Conjunctive grammars, cellular automata and logic

Théo Grente, Étienne Grandjean

AMACC - GREYC - Université Caen Normandie - Normandie Université

Automata 2021, Marseille France

Conjunctive grammars

"Context-free grammars may be thought of as a logic *for inductive description of syntax in which the propositional connectives available. . .are restricted to* disjunction only*."*[Okhotin]

Conjunctive grammars are an extension of context-free grammars by adding an explicit **conjunction** operation within the grammar rules.

An example of conjunctive grammar

The following grammar generates the language $\{a^n b^n c^n \mid n \geq 1\}$, known to not be context-free.

$S \to AB \& DC$
$A \to aA \mid a$
$B \to bBc \mid bc$
$C \to Cc \mid c$
$D \to aDb \mid ab$

$$\underbrace{\{a^i b^j c^k \mid j = k\}}_{L(AB)} \quad \cap \quad \underbrace{\{a^i b^j c^k \mid i = j\}}_{L(DC)} \quad = \quad \underbrace{\{a^n b^n c^n \mid n \geq 1\}}_{L(S)}$$

An example of conjunctive grammar

The following grammar generates the language $\{a^n b^n c^n \mid n \geq 1\}$, known to not be context-free.

$S \rightarrow AB \& DC$
$A \rightarrow aA \mid a$
$B \rightarrow bBc \mid bc$
$C \rightarrow Cc \mid c$
$D \rightarrow aDb \mid ab$

Each rule of a conjunctive grammar $G = (\Sigma, N, P, S)$ is of the form :

$$A \rightarrow \alpha_1 \& ... \& \alpha_m, \text{ for } m \geq 1 \text{ and } \alpha_i \in (\Sigma \cup N)^+$$

Binary normal form

Each conjunctive grammar can be rewritten in an equivalent **binary normal form** (extension of the Chomsky normal form for CFL).

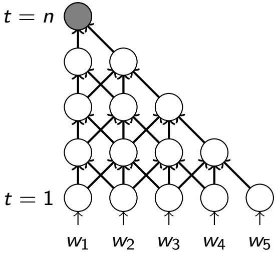
A conjunctive grammar $G = (\Sigma, N, P, S)$ is in *binary normal form* if each rule in P has one of the two following forms :

- a long rule : $A \rightarrow B_1 C_1 \& ... \& B_m C_m$ $(m \geq 1, B_i, C_j \in N)$;
- a short rule : $A \rightarrow a$ $(a \in \Sigma)$.

Real time cellular automata as language recognizers

Cellular automata as word acceptors :

- **Input :** the initial configuration of the CA is only determined by the input word ;
- **Output :** one specific cell called the *output cell* gives the output, "accept" or "reject", of the computation ;
- **Acceptance :** an input word is *accepted* by the CA at time t if the output cell enters an accepting state at time t.

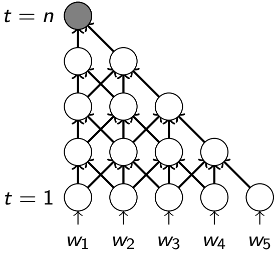


RealTimeCA

Real time cellular automata as language recognizers

A word is **accepted in real time** by a CA if the word is accepted in minimal time for the output cell to receive each of its letters.

A language is **recognized in real time** by a CA if its the set of word that it accepts in real-time.



RealTimeCA

Conjunctive grammars and cellular automata

$$\texttt{LinConj} = \texttt{Trellis}$$

`LinConj` is the **linear** restriction of conjunctive grammars.

`Trellis` is the **one-way** restriction of `RealTimeCA`.

A question and its consequences

Is $\texttt{Conj}$ a subset of $\texttt{RealTimeCA}$?

- $\texttt{Conj} \subseteq \texttt{RealTimeCA}$ would implies that $\texttt{Conj}$ and $\texttt{CFL} \subseteq \texttt{DTIME}(n^2)$.
- $\texttt{Conj} \not\subseteq \texttt{RealTimeCA}$ would implies that either $\texttt{Conj} \subsetneq \texttt{DSPACE}(n)$ or $\texttt{RealTimeCA} \subsetneq \texttt{DSPACE}(n)$.

In this paper we prove two weakened versions of this question.

