoded using regular notation

Example

Machine

```
abcd
q
```

e

String

```
(0a0b0cqd0e)k → (0a0b0c0d'q'e)k
```

If $(0a0b0c0d'q'e)_k$ is accepting we want to add

```
(0a0b0cqd0e)_k - (0d'q'e)_k
```

Using the trick with intersection with regular sets.
Problem

*Given a resolved system with \( \cap, \cup, + \) and a number \( n \), does \( n \in S_1 \).*

EXPTIME-complete
Problem

Given a resolved system with $\cap, \cup, +$ and a number $n$, does $n \in S_1$.

EXPTIME-complete

Idea

- The state of the machine is a string—encode as a number
- It's easy to define a final accepting computation
- We recurse back
- A transition is a local change that's easily encoded using regular notation

Example

Machine: $abcd$

String: $(0a0b0cqd0e)k \rightarrow (0a0b0c0dq'0e)k$

If $(0a0b0c0dq'0e)k$ is accepting, we want to add $(qd0d'q'e)k - (0d'q'e)k$.

Using the trick with intersection with regular sets.
Problem

Given a resolved system with $\cap, \cup, +$ and a number $n$, does $n \in S_1$.

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
Problem

Given a resolved system with \( \cap, \cup, + \) and a number \( n \), does \( n \in S_1 \).

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
- easy to define final accepting computation
Problem

Given a resolved system with \( \cap, \cup, + \) and a number \( n \), does \( n \in S_1 \).

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
- easy to define final accepting computation
- recurse back
Problem

Given a resolved system with $\cap, \cup, +$ and a number $n$, does $n \in S_1$.

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
- easy to define final accepting computation
- recurse back
- transition is a local change
Problem

Given a resolved system with $\cap$, $\cup$, $+$ and a number $n$, does $n \in S_1$.

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
- easy to define final accepting computation
- recurse back
- transition is a local change
- easily encoded using regular notation
Problem

Given a resolved system with $\cap, \cup, +$ and a number $n$, does $n \in S_1$.

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
- easy to define final accepting computation
- recurse back
- transition is a local change
- easily encoded using regular notation

Example
Problem

Given a resolved system with $\cap$, $\cup$, $+$ and a number $n$, does $n \in S_1$.

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
- easy to define final accepting computation
- recurse back
- transition is a local change
- easily encoded using regular notation

Example

- Machine $abcd^q e \rightarrow abcd' e^{q'}$
Problem

Given a resolved system with $\cap, \cup, +$ and a number $n$, does $n \in S_1$.

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
- easy to define final accepting computation
- recurse back
- transition is a local change
- easily encoded using regular notation

Example

- Machine $abcd^q e \rightarrow abcd' e^{q'}$
- String $(0a0b0cqd0e)_k \rightarrow (0a0b0c0d'q'e)_k$
Problem

Given a resolved system with $\cap, \cup, +$ and a number $n$, does $n \in S_1$.

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
- easy to define final accepting computation
- recurse back
- transition is a local change
- easily encoded using regular notation

Example

- Machine $abcd^qe \rightarrow abcd'q'e'$
- String $(0a0b0cqd0e)_k \rightarrow (0a0b0c0d'q'e)_k$
- If $(0a0b0c0d'q'e)_k$ is accepting we want to add $((qd0)_k - (0d'q')_k)$
Problem

Given a resolved system with $\cap$, $\cup$, $+$ and a number $n$, does $n \in S_1$.

EXPTIME-complete

Idea

- state of the machine is a string—encode as a number
- easy to define final accepting computation
- recurse back
- transition is a local change
- easily encoded using regular notation

Example

- Machine $abcdqe \rightarrow abcd'q'e'$
- String $(0a0b0cqd0e)_k \rightarrow (0a0b0c0d'q'e)_k$
- If $(0a0b0c0d'q'e)_k$ is accepting we want to add $((qd0)_k - (0d'q')_k)$
- Using the trick with intersection with regular sets.
More results—greater expressive power

Problem

Regular sets are very easy. Slow growth, decidable properties etc. Can we do better?

Idea

For regular languages we expanded numbers to the left. Maybe we can expand in both directions?

We can. But this is not easy.

Theorem

For every \( k \) and \( R \subset \{0, \ldots, k-1\} \)\(^\star\) if \( R \) is recognised by a trellis automaton \( M \) then \( (R^k) \in \text{EQ} \).
More results—greater expressive power

Problem

Regular sets are very easy. Slow growth, decidable properties etc. Can we do better?

