A simple P-complete problem and its representations by language equations

Alexander Okhotin

University of Turku *and* Academy of Finland

September 13, 2007

Representing hardest sets

- Family of devices.

Representing hardest sets

- Family of devices.

1. Turing machines
2. Linear context-free grammars
3. Trellis automata

Representing hardest sets

- Family of devices.
- Family of languages.

1. Turing machines
2. Linear context-free grammars
3. Trellis automata

Representing hardest sets

- Family of devices.
- Family of languages.

1. Turing machines:

   RE

2. Linear context-free grammars:

   $\subsetneq$ NL

3. Trellis automata:

   $\subsetneq$ P

Representing hardest sets

- Family of devices.
- Family of languages.
- Hardest set.

1. Turing machines:

   RE

2. Linear context-free grammars:

   $\subsetneq$ NL

3. Trellis automata:

   $\subsetneq$ P

Representing hardest sets

- Family of devices.
- Family of languages.
- Hardest set.

1. Turing machines:
   RE-complete
2. Linear context-free grammars:
   NL-complete
3. Trellis automata:
   P-complete

Representing hardest sets

- Family of devices.
- Family of languages.
- Hardest set.
- Its representation.

1. Turing machines:
   RE-complete
2. Linear context-free grammars:
   NL-complete
3. Trellis automata:
   P-complete

Representing hardest sets

- Family of devices.
- Family of languages.
- Hardest set.
- Its representation.
- Succinctness.

1. Turing machines:
   RE-complete
2. Linear context-free grammars:
   NL-complete
3. Trellis automata:
   P-complete

Representing hardest sets

- Family of devices.
- Family of languages.
- Hardest set.
- Its representation.
- Succinctness.

Motivation:

1. Turing machines:
   RE-complete
2. Linear context-free grammars:
   NL-complete
3. Trellis automata:
   P-complete

Representing hardest sets

- Family of devices.
- Family of languages.
- Hardest set.
- Its representation.
- Succinctness.

1. Turing machines:
   RE-complete
2. Linear context-free grammars:
   NL-complete
3. Trellis automata:
   P-complete

Motivation:

- Understanding important models.

Representing hardest sets

- Family of devices.
- Family of languages.
- Hardest set.
- Its representation.
- Succinctness.

1. Turing machines:
   RE-complete
2. Linear context-free grammars:
   NL-complete
3. Trellis automata:
   P-complete

Motivation:

- Understanding important models.
- Pædagogic value.

Representing hardest sets

- Family of devices.
- Family of languages.
- Hardest set.
- Its representation.
- Succinctness.

1. Turing machines:
   RE-complete
2. Linear context-free grammars:
   NL-complete
3. Trellis automata:
   P-complete

Motivation:

- Understanding important models.
- Pædagogic value.

Problem

Find small Boolean grammars for P-complete sets.

Boolean grammars

Context-free grammars: Rules of the form

$$A \to \alpha$$

"If w is generated by α, then w is generated by A".

Boolean grammars

Context-free grammars: Rules of the form

$$A \to \alpha$$

"If w is generated by α, then w is generated by A".

✓ Multiple rules for A: *disjunction*.

Boolean grammars

Context-free grammars: Rules of the form

$$A \to \alpha$$

"If w is generated by α, then w is generated by A".

✓ Multiple rules for A: *disjunction*.

Conjunctive grammars (Okhotin, 2000) Rules of the form

$$A \to \alpha_1 \& \ldots \& \alpha_m$$

"If w is generated by each α_i, then w is generated by A".

Boolean grammars

Context-free grammars: Rules of the form

$$A \to \alpha$$

"If w is generated by α, then w is generated by A".

✓ Multiple rules for A: *disjunction*.

Conjunctive grammars (Okhotin, 2000) Rules of the form

$$A \to \alpha_1 \& \ldots \& \alpha_m$$

"If w is generated by each α_i, then w is generated by A".

Boolean grammars (Okhotin, 2003) Rules of the form

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n$$

"If w is generated by each α_i and by none of β_j, then w is generated by A".

Formal definition

- Quadruple $G = (\Sigma, N, P, S)$, where $S \in N$ and rules in P are

$$A \rightarrow \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \quad \text{with } A \in N,\ \alpha_i, \beta_i \in (\Sigma \cup N)^*$$

Formal definition

- Quadruple $G = (\Sigma, N, P, S)$, where $S \in N$ and rules in P are

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \quad \text{with } A \in N,\ \alpha_i, \beta_i \in (\Sigma \cup N)^*$$

- Language equations for G:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \ \cap \ \bigcap_{j=1}^{n} \overline{\beta_j} \right] \qquad (*)$$

Formal definition

- Quadruple $G = (\Sigma, N, P, S)$, where $S \in N$ and rules in P are

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \quad \text{with } A \in N,\ \alpha_i, \beta_i \in (\Sigma \cup N)^*$$

- Language equations for G:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \ \cap \ \bigcap_{j=1}^{n} \overline{\beta_j} \right] \qquad (*)$$

- A solution: a vector of languages $(\ldots, L_C, \ldots)_{C \in N}$.

Formal definition

- Quadruple $G = (\Sigma, N, P, S)$, where $S \in N$ and rules in P are

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \quad \text{with } A \in N,\ \alpha_i, \beta_i \in (\Sigma \cup N)^*$$

- Language equations for G:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \ \cap\ \bigcap_{j=1}^{n} \overline{\beta_j} \right] \qquad (*)$$

- A solution: a vector of languages $(\ldots, L_C, \ldots)_{C \in N}$.
- Semantics of *strongly unique solution*:

Formal definition

- Quadruple $G = (\Sigma, N, P, S)$, where $S \in N$ and rules in P are

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \quad \text{with } A \in N,\ \alpha_i, \beta_i \in (\Sigma \cup N)^*$$

- Language equations for G:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \ \cap\ \bigcap_{j=1}^{n} \overline{\beta_j} \right] \qquad (*)$$

- A solution: a vector of languages $(\ldots, L_C, \ldots)_{C \in N}$.
- Semantics of *strongly unique solution*:
  - Assume (*) has a unique solution $(\ldots, L_C, \ldots)_{C \in N}$.