Overview

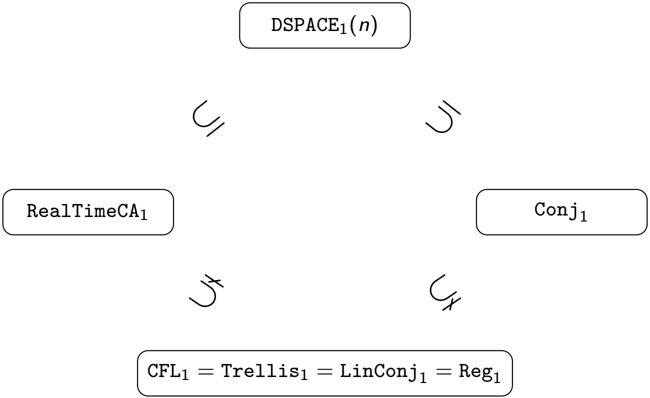
Expression power

Conjunctive grammars over a unary alphabet generate more than regular languages [Jez].

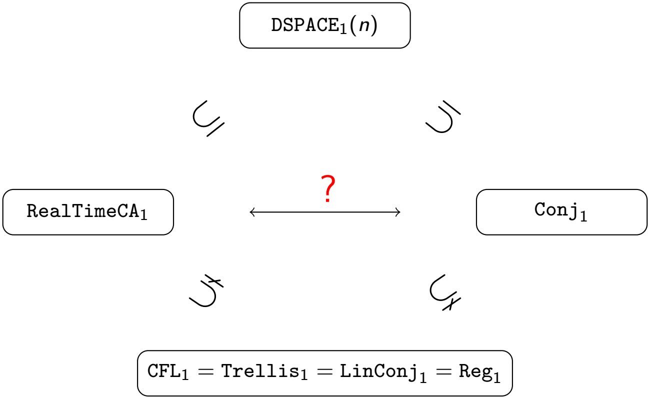
Example of the language $\{a^{4^n} \mid n \geq 0\} \subset \{a\}^+$, generated by the grammar :

$$
\begin{aligned}
A_1 &\rightarrow A_1A_3 \,\&\, A_2A_2 \mid a \\
A_2 &\rightarrow A_1A_1 \,\&\, A_2A_{12} \mid A_{1'}A_{1'} \\
A_3 &\rightarrow A_1A_2 \,\&\, A_{12}A_{12} \mid A_{1'}A_{2'} \\
A_{12} &\rightarrow A_1A_2 \,\&\, A_3A_3 \\
A_{1'} &\rightarrow a \\
A_{2'} &\rightarrow A_{1'}A_{1'}
\end{aligned}
$$

Context



Context



Our result

$$\mathtt{Conj}_1 \subseteq \mathtt{RealTimeCA}_1$$

The inclusion $\mathtt{Conj} \subseteq \mathtt{RealTimeCA}$ holds when restricted to unary languages.

$$\mathtt{DSPACE}_1(n)$$

$$\cup\!|$$

$$\mathtt{RealTimeCA}_1$$

$$\cup\!|$$

$$\mathtt{Conj}_1$$

$$\cup\!\!\not|$$

$$\mathtt{CFL}_1 = \mathtt{Trellis}_1 = \mathtt{LinConj}_1 = \mathtt{Reg}_1$$

Logic as a bridge from grammars to CA

- Computation of CA is **deterministic** $\rightarrow$ **Horn formulae**
- Computation of CA is **local** $\rightarrow$ **predecessor operator**
- Computation on **2 dimensions** (time and space) $\rightarrow$ **2 variables** (with a symmetric role in the logic)

Our logic

$\texttt{pred-ESO-HORN}$ is the set of formulae of the form $\forall x \forall y \psi(x, y)$ where ψ is a **conjunction of Horn clauses** of one the three following forms :

- *an input clause :* $\texttt{min}(x) \wedge (\neg)\,\texttt{min}(y) \wedge Q_s(y) \rightarrow R(x, y)$ with $s \in \Sigma$;
- *a computation clause :* $\delta_1 \wedge \ldots \wedge \delta_r \rightarrow R(x, y)$ where each δ_h is a conjunction $S(x - i, y - j) \wedge x > i \wedge y > j$, with $i, j \geq 0$;
- *a contradiction clause :* $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R(x, y) \rightarrow \perp$.