Idea

For regular languages we expanded numbers to the left. Maybe we can expand in both directions?
More results—greater expressive power

**Problem**

*Regular sets are very easy. Slow growth, decidable properties etc. Can we do better?*

**Idea**

*For regular languages we expanded numbers to the left. Maybe we can expand in both directions?*

We can. But this is not easy.
Problem

Regular sets are very easy. Slow growth, decidable properties etc. Can we do better?

Idea

For regular languages we expanded numbers to the left. Maybe we can expand in both directions?

We can. But this is not easy.

Theorem

For every k and $R \subset \{0, \ldots, k - 1\}^*$ if $R$ is recognised by a trellis automaton $M$ then $(R)^k \in EQ$. 
A trellis automaton is a
\[ M = (\Sigma, Q, I, \delta, F) \] where:

- \( \Sigma \): input alphabet;
- \( Q \): finite set of states;
- \( I \): \( \Sigma \rightarrow Q \) sets initial states;
- \( \delta \): \( Q \times Q \rightarrow Q \), transition function;
- \( F \subseteq Q \): accepting states.

Closed under \( \cup \), \( \cap \), \( \sim \), not closed under concatenation.

Can recognize \( \{wcw\} \), \( \{a^n b^n c^n\} \), \( \{a^n b^{2n} c^n\} \), VALC.

Theorem
For every \( k \) and \( R \subset \{0, \ldots, k-1\}^* \) if \( R \) is recognised by a trellis automaton \( M \) then \( (R)_k \in EQ. \)
Trellis automata

Definition (Culik, Gruska, Salomaa, 1981)

A trellis automaton is a $M = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I$: $\Sigma \rightarrow Q$ sets initial states;
- $\delta$: $Q \times Q \rightarrow Q$, transition function;
- $F \subseteq Q$: accepting states.

Closed under $\cup$, $\cap$, $\sim$, not closed under concatenation.

Can recognize $\{wcw\}$, $\{a^n b^n c^n\}$, $\{a^n b^{2n} c^n\}$, VALC.

Theorem

For every $k$ and $R \subset \{0, \ldots, k - 1\}^*$ if $R$ is recognised by a trellis automaton $M$ then $(R)^k \in EQ.$
Trellis automata

Definition (Culik, Gruska, Salomaa, 1981)

A trellis automaton is a $\mathcal{M} = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I: \Sigma \rightarrow Q$ sets initial states;
- $\delta: Q \times Q \rightarrow Q$, transition function;
- $F \subseteq Q$: accepting states.

Closed under $\cup$, $\cap$, $\sim$, not closed under concatenation.

Can recognize $\{wcw\}$, $\{a^n b^n c^n\}$, $\{a^n b^{2n} c^n\}$, VALC.

Theorem

For every $k$ and $R \subseteq \{0, \ldots, k-1\}^*$ if $R$ is recognized by a trellis automaton $\mathcal{M}$ then $(R)^k \in EQ$. 

Equations over sets of natural numbers

May 22, 2008 14 / 1
A **trellis automaton** is a $M = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I : \Sigma \rightarrow Q$ sets initial states;
- $\delta : Q \times Q \rightarrow Q$, transition function;
- $F \subseteq Q$: accepting states.

Closed under $\cup$, $\cap$, $\sim$, not closed under concatenation.

Can recognize $\{wcw\}$, $\{a^n b^n c^n\}$, $\{a^n b^{2n} c^n\}$, VALC.

Theorem
For every $k$ and $R \subset \{0, \ldots, k - 1\}^*$ if $R$ is recognised by a trellis automaton $M$ then $(R^k) \in \mathcal{EQ}$. 

Artur Jeż (Wrocław)
A trellis automaton is a $M = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I: \Sigma \to Q$ sets initial states;

$\delta: Q \times Q \to Q$, transition function;

$F \subseteq Q$: accepting states.

Closed under $\cup$, $\cap$, $\sim$, not closed under concatenation.

Can recognize $\{wcw\}$, $\{a^n b^n c^n\}$, $\{a^n b^2 c^n\}$, VALC.

Theorem

For every $k$ and $R \subseteq \{0, \ldots, k-1\}^*$ if $R$ is recognised by a trellis automaton $M$ then $(R)_k \in EQ.$
A trellis automaton is a $M = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I: \Sigma \to Q$ sets initial states;
- $\delta: Q \times Q \to Q$, transition function;
- $F \subseteq Q$: accepting states.