Formal definition

- Quadruple $G = (\Sigma, N, P, S)$, where $S \in N$ and rules in P are

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \quad \text{with } A \in N,\ \alpha_i, \beta_i \in (\Sigma \cup N)^*$$

- Language equations for G:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \cap \bigcap_{j=1}^{n} \overline{\beta_j} \right] \qquad (*)$$

- A solution: a vector of languages $(\ldots, L_C, \ldots)_{C \in N}$.
- Semantics of *strongly unique solution*:
  - Assume (*) has a unique solution $(\ldots, L_C, \ldots)_{C \in N}$.
  - Furthermore, for each finite subword-closed $M \subset \Sigma^*$, the solution modulo M is unique.

Formal definition

- Quadruple $G = (\Sigma, N, P, S)$, where $S \in N$ and rules in P are

$$A \to \alpha_1 \& \dots \& \alpha_m \& \neg\beta_1 \& \dots \& \neg\beta_n, \quad \text{with } A \in N,\ \alpha_i, \beta_i \in (\Sigma \cup N)^*$$

- Language equations for G:

$$A = \bigcup_{A \to \alpha_1 \& \dots \& \alpha_m \& \neg\beta_1 \& \dots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \ \cap \ \bigcap_{j=1}^{n} \overline{\beta_j} \right] \qquad (*)$$

- A solution: a vector of languages $(\dots, L_C, \dots)_{C \in N}$.
- Semantics of *strongly unique solution*:
  - Assume (*) has a unique solution $(\dots, L_C, \dots)_{C \in N}$.
  - Furthermore, for each finite subword-closed $M \subset \Sigma^*$, the solution modulo M is unique.
  - Then $L_G(C) := L_C$ for all $C \in N$, and $L(G) := L_G(S)$.

Formal definition

- Quadruple $G = (\Sigma, N, P, S)$, where $S \in N$ and rules in P are

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \quad \text{with } A \in N,\ \alpha_i, \beta_i \in (\Sigma \cup N)^*$$

- Language equations for G:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \cap \bigcap_{j=1}^{n} \overline{\beta_j} \right] \qquad (*)$$

- A solution: a vector of languages $(\ldots, L_C, \ldots)_{C \in N}$.
- Semantics of *strongly unique solution*:
  - Assume (*) has a unique solution $(\ldots, L_C, \ldots)_{C \in N}$.
  - Furthermore, for each finite subword-closed $M \subset \Sigma^*$, the solution modulo M is unique.
  - Then $L_G(C) := L_C$ for all $C \in N$, and $L(G) := L_G(S)$.
- Another semantics: Kountouriotis et al. (DLT 2006).

Properties of Boolean grammars

- Generate $\{a^n b^n c^n \mid n \geqslant 0\}$, $\{ww \mid w \in \{a,b\}^*\}$, $\{a^{2^n} \mid n \geqslant 0\}$, etc.

Example

$$\begin{array}{lcl} S & \to & AB \& \neg DC \\ A & \to & aA \mid \varepsilon \\ B & \to & bBc \mid \varepsilon \\ C & \to & cC \mid \varepsilon \\ D & \to & aDb \mid \varepsilon \end{array}$$

Properties of Boolean grammars

- Generate $\{a^n b^n c^n \mid n \geqslant 0\}$, $\{ww \mid w \in \{a,b\}^*\}$, $\{a^{2^n} \mid n \geqslant 0\}$, etc.

Example

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
S & \to & AB \& \neg DC & S & = & AB \cap \overline{DC} \\
A & \to & aA \mid \varepsilon & A & = & \{a\}A \cup \{\varepsilon\} \\
B & \to & bBc \mid \varepsilon & B & = & \{b\}B\{c\} \cup \{\varepsilon\} \\
C & \to & cC \mid \varepsilon & C & = & \{c\}C \cup \{\varepsilon\} \\
D & \to & aDb \mid \varepsilon & D & = & \{a\}D\{b\} \cup \{\varepsilon\}
\end{array}
$$

Properties of Boolean grammars

- Generate $\{a^n b^n c^n \mid n \geqslant 0\}$, $\{ww \mid w \in \{a,b\}^*\}$, $\{a^{2^n} \mid n \geqslant 0\}$, etc.

Example

S	$\rightarrow$	$AB \& \neg DC$	S	$=$	$AB \cap \overline{DC}$	$\{a^i b^i c^j \mid i \neq j\}$
A	$\rightarrow$	$aA \mid \varepsilon$	A	$=$	$\{a\}A \cup \{\varepsilon\}$	a^*
B	$\rightarrow$	$bBc \mid \varepsilon$	B	$=$	$\{b\}B\{c\} \cup \{\varepsilon\}$	$\{b^j c^j \mid j \geqslant 0\}$
C	$\rightarrow$	$cC \mid \varepsilon$	C	$=$	$\{c\}C \cup \{\varepsilon\}$	c^*
D	$\rightarrow$	$aDb \mid \varepsilon$	D	$=$	$\{a\}D\{b\} \cup \{\varepsilon\}$	$\{a^i b^i \mid i \geqslant 0\}$

Properties of Boolean grammars

- Generate $\{a^n b^n c^n \mid n \geqslant 0\}$, $\{ww \mid w \in \{a,b\}^*\}$, $\{a^{2^n} \mid n \geqslant 0\}$, etc.