Expressing unary conjunctive grammars in our logic

Rules of a grammar $G = (\{a\}, N, P, S)$ in binary normal form :

- $A \to B_1 C_1 \& \ldots \& B_m C_m$ $(m \geq 1, B_i, C_j \in N)$;
- $A \to a$.

The grammar is expressed in our logic by using three types of binary predicates :

- $\mathtt{Maj}_A(x, y) \iff \left\lceil \frac{y}{2} \right\rceil \leq x \leq y$ and $a^x \in L(A)$;
- $\mathtt{Min}_A(x, y) \iff \left\lceil \frac{y}{2} \right\rceil \leq x < y$ and $a^{y-x} \in L(A)$;
- $\mathtt{Sum}_{\mathtt{BC}}(x, y) \iff$ there is some x' with $\left\lceil \frac{y}{2} \right\rceil \leq x' \leq x$ such that either $a^{x'} \in L(B)$ and $a^{y-x'} \in L(C)$, or $a^{y-x'} \in L(B)$ and $a^{x'} \in L(C)$.

(x, y) corresponds to the concatenations $a^x a^{y-x}$ and $a^{y-x} a^x$.

Expressing unary conjunctive grammars in our logic

Rules of a grammar $G = (\{a\}, N, P, S)$ in binary normal form :

- $A \to B_1 C_1 \& \ldots \& B_m C_m$ $(m \geq 1, B_i, C_j \in N)$;
- $A \to a$.

(x, y) corresponds to the concatenations $a^x a^{y-x}$ and $a^{y-x} a^x$.

Sample of clauses

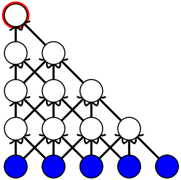
- $\mathtt{Maj}_{B_i}(x,y) \wedge \mathtt{Min}_{C_i}(x,y) \to \mathtt{Sum}_{\mathtt{B_i C_i}}(x,y)$;
- $\mathtt{Min}_{B_i}(x,y) \wedge \mathtt{Maj}_{C_i}(x,y) \to \mathtt{Sum}_{\mathtt{B_i C_i}}(x,y)$;
- $\neg\mathtt{min}(x) \wedge \mathtt{Sum}_{\mathtt{B_i C_i}}(x-1,y) \to \mathtt{Sum}_{\mathtt{B_i C_i}}(x,y)$;
- $x = y \wedge \mathtt{Sum}_{\mathtt{B_1 C_1}}(x,y) \wedge \cdots \wedge \mathtt{Sum}_{\mathtt{B_m C_m}}(x,y) \to \mathtt{Maj}_A(x,y)$.

Equivalence of our logic with real time CA

Logic

Cellular Automata

pred-ESO-HORN



RealTimeCA

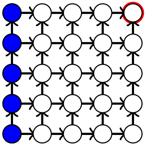
Equivalence of our logic with real time CA

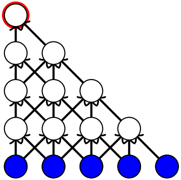
Logic

Grid-Circuit

Cellular Automata

pred-ESO-HORN





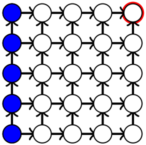
RealTimeCA

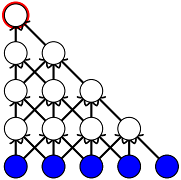
Equivalence of our logic with real time CA

Normalization

Logic → Grid-Circuit   Cellular Automata

pred-ESO-HORN ⇒

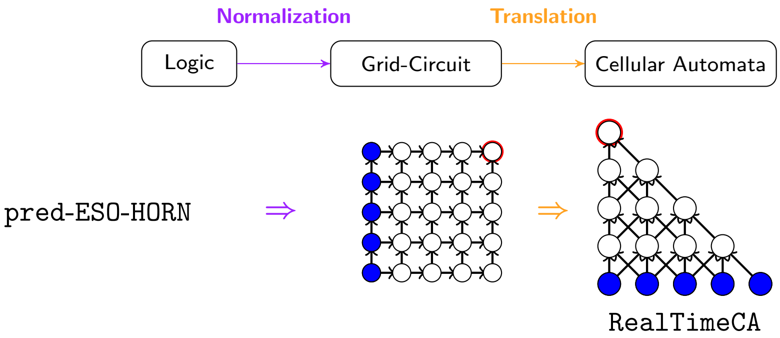




RealTimeCA

- The logic of the grid-circuit corresponds to a normalized version of our starting logic.