Closed under $\cup$, $\cap$, $\sim$, not closed under concatenation.

Can recognize $\{wcw\}$, $\{a^n b^n c^n\}$, $\{a^n b^2 c^n\}$, VALC.

Theorem
For every $k$ and $R \subset \{0, \ldots, k - 1\}^*$ if $R$ is recognised by a trellis automaton $M$ then $(R)_k \in \text{EQ}$. 

Artur Jeż (Wrocław)
Trellis automata

Definition (Culik, Gruska, Salomaa, 1981)

A trellis automaton is a $M = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I : \Sigma \rightarrow Q$ sets initial states;
- $\delta : Q \times Q \rightarrow Q$, transition function;
- $F \subseteq Q$: accepting states.

Closed under $\cup$, $\cap$, $\sim$, not closed under concatenation.

Can recognize $\{wcw\}$, $\{a^n b^n c^n\}$, $\{a^n b^{2n} c^n\}$, VALC.

Theorem

For every $k$ and $R \subset \{0, \ldots, k-1\}^*$ if $R$ is recognised by a trellis automaton $M$ then $(R)^k \in \text{EQ}$. 

Artur Jeż (Wrocław)
A trellis automaton is a $M = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I: \Sigma \rightarrow Q$ sets initial states;
- $\delta: Q \times Q \rightarrow Q$, transition function;
- $F \subseteq Q$: accepting states.

Closed under $\cup$, $\cap$, $\sim$, not closed under concatenation.

Can recognize $\{wcw\}$, $\{a^n b^n c^n\}$, $\{a^n b^{2n} c^n\}$, VALC.

Theorem

For every $k$ and $R \subset \{0, \ldots, k-1\}^*$ if $R$ is recognized by a trellis automaton $M$ then $(R)^k \in EQ.$
A trellis automaton is a $M = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I : \Sigma \rightarrow Q$ sets initial states;
- $\delta : Q \times Q \rightarrow Q$, transition function;
- $F \subseteq Q$: accepting states.
A trellis automaton is a $M = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I : \Sigma \rightarrow Q$ sets initial states;
- $\delta : Q \times Q \rightarrow Q$, transition function;
- $F \subseteq Q$: accepting states.

Closed under $\cup$, $\cap$, $\sim$, not closed under concatenation.
Trellis automata

Definition (Culik, Gruska, Salomaa, 1981)

A trellis automaton is a $M = (\Sigma, Q, I, \delta, F)$ where:

- $\Sigma$: input alphabet;
- $Q$: finite set of states;
- $I : \Sigma \rightarrow Q$ sets initial states;
- $\delta : Q \times Q \rightarrow Q$, transition function;
- $F \subseteq Q$: accepting states.

Closed under $\cup, \cap, \sim$, not closed under concatenation.
Can recognize $\{wcw\}$, $\{a^n b^n c^n\}$, $\{a^n b^{2n}\}$, VALC.

Theorem

For every $k$ and $R \subset \{0, \ldots, k - 1\}^*$ if $R$ is recognised by a trellis automaton $M$ then $(R)_k \in EQ$. 

Artur Jeż (Wrocław)  Equations over sets of natural numbers  May 22, 2008 14 / 1
Computational completeness of language equations

- Model of computation: Turing Machine

Recursive sets:

**Definition**

$S$ is recursive if there exists $M$, such that $M[w] = 1$ for $w \in S$ and $M[w] = 0$ for $w \not\in S$.

Language equations over $\Sigma$, with $|\Sigma| \geq 2$.

**Theorem**

$L \subseteq \Sigma^*$ is given by unique solution of a system with $\{\cup, \cap, \sim, \cdot\}$ if and only if $L$ is recursive.

Multiple-letter alphabet essentially used.

✓ Remaking the argument for sets of numbers!
Computational completeness of language equations

- Model of computation: Turing Machine
- Recursive sets:

**Definition**

S is recursive if there exists M, such that $M[w] = 1$ for $w \in S$ and $M[w] = 0$ for $w \notin S$
Computational completeness of language equations

- Model of computation: Turing Machine
- Recursive sets:

**Definition**

$S$ is recursive if there exists $M$, such that $M[w] = 1$ for $w \in S$ and $M[w] = 0$ for $w \notin S$.