Example

$$
\begin{array}{lcl@{\qquad}lcl@{\qquad}l}
S & \rightarrow & AB \& \neg DC & S & = & AB \cap \overline{DC} & \{a^i b^i c^j \mid i \neq j\} \\
A & \rightarrow & aA \mid \varepsilon & A & = & \{a\}A \cup \{\varepsilon\} & a^* \\
B & \rightarrow & bBc \mid \varepsilon & B & = & \{b\}B\{c\} \cup \{\varepsilon\} & \{b^j c^j \mid j \geqslant 0\} \\
C & \rightarrow & cC \mid \varepsilon & C & = & \{c\}C \cup \{\varepsilon\} & c^* \\
D & \rightarrow & aDb \mid \varepsilon & D & = & \{a\}D\{b\} \cup \{\varepsilon\} & \{a^i b^i \mid i \geqslant 0\}
\end{array}
$$

- Languages contained in $DTIME(n^3) \cap DSPACE(n)$.

Properties of Boolean grammars

- Generate $\{a^n b^n c^n \mid n \geqslant 0\}$, $\{ww \mid w \in \{a,b\}^*\}$, $\{a^{2^n} \mid n \geqslant 0\}$, etc.

Example

$$\begin{array}{lcl@{\qquad}lcl@{\qquad}l}
S & \to & AB \& \neg DC & S & = & AB \cap \overline{DC} & \{a^i b^i c^j \mid i \neq j\} \\
A & \to & aA \mid \varepsilon & A & = & \{a\}A \cup \{\varepsilon\} & a^* \\
B & \to & bBc \mid \varepsilon & B & = & \{b\}B\{c\} \cup \{\varepsilon\} & \{b^j c^j \mid j \geqslant 0\} \\
C & \to & cC \mid \varepsilon & C & = & \{c\}C \cup \{\varepsilon\} & c^* \\
D & \to & aDb \mid \varepsilon & D & = & \{a\}D\{b\} \cup \{\varepsilon\} & \{a^i b^i \mid i \geqslant 0\}
\end{array}$$

- Languages contained in $DTIME(n^3) \cap DSPACE(n)$.
- Practical parsing methods: recursive descent, generalized LR.

Properties of Boolean grammars

- Generate $\{a^n b^n c^n \mid n \geqslant 0\}$, $\{ww \mid w \in \{a,b\}^*\}$, $\{a^{2^n} \mid n \geqslant 0\}$, etc.

Example

$$\begin{array}{lcllcll}
S & \to & AB\&\neg DC & S & = & AB \cap \overline{DC} & \{a^i b^j c^j \mid i \neq j\} \\
A & \to & aA \mid \varepsilon & A & = & \{a\}A \cup \{\varepsilon\} & a^* \\
B & \to & bBc \mid \varepsilon & B & = & \{b\}B\{c\} \cup \{\varepsilon\} & \{b^j c^j \mid j \geqslant 0\} \\
C & \to & cC \mid \varepsilon & C & = & \{c\}C \cup \{\varepsilon\} & c^* \\
D & \to & aDb \mid \varepsilon & D & = & \{a\}D\{b\} \cup \{\varepsilon\} & \{a^i b^i \mid i \geqslant 0\}
\end{array}$$

- Languages contained in $DTIME(n^3) \cap DSPACE(n)$.
- Practical parsing methods: recursive descent, generalized LR.
- ✓ Completion of the context-free grammars.

Trellis automata

(one-way real-time cellular automata)

Definition

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

Trellis automata

(one-way real-time cellular automata)

Definition

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

- Σ: input alphabet;

Trellis automata

(one-way real-time cellular automata)

Definition

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

- Σ: input alphabet;
- Q: finite set of states;

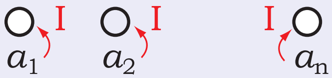
Trellis automata

(one-way real-time cellular automata)

Definition

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

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



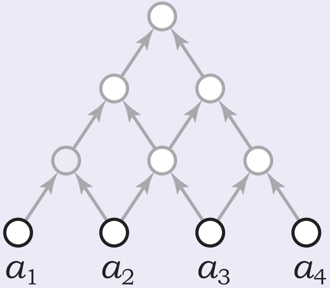
Trellis automata

(one-way real-time cellular automata)

Definition

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

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



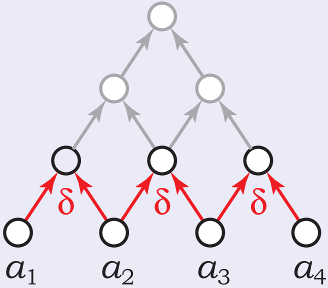
Trellis automata

(one-way real-time cellular automata)

Definition

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

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



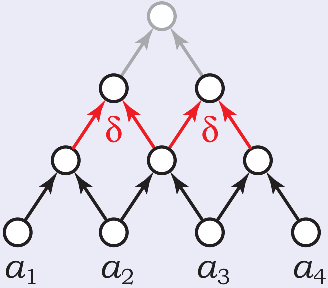
Trellis automata

(one-way real-time cellular automata)

Definition

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

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



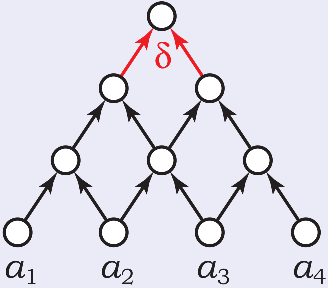
Trellis automata

(one-way real-time cellular automata)

Definition

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

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



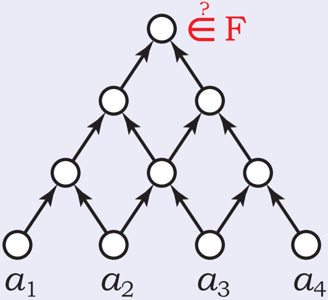
Trellis automata

(one-way real-time cellular automata)

Definition

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

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



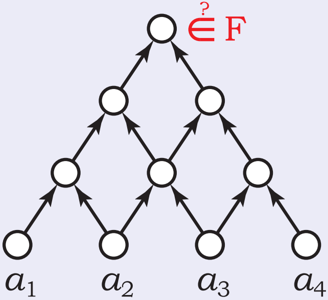
Trellis automata

(one-way real-time cellular automata)

Definition

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

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



- Equivalent to linear conjunctive grammars.