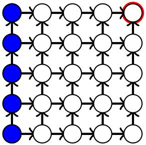
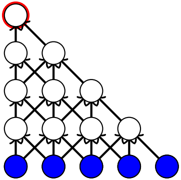
Equivalence of our logic with real time CA



- The logic of the grid-circuit corresponds to a normalized version of our starting logic.
- The computation of the grid-circuit is local and uniform as for a CA ⇒ direct translation of the grid-circuit into a CA.

Equivalence of our logic with real time CA

Logic ⟷ Grid-Circuit ⟷ Cellular Automata

pred-ESO-HORN $\Leftrightarrow$  $\Leftrightarrow$ 

RealTimeCA

- The logic of the grid-circuit corresponds to a normalized version of our starting logic.
- The computation of the grid-circuit is local and uniform as for a CA $\Rightarrow$ direct translation of the grid-circuit into a CA.

Computation example

Grammar

```
A1->A1.A3&A2.A2|a
A2->A1.A1&A2.A6|A'.A'
A3->A1.A2&A6.A6|A''.A'
A6->A1.A2&A3.A3
A'->a
A''->A'.A'
```

```
-:---  a4n.conj       All L7      (Fundamental)
```

Computation example

Grammar → **Formula**

```
min(x)&min(y)->Eq(x,y)
-min(x)&-min(y)&Eq(x-1,y-1)->Eq(x,y)
min(x)&min(y)->MinMin(x,y)
min(x)&-min(y)&MinMin(x,y-1)->Y=2X(x,y)
-min(x)&-min(y)&Y=2X(x-1,y-1)->Y=2X-1(x,y)
-min(x)&-min(y)&Y=2X-1(x,y-1)->Y=2X(x,y)
Y=2X(x,y)->Y<2X(x,y)
Y=2X-1(x,y)->Y<2X(x,y)
-min(x)&Y<2X(x-1,y)->Y<2X(x,y)
-min(y)&Maj{A1}(x,y-1)&Y<2X(x,y)->Maj{A1}(x,y)
-min(y)&Maj{A1}(x,y-1)&Y=2X(x,y)->Min{A1}(x,y)
-min(x)&-min(y)&Min{A1}(x-1,y-1)->Min{A1}(x,y)
Maj{A1}(x,y)&Min{A1}(x,y)->Sum{A1A1}(x,y)
-min(x)&Sum{A1A1}(x-1,y)->Sum{A1A1}(x,y)
Maj{A1}(x,y)&Min{A2}(x,y)->Sum{A1A2}(x,y)
-min(x)&Sum{A1A2}(x-1,y)->Sum{A1A2}(x,y)
Maj{A1}(x,y)&Min{A3}(x,y)->Sum{A1A3}(x,y)
-min(x)&Sum{A1A3}(x-1,y)->Sum{A1A3}(x,y)
Maj{A1}(x,y)&Min{A6}(x,y)->Sum{A1A6}(x,y)