- Language equations over $\Sigma$, with $|\Sigma| \geq 2$.

Theorem $L \subseteq \Sigma^*$ is given by unique solution of a system with \{∪, ∩, ∼, ·\} if and only if $L$ is recursive.

Multiple-letter alphabet essentially used.

✓ Remaking the argument for sets of numbers!
Computational completeness of language equations

- Model of computation: Turing Machine
- Recursive sets:

**Definition**

S is recursive if there exists M, such that $M[w] = 1$ for $w \in S$ and $M[w] = 0$ for $w \notin S$

- Language equations over $\Sigma$, with $|\Sigma| \geq 2$.

**Theorem**

$L \subseteq \Sigma^*$ is given by unique solution of a system with $\{\cup, \cap, \sim, \cdot\}$ if and only if $L$ is recursive.
Computational completeness of language equations

- Model of computation: Turing Machine
- Recursive sets:

**Definition**

$S$ is recursive if there exists $M$, such that $M[w] = 1$ for $w \in S$ and $M[w] = 0$ for $w \notin S$.

- Language equations over $\Sigma$, with $|\Sigma| \geq 2$.

**Theorem**

$L \subseteq \Sigma^*$ is given by unique solution of a system with $\{\cup, \cap, \sim, \cdot\}$ if and only if $L$ is recursive.

- Multiple-letter alphabet essentially used.
Computational completeness of language equations

- Model of computation: Turing Machine
- Recursive sets:

**Definition**

$S$ is recursive if there exists $M$, such that $M[w] = 1$ for $w \in S$ and $M[w] = 0$ for $w \notin S$

- Language equations over $\Sigma$, with $|\Sigma| \geq 2$.

**Theorem**

$L \subseteq \Sigma^*$ is given by unique solution of a system with $\{\cup, \cap, \sim, \cdot\}$ if and only if $L$ is recursive.

- Multiple-letter alphabet essentially used.
- ✓ Remaking the argument for sets of numbers!
Outline of the construction

Theorem

$S \subseteq \mathbb{N}_0$ is given by unique solution of a system with $\{\cup, +\}$ ($\{\cap, +\}$) if and only if $S$ is recursive.
Outline of the construction

Theorem

$S \subseteq \mathbb{N}_0$ is given by unique solution of a system with $\{\cup, +\} \ (\{\cap, +\})$

if and only if

$S$ is recursive.

- Turing Machine $T$ for $S$
Outline of the construction

Theorem

\[ S \subseteq \mathbb{N}_0 \text{ is given by unique solution of a system with } \{ \cup, + \} \text{ and } \{ \cap, + \} \]

if and only if

\[ S \text{ is recursive.} \]

- Turing Machine \( T \) for \( S \)
- \( \text{VALC}(T) \) (transcription of computation): recognised by a trellis automaton
Outline of the construction

Theorem

\[ S \subseteq \mathbb{N}_0 \] is given by unique solution of a system with \( \{\cup, +\} \) \( \{\cap, +\} \)

if and only if

\( S \) is recursive.

- Turing Machine \( T \) for \( S \)
- \( \text{VALC}(T) \) (transcription of computation): recognised by a trellis automaton
- Trellis automata \( \rightarrow \) resolved equations over sets of numbers
Outline of the construction

Theorem

$S \subseteq \mathbb{N}_0$ is given by unique solution of a system with $\{\cup, +\}$ ($\{\cap, +\}$) if and only if $S$ is recursive.

- Turing Machine $T$ for $S$
- $\text{VALC}(T)$ (transcription of computation): recognised by a trellis automaton
- Trellis automata $\rightarrow$ resolved equations over sets of numbers
- Technical trick: resolved equations with $\cap, \cup, + \rightarrow$ unresolved with $\cup, +$ (or $\cap, +$)
Outline of the construction

**Theorem**

\[ S \subseteq \mathbb{N}_0 \text{ is given by unique solution of a system with } \{\cup, +\} (\{\cap, +\}) \]

if and only if \( S \) is recursive.

- Turing Machine \( T \) for \( S \)
- \( \text{VALC}(T) \) (transcription of computation): recognised by a trellis automaton
- Trellis automata \( \rightarrow \) resolved equations over sets of numbers
- Technical trick: resolved equations with \( \cap, \cup, + \) \( \rightarrow \) unresolved with \( \cup, + \) (or \( \cap, + \))
- Extracting numbers with notation \( L(T) \)

from numbers with notation \( \text{VALC}(T) \)
Outline of the construction

Theorem

$S \subseteq \mathbb{N}_0$ is given by unique solution of a system with \{\cup, +\} (\{\cap, +\}) if and only if \(S\) is recursive.