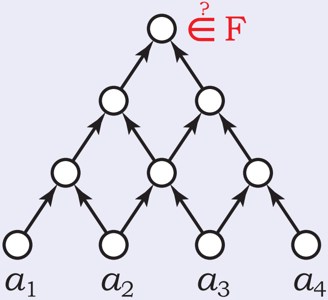
Trellis automata

(one-way real-time cellular automata)

Definition

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

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



- Equivalent to linear conjunctive grammars.
- Closed under $\cup, \cap, \sim$, not closed under concatenation and star.

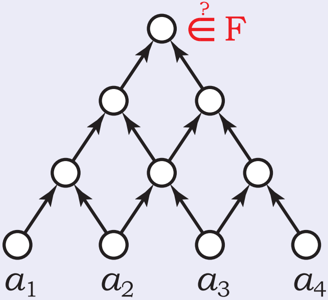
Trellis automata

(one-way real-time cellular automata)

Definition

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

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



- Equivalent to linear conjunctive grammars.
- Closed under $\cup, \cap, \sim$, not closed under concatenation and star.
- Can recognize $\{wcw\}$, $\{a^n b^n c^n\}$, $\{a^n b^{2^n}\}$, VALC.

P-completeness results

Ibarra, Kim (1984): trellis automata recognize P-complete languages.

P-completeness results

Ibarra, Kim (1984): trellis automata recognize P-complete languages.

- Simulate VALC(M).

P-completeness results

Ibarra, Kim (1984): trellis automata recognize P-complete languages.

- Simulate VALC(M).
- No construction given; obviously large.

P-completeness results

Ibarra, Kim (1984): trellis automata recognize P-complete languages.

- Simulate VALC(M).
- No construction given; obviously large.

Okhotin (2003): specifying Circuit Value Problem.

P-completeness results

Ibarra, Kim (1984): trellis automata recognize P-complete languages.

- Simulate VALC(M).
- No construction given; obviously large.

Okhotin (2003): specifying Circuit Value Problem.

- TA: 45 states, 9 symbols.

P-completeness results

Ibarra, Kim (1984): trellis automata recognize P-complete languages.

- Simulate VALC(M).
- No construction given; obviously large.

Okhotin (2003): specifying Circuit Value Problem.

- TA: 45 states, 9 symbols.
- Lin.Conj. grammar: $\leqslant 45^2 \cdot 9^2 + 1 = 164026$ rules.

P-completeness results

Ibarra, Kim (1984): trellis automata recognize P-complete languages.

- Simulate VALC(M).
- No construction given; obviously large.

Okhotin (2003): specifying Circuit Value Problem.

- TA: 45 states, 9 symbols.
- Lin.Conj. grammar: $\leqslant 45^2 \cdot 9^2 + 1 = 164026$ rules.
- Conj. grammar: $\leqslant 45^2 + 9 + 1 = 2035$ rules.

P-completeness results

Ibarra, Kim (1984): trellis automata recognize P-complete languages.

- Simulate VALC(M).
- No construction given; obviously large.

Okhotin (2003): specifying Circuit Value Problem.

- TA: 45 states, 9 symbols.
- Lin.Conj. grammar: $\leqslant 45^2 \cdot 9^2 + 1 = 164026$ rules.
- Conj. grammar: $\leqslant 45^2 + 9 + 1 = 2035$ rules.

Problem

Construct smaller descriptions.

P-complete circuit value problems

Circuit Value Problem (Ladner, 1975)

Given a Boolean circuit with gates $\{\vee, \wedge, \neg\}$
and a vector of input values,
determine whether it evaluates to true.

P-complete circuit value problems

Circuit Value Problem (Ladner, 1975)

Given a Boolean circuit with gates $\{\vee, \wedge, \neg\}$
and a vector of input values,
determine whether it evaluates to true.

Monotone Circuit Value Problem (Goldschlager, 1977)

Given a Boolean circuit with gates $\{\vee, \wedge\}$, $\langle \ldots \rangle$

P-complete circuit value problems

Circuit Value Problem (Ladner, 1975)

Given a Boolean circuit with gates $\{\vee, \wedge, \neg\}$
and a vector of input values,
determine whether it evaluates to true.

Monotone Circuit Value Problem (Goldschlager, 1977)

Given a Boolean circuit with gates $\{\vee, \wedge\}$*,* $\langle \ldots \rangle$

Planar Circuit Value Problem (Goldschlager, 1977)

Given a planar Boolean circuit with gates $\{\vee, \wedge, \neg\}$*,* $\langle \ldots \rangle$

P-complete circuit value problems

Circuit Value Problem (Ladner, 1975)

Given a Boolean circuit with gates $\{\vee, \wedge, \neg\}$
and a vector of input values,
determine whether it evaluates to true.

Monotone Circuit Value Problem (Goldschlager, 1977)

Given a Boolean circuit with gates $\{\vee, \wedge\}$, $\langle \dots \rangle$

Planar Circuit Value Problem (Goldschlager, 1977)

Given a planar Boolean circuit with gates $\{\vee, \wedge, \neg\}$, $\langle \dots \rangle$

- More P-complete variants.