-min(x)&Sum{A1A6}(x-1,y)->Sum{A1A6}(x,y)
Maj{A1}(x,y)&Min{A'}(x,y)->Sum{A'A1}(x,y)
-min(x)&Sum{A'A1}(x-1,y)->Sum{A'A1}(x,y)
Maj{A1}(x,y)&Min{A''}(x,y)->Sum{A''A1}(x,y)
-min(x)&Sum{A''A1}(x-1,y)->Sum{A''A1}(x,y)
-min(x)&Eq(x,y)&Sum{A2A2}(x-1,y)&Sum{A1A3}(x-1,y)->Maj{A1}(x,y)
min(x)&min(y)->Maj{A1}(x,y)
-min(y)&Maj{A2}(x,y-1)&Y<2X(x,y)->Maj{A2}(x,y)
-min(y)&Maj{A2}(x,y-1)&Y=2X(x,y)->Min{A2}(x,y)
-min(x)&-min(y)&Min{A2}(x-1,y-1)->Min{A2}(x,y)
Maj{A2}(x,y)&Min{A1}(x,y)->Sum{A1A2}(x,y)
-min(x)&Sum{A1A2}(x-1,y)->Sum{A1A2}(x,y)
Maj{A2}(x,y)&Min{A2}(x,y)->Sum{A2A2}(x,y)
-min(x)&Sum{A2A2}(x-1,y)->Sum{A2A2}(x,y)
Maj{A2}(x,y)&Min{A3}(x,y)->Sum{A2A3}(x,y)
-min(x)&Sum{A2A3}(x-1,y)->Sum{A2A3}(x,y)
Maj{A2}(x,y)&Min{A6}(x,y)->Sum{A2A6}(x,y)
-min(x)&Sum{A2A6}(x-1,y)->Sum{A2A6}(x,y)
Maj{A2}(x,y)&Min{A'}(x,y)->Sum{A'A2}(x,y)
-min(x)&Sum{A'A2}(x-1,y)->Sum{A'A2}(x,y)
Maj{A2}(x,y)&Min{A''}(x,y)->Sum{A''A2}(x,y)
-min(x)&Sum{A''A2}(x-1,y)->Sum{A''A2}(x,y)
-min(x)&Eq(x,y)&Sum{A'A'}(x-1,y)->Maj{A2}(x,y)
-min(x)&Eq(x,y)&Sum{A2A6}(x-1,y)&Sum{A1A1}(x-1,y)->Maj{A2}(x,y)
-min(y)&Maj{A3}(x,y-1)&Y<2X(x,y)->Maj{A3}(x,y)
-min(y)&Maj{A3}(x,y-1)&Y=2X(x,y)->Min{A3}(x,y)
-min(x)&-min(y)&Min{A3}(x-1,y-1)->Min{A3}(x,y)
Maj{A3}(x,y)&Min{A1}(x,y)->Sum{A1A3}(x,y)
-min(x)&Sum{A1A3}(x-1,y)->Sum{A1A3}(x,y)
Maj{A3}(x,y)&Min{A2}(x,y)->Sum{A2A3}(x,y)
-min(x)&Sum{A2A3}(x-1,y)->Sum{A2A3}(x,y)
Maj{A3}(x,y)&Min{A3}(x,y)->Sum{A3A3}(x,y)
-min(x)&Sum{A3A3}(x-1,y)->Sum{A3A3}(x,y)
Maj{A3}(x,y)&Min{A6}(x,y)->Sum{A3A6}(x,y)
-min(x)&Sum{A3A6}(x-1,y)->Sum{A3A6}(x,y)
Maj{A3}(x,y)&Min{A'}(x,y)->Sum{A'A3}(x,y)
-:---  a4n.pred       Top L1     (Fundamental)
For information about GNU Emacs and the GNU system, type C-h C-a.
```