- Turing Machine \(T\) for \(S\)
- VALC(\(T\)) (transcription of computation): recognised by a trellis automaton
- Trellis automata \(\rightarrow\) resolved equations over sets of numbers
- Technical trick: resolved equations with \(\cap, \cup, +\) \(\rightarrow\) unresolved with \(\cup, +\) (or \(\cap, +\))
- Extracting numbers with notation \(L(T)\) from numbers with notation VALC(\(T\))

Remark

Least (greatest) solution—RE-sets (co-RE-sets).
One variable

Problem

*How many variables are needed to define something interesting?*
One variable

Problem

*How many variables are needed to define something interesting?*

Idea

*Encoding*

\[(S_1, \ldots, S_k) \rightarrow \bigcup_{i=1}^{k} p \cdot S_i - d_i.\]
One variable

**Problem**

*How many variables are needed to define something interesting?*

**Idea**

*Encoding*

\[(S_1, \ldots, S_k) \rightarrow \bigcup_{i=1}^{k} p \cdot S_i - d_i.\]

- EXPTIME holds for \( X = \varphi(X) \)
One variable

Problem

*How many variables are needed to define something interesting?*

Idea

*Encoding*

\[(S_1, \ldots, S_k) \rightarrow \bigcup_{i=1}^{k} p \cdot S_i - d_i.\]

- EXPTIME holds for \(X = \varphi(X)\)
- unique solution \(\varphi(X) = \psi(X)\) — recursively-hard (\(\cap, \cup, +\))
Is addition enough to define something interesting? (general case)
Addition only

Problem
Is addition enough to define something interesting? (general case)

Idea

Encoding

\[(S_1, \ldots, S_k) \rightarrow \bigcup_{i=1}^{k} p \cdot S_i - d_i .\]

plus something extra simulates \( \cup \) and \( + \).
**Problem**

Is addition enough to define something interesting? (general case)

**Idea**

**Encoding**

\[
(S_1, \ldots, S_k) \rightarrow \bigcup_{i=1}^{k} p \cdot S_i - d_i .
\]

plus something extra simulates $\cup$ and $\oplus$.

- unique solution $\varphi(X) = \psi(X)$—recursively-hard ($\cap$, $\cup$, $\oplus$)
Conclusion

- A basic mathematical object.
Conclusion

- A basic mathematical object.
- Using methods of theoretical computer science.
Conclusion

- A basic mathematical object.
- Using methods of theoretical computer science.
- cf. Diophantine equations.
Conclusion

- A basic mathematical object.
- Using methods of theoretical computer science.
- cf. Diophantine equations.

Example

Let PRIMES be the set of all primes.
Conclusion

- A basic mathematical object.
- Using methods of theoretical computer science.
- cf. Diophantine equations.

Example

Let PRIMES be the set of all primes.

1. A Diophantine equation with PRIMES as the range of $x$. 
Conclusion

- A basic mathematical object.
- Using methods of theoretical computer science.
- cf. Diophantine equations.

Example

Let PRIMES be the set of all primes.

1. A Diophantine equation with PRIMES as the range of $x$.
2. An equation over sets of numbers with PRIMES as the unique value of $X$. 

Problem

Construct a set not representable by equations with \{∪, ∩, +\}.
Conclusion

- A basic mathematical object.
- Using methods of theoretical computer science.
- cf. Diophantine equations.

Example

Let PRIMES be the set of all primes.

1. A Diophantine equation with PRIMES as the range of $x$.
2. An equation over sets of numbers with PRIMES as the unique value of $X$.

- Any number-theoretic methods?
Conclusion

- A basic mathematical object.
- Using methods of theoretical computer science.
- cf. Diophantine equations.

Example

Let PRIMES be the set of all primes.

1. A Diophantine equation with PRIMES as the range of $x$.
2. An equation over sets of numbers with PRIMES as the unique value of $X$.

- Any number-theoretic methods?

Problem

*Construct a set not representable by equations with* $\{\cup, \cap, +\}$. 

Artur Jeż (Wrocław)  Equations over sets of natural numbers  May 22, 2008