  R. Greenlaw, H. Hoover, W. Ruzzo,
  Limits to parallel computation: P-completeness theory, 1995.

P-complete circuit value problems

Circuit Value Problem (Ladner, 1975)

Given a Boolean circuit with gates $\{\vee, \wedge, \neg\}$
and a vector of input values,
determine whether it evaluates to true.

Monotone Circuit Value Problem (Goldschlager, 1977)

Given a Boolean circuit with gates $\{\vee, \wedge\}$, $\langle \dots \rangle$

Planar Circuit Value Problem (Goldschlager, 1977)

Given a planar Boolean circuit with gates $\{\vee, \wedge, \neg\}$, $\langle \dots \rangle$

- More P-complete variants.

  R. Greenlaw, H. Hoover, W. Ruzzo,
  Limits to parallel computation: P-completeness theory, 1995.

✓ A new variant of CVP.

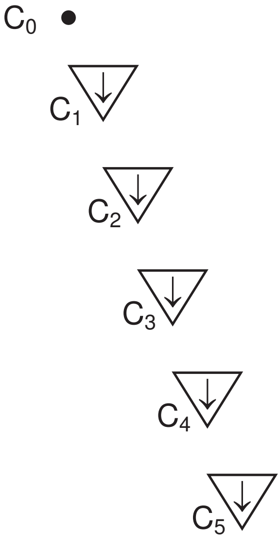
Sequential NOR circuits

- No inputs, root: $C_0 = true$.

Sequential NOR circuits

- No inputs, root: $C_0 = true$.
- One type of gate:

$$x \downarrow y = \neg(x \vee y)$$



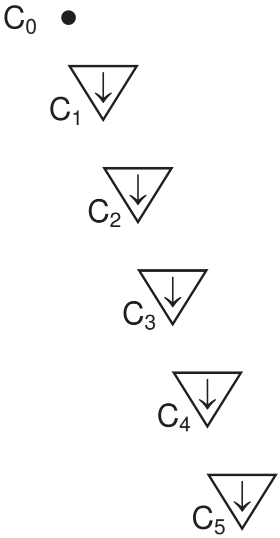
Sequential NOR circuits

- No inputs, root: $C_0 = true$.
- One type of gate:

$$x \downarrow y = \neg(x \vee y)$$

- Each gate:

$$C_i = C_{i-1} \downarrow C_{j_i}$$



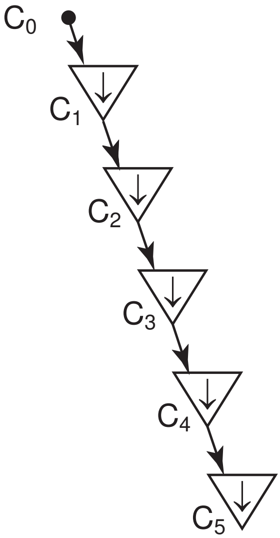
Sequential NOR circuits

- No inputs, root: $C_0 = \mathit{true}$.
- One type of gate:

$$x \downarrow y = \neg(x \vee y)$$

- Each gate:

$$C_i = C_{i-1} \downarrow C_{j_i}$$



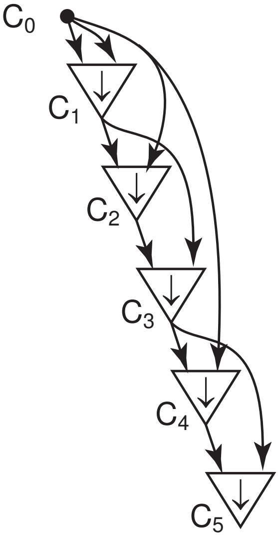
Sequential NOR circuits

- No inputs, root: $C_0 = \mathit{true}$.
- One type of gate:

$$x \downarrow y = \neg(x \vee y)$$

- Each gate:

$$C_i = C_{i-1} \downarrow C_{j_i}$$



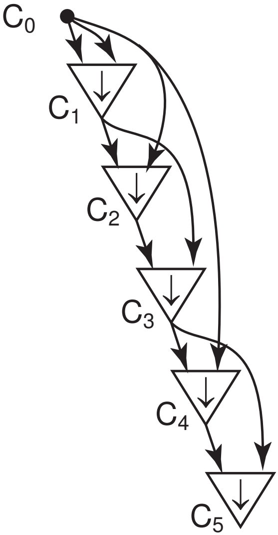
Sequential NOR circuits

- No inputs, root: $C_0 = \mathit{true}$.
- One type of gate:

$$x \downarrow y = \neg(x \vee y)$$

- Each gate:

$$C_i = C_{i-1} \downarrow C_{j_i}$$

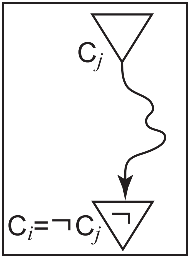


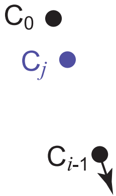
Theorem

Sequential NOR CVP is P-complete.

Reducing CVP to Sequential NOR CVP

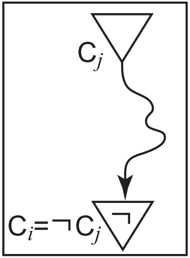
- Simulating negation.

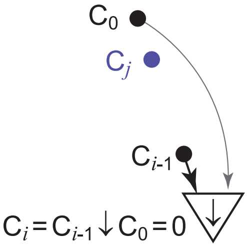




Reducing CVP to Sequential NOR CVP

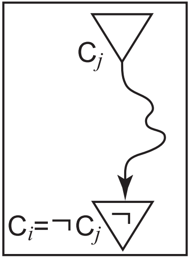
- Simulating negation.