Computation example

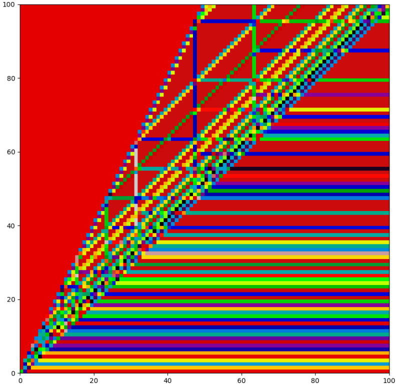
Grammar → Formula→ **Normalized formula**

```
min(x) & min(y) --> Eq(x,y)
min(x) & min(y) --> MinMin(x,y)
min(x) & min(y) --> Maj{A1}(x,y)
min(x) & min(y) --> Maj{A'}(x,y)min(x) & -min(y) & MinMin(x,y-1) --> Y=2X(x,y)
min(x) & -min(y) & MinMin(x,y-1) --> Y<2X(x,y)
-min(y) & Maj{A1}(x,y-1) & min(x) & MinMin(x,y-1) --> Min{A1}(x,y)
-min(y) & Maj{A2}(x,y-1) & min(x) & MinMin(x,y-1) --> Min{A2}(x,y)
-min(y) & Maj{A3}(x,y-1) & min(x) & MinMin(x,y-1) --> Min{A3}(x,y)
-min(y) & Maj{A6}(x,y-1) & min(x) & MinMin(x,y-1) --> Min{A6}(x,y)
-min(y) & Maj{A'}(x,y-1) & min(x) & MinMin(x,y-1) --> Min{A'}(x,y)
-min(y) & Maj{A''}(x,y-1) & min(x) & MinMin(x,y-1) --> Min{A''}(x,y)
-min(y) & Maj{A1}(x,y-1) & min(x) & MinMin(x,y-1) --> Maj{A1}(x,y)
-min(y) & Maj{A2}(x,y-1) & min(x) & MinMin(x,y-1) --> Maj{A2}(x,y)
-min(y) & Maj{A3}(x,y-1) & min(x) & MinMin(x,y-1) --> Maj{A3}(x,y)
-min(y) & Maj{A6}(x,y-1) & min(x) & MinMin(x,y-1) --> Maj{A6}(x,y)
-min(y) & Maj{A'}(x,y-1) & min(x) & MinMin(x,y-1) --> Maj{A'}(x,y)
-min(y) & Maj{A''}(x,y-1) & min(x) & MinMin(x,y-1) --> Maj{A''}(x,y)
-min(y) & Maj{A''}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A'}(x,y-1) --> Sum{A'A''}(x,y)
-min(y) & Maj{A'}(x,y-1) & min(x) & MinMin(x,y-1) --> Sum{A'A'}(x,y)
-min(y) & Maj{A6}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A'}(x,y-1) --> Sum{A'A6}(x,y)
-min(y) & Maj{A3}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A'}(x,y-1) --> Sum{A'A3}(x,y)
-min(y) & Maj{A2}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A'}(x,y-1) --> Sum{A'A2}(x,y)
-min(y) & Maj{A1}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A'}(x,y-1) --> Sum{A'A1}(x,y)
-min(y) & Maj{A''}(x,y-1) & min(x) & MinMin(x,y-1) --> Sum{A''A''}(x,y)
-min(y) & Maj{A6}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A''}(x,y-1) --> Sum{A''A6}(x,y)
-min(y) & Maj{A3}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A''}(x,y-1) --> Sum{A''A3}(x,y)
-min(y) & Maj{A2}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A''}(x,y-1) --> Sum{A''A2}(x,y)
-min(y) & Maj{A1}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A''}(x,y-1) --> Sum{A''A1}(x,y)
-min(y) & Maj{A6}(x,y-1) & min(x) & MinMin(x,y-1) --> Sum{A6A6}(x,y)
-min(y) & Maj{A3}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A6}(x,y-1) --> Sum{A3A6}(x,y)
-min(y) & Maj{A2}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A6}(x,y-1) --> Sum{A2A6}(x,y)
-min(y) & Maj{A1}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A6}(x,y-1) --> Sum{A1A6}(x,y)
-min(y) & Maj{A3}(x,y-1) & min(x) & MinMin(x,y-1) --> Sum{A3A3}(x,y)
-min(y) & Maj{A2}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A3}(x,y-1) --> Sum{A2A3}(x,y)
-min(y) & Maj{A1}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A3}(x,y-1) --> Sum{A1A3}(x,y)
-min(y) & Maj{A2}(x,y-1) & min(x) & MinMin(x,y-1) --> Sum{A2A2}(x,y)
-min(y) & Maj{A1}(x,y-1) & min(x) & MinMin(x,y-1) & Maj{A2}(x,y-1) --> Sum{A1A2}(x,y)
-min(y) & Maj{A1}(x,y-1) & min(x) & MinMin(x,y-1) --> Sum{A1A1}(x,y)-min(x) & Y<2X(x-1,y) --> Y<2X(x,y)
-min(x) & Sum{A1A1}(x-1,y) --> Sum{A1A1}(x,y)
-min(x) & Sum{A1A2}(x-1,y) --> Sum{A1A2}(x,y)
-min(x) & Sum{A1A3}(x-1,y) --> Sum{A1A3}(x,y)
-min(x) & Sum{A1A6}(x-1,y) --> Sum{A1A6}(x,y)
-min(x) & Sum{A'A1}(x-1,y) --> Sum{A'A1}(x,y)
-min(x) & Sum{A''A1}(x-1,y) --> Sum{A''A1}(x,y)
-min(x) & Sum{A2A2}(x-1,y) --> Sum{A2A2}(x,y)
-min(x) & Sum{A2A3}(x-1,y) --> Sum{A2A3}(x,y)
-min(x) & Sum{A2A6}(x-1,y) --> Sum{A2A6}(x,y)
-min(x) & Sum{A'A2}(x-1,y) --> Sum