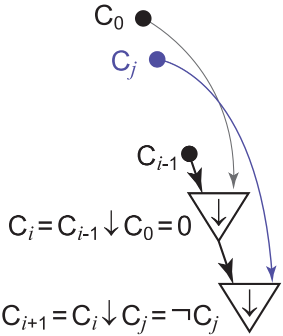




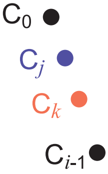
Reducing CVP to Sequential NOR CVP

- Simulating negation.

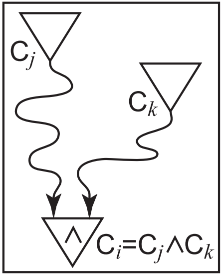




Reducing CVP to Sequential NOR CVP

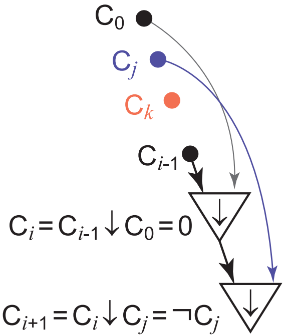


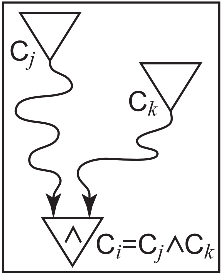
- Simulating negation.
- Simulating conjunction.



Reducing CVP to Sequential NOR CVP

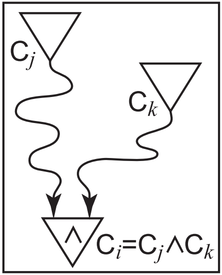
- Simulating negation.
- Simulating conjunction.

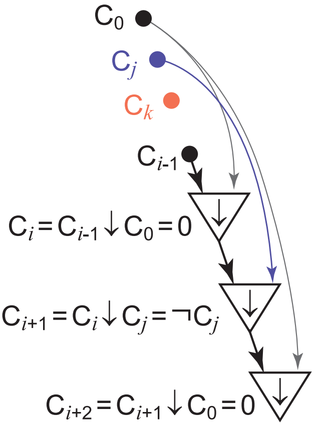




Reducing CVP to Sequential NOR CVP

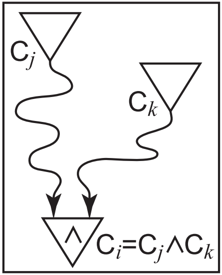
- Simulating negation.
- Simulating conjunction.

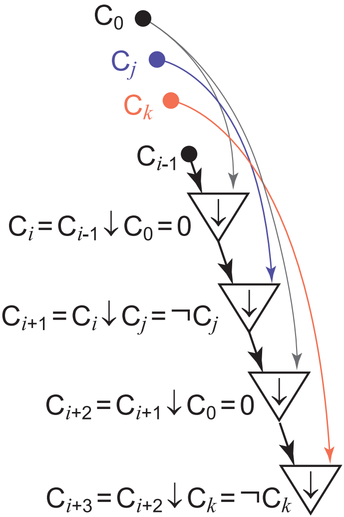




Reducing CVP to Sequential NOR CVP

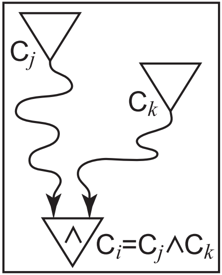
- Simulating negation.
- Simulating conjunction.

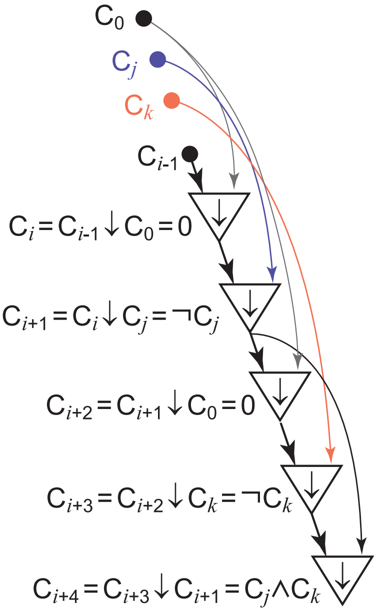




Reducing CVP to Sequential NOR CVP

- Simulating negation.
- Simulating conjunction.





Yes-instances as a formal language

- Alphabet $\{a, b\}$

Yes-instances as a formal language

- Alphabet $\{a, b\}$
- Gate $C_i = C_{i-1} \downarrow C_{j_i}$ represented by $a^{i-j_i-1}b$.

Yes-instances as a formal language

- Alphabet $\{a, b\}$
- Gate $C_i = C_{i-1} \downarrow C_{j_i}$ represented by $a^{i-j_i-1}b$.
- Circuit represented by $C_n \ldots C_2 C_1$.

Yes-instances as a formal language

- Alphabet $\{a, b\}$
- Gate $C_i = C_{i-1} \downarrow C_{j_i}$ represented by $a^{i-j_i-1}b$.
- Circuit represented by $C_n \dots C_2 C_1$.

$$\Big\{a^{n-j_n-1}b \dots a^{2-j_2-1}ba^{1-j_1-1}b \;\Big|\; n \geqslant 0 \text{ and } \exists y_0, y_1, \dots, y_n : \\ y_0 = y_n = 1 \text{ and } \\ \forall i,\ 0 \leqslant j_i < i \text{ and } y_i = \neg(y_{i-1} \vee y_{j_i})\Big\}$$

Representing by a language equation

- ε is a circuit that has value 1.

Representing by a language equation

- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if

Representing by a language equation

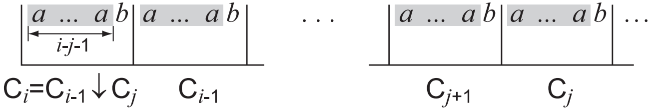
- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if
  1. w is not a circuit that has value 1.

Representing by a language equation

- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if
  1. w is not a circuit that has value 1.
  2. $w = (a^* b)^m u$, where u is not a circuit that has value 1.

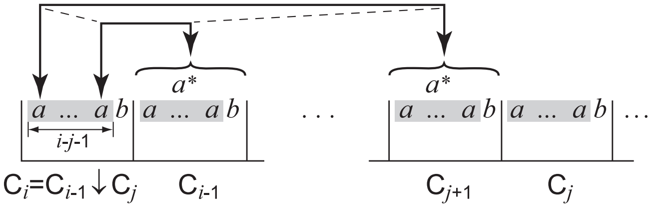
Representing by a language equation

- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if
  1. w is not a circuit that has value 1.
  2. $w = (a^* b)^m u$, where u is not a circuit that has value 1.



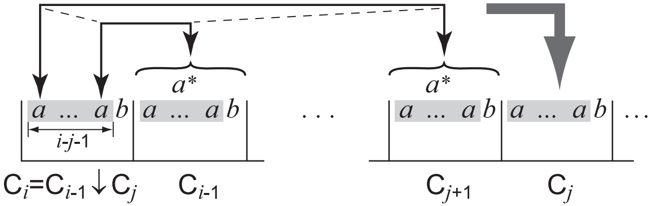
Representing by a language equation

- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if
  1. w is not a circuit that has value 1.
  2. $w = (a^* b)^m u$, where u is not a circuit that has value 1.



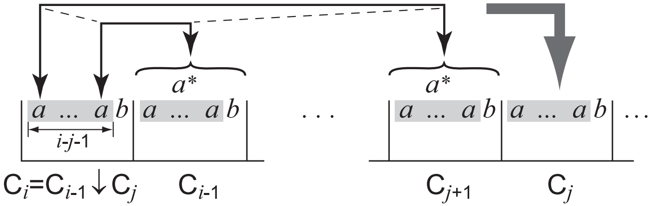
Representing by a language equation

- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if
  1. w is not a circuit that has value 1.
  2. $w = (a^* b)^m u$, where u is not a circuit that has value 1.



Representing by a language equation

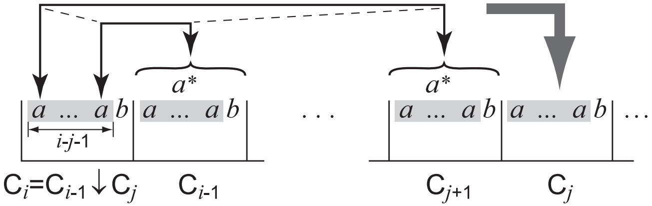
- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if
  1. w is not a circuit that has value 1.
  2. $w = (a^*b)^m u$, where u is not a circuit that has value 1.



- LinCFL L_0 defined by $\{S \to aSAb,\ S \to b,\ A \to aA,\ A \to \varepsilon\}$.

Representing by a language equation

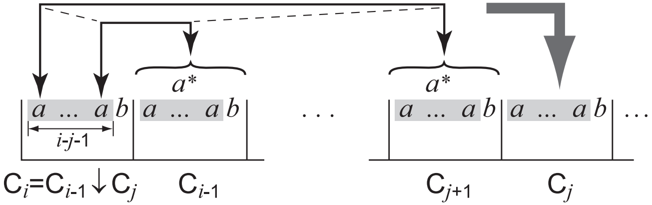
- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if
  1. w is not a circuit that has value 1.
  2. $w = (a^* b)^m u$, where u is not a circuit that has value 1.



- LinCFL L_0 defined by $\{S \to aSAb,\ S \to b,\ A \to aA,\ A \to \varepsilon\}$.
- LinCFL $L_1 = L_0 \cup a^* b$.

Representing by a language equation

- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if
  1. w is not a circuit that has value 1.
  2. $w = (a^* b)^m u$, where u is not a circuit that has value 1.



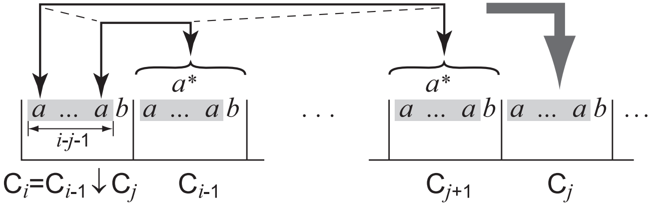
- LinCFL L_0 defined by $\{S \to aSAb,\ S \to b,\ A \to aA,\ A \to \varepsilon\}$.
- LinCFL $L_1 = L_0 \cup a^* b$.

Theorem

Equations $X = \overline{a^* bX} \cap \overline{L_0 X}$ and $X = \overline{L_1 X}$ have P-complete solutions.

Representing by a language equation

- ε is a circuit that has value 1.
- $a^m bw$ has value 1 if and only if
  1. w is not a circuit that has value 1.
  2. $w = (a^*b)^m u$, where u is not a circuit that has value 1.



- LinCFL L_0 defined by $\{S \to aSAb,\ S \to b,\ A \to aA,\ A \to \varepsilon\}$.
- LinCFL $L_1 = L_0 \cup a^*b$.

Theorem

Equations $X = \overline{a^*bX} \cap \overline{L_0X}$ and $X = \overline{L_1X}$ have P-complete solutions.

A small Boolean grammar

- Can express both LinCFGs and equations.

A small Boolean grammar

- Can express both LinCFGs and equations.

Boolean grammar for Sequential NOR CVP

$$
\begin{array}{lll}
S & \to & \neg AbS \& \neg CS \\
A & \to & aA \mid \varepsilon \\
C & \to & aCAb \mid b
\end{array}
$$

A small Boolean grammar

- Can express both LinCFGs and equations.

Boolean grammar for Sequential NOR CVP

$$\begin{aligned} S &\to \neg AbS \& \neg CS \\ A &\to aA \mid \varepsilon \\ C &\to aCAb \mid b \end{aligned}$$

LL(1) Boolean grammar for Sequential NOR CVP

$$\begin{aligned} S &\to E \& \neg AbS \& \neg CS \\ A &\to aA \mid \varepsilon \\ C &\to aCAb \mid b \\ E &\to aE \mid bE \mid \varepsilon \end{aligned}$$

A small Boolean grammar

- Can express both LinCFGs and equations.

Boolean grammar for Sequential NOR CVP

$$
\begin{aligned}
S &\rightarrow \neg AbS \& \neg CS \\
A &\rightarrow aA \mid \varepsilon \\
C &\rightarrow aCAb \mid b
\end{aligned}
$$

LL(1) Boolean grammar for Sequential NOR CVP

$$
\begin{aligned}
S &\rightarrow E \& \neg AbS \& \neg CS \\
A &\rightarrow aA \mid \varepsilon \\
C &\rightarrow aCAb \mid b \\
E &\rightarrow aE \mid bE \mid \varepsilon
\end{aligned}
$$

- Recursive descent parser.

A small Boolean grammar

- Can express both LinCFGs and equations.

Boolean grammar for Sequential NOR CVP

$$\begin{array}{lcl} S & \to & \neg AbS \& \neg CS \\ A & \to & aA \mid \varepsilon \\ C & \to & aCAb \mid b \end{array}$$

LL(1) Boolean grammar for Sequential NOR CVP

$$\begin{array}{lcl} S & \to & E \& \neg AbS \& \neg CS \\ A & \to & aA \mid \varepsilon \\ C & \to & aCAb \mid b \\ E & \to & aE \mid bE \mid \varepsilon \end{array}$$

- Recursive descent parser.
- Time $O(n)$ using memoization.

A small conjunctive grammar

- No explicit negation to express $\neg(x \vee y)$.

A small conjunctive grammar

- No explicit negation to express $\neg(x \vee y)$.
- Two variables for *true* and *false* circuits.

A small conjunctive grammar

- No explicit negation to express $\neg(x \vee y)$.
- Two variables for *true* and *false* circuits.

Conjunctive grammar for Sequential NOR CVP

$$\begin{aligned}
T &\rightarrow AbF \& CF \mid \varepsilon \\
F &\rightarrow AbT \mid CT \\
A &\rightarrow aA \mid \varepsilon \\
C &\rightarrow aCAb \mid b
\end{aligned}$$

A small conjunctive grammar

- No explicit negation to express $\neg(x \vee y)$.
- Two variables for *true* and *false* circuits.

Conjunctive grammar for Sequential NOR CVP

$$\begin{aligned} T &\rightarrow AbF \& CF \mid \varepsilon \\ F &\rightarrow AbT \mid CT \\ A &\rightarrow aA \mid \varepsilon \\ C &\rightarrow aCAb \mid b \end{aligned}$$

Problem

Any LL(k) conjunctive grammars?

The linear conjunctive case

Problem

Construct small trellis automata for P-complete sets.

The linear conjunctive case

Problem

Construct small trellis automata for P-complete sets.

- 45 states, 9 symbols (Okhotin, 2003).

The linear conjunctive case

Problem

Construct small trellis automata for P-complete sets.

- 45 states, 9 symbols (Okhotin, 2003).

Problem

Construct small linear conjunctive grammars.

The linear conjunctive case

Problem

Construct small trellis automata for P-complete sets.

- 45 states, 9 symbols (Okhotin, 2003).

Problem

Construct small linear conjunctive grammars.

- Not much fewer than 164026 rules (Okhotin, 2003).

Research problems

1. Any LL(k) conjunctive grammars for P-complete sets?

Research problems

1. Any LL(k) conjunctive grammars for P-complete sets?
2. Small linear conjunctive grammars?

Research problems

1. Any LL(k) conjunctive grammars for P-complete sets?
2. Small linear conjunctive grammars?
3. Small trellis automata?

Research problems

1. Any LL(k) conjunctive grammars for P-complete sets?
2. Small linear conjunctive grammars?
3. Small trellis automata?

More questions on conjunctive and Boolean grammars.

Research problems

1. Any LL(k) conjunctive grammars for P-complete sets?
2. Small linear conjunctive grammars?
3. Small trellis automata?

More questions on conjunctive and Boolean grammars.

- A list of 9 problems (Okhotin, BEATCS Feb. 2007)

Research problems

1. Any LL(k) conjunctive grammars for P-complete sets?
2. Small linear conjunctive grammars?
3. Small trellis automata?

More questions on conjunctive and Boolean grammars.

- A list of 9 problems (Okhotin, BEATCS Feb. 2007)
- Award of $240 Canadian per problem.

Research problems

1. Any LL(k) conjunctive grammars for P-complete sets?
2. Small linear conjunctive grammars?
3. Small trellis automata?

More questions on conjunctive and Boolean grammars.

- A list of 9 problems (Okhotin, BEATCS Feb. 2007)
- Award of $240 Canadian per problem.
- One problem solved (Jeż, DLT 2007).

Research problems

1. Any LL(k) conjunctive grammars for P-complete sets?
2. Small linear conjunctive grammars?
3. Small trellis automata?

More questions on conjunctive and Boolean grammars.

- A list of 9 problems (Okhotin, BEATCS Feb. 2007)
- Award of \$240 Canadian per problem.
- One problem solved (Jeż, DLT 2007).
- 8 remain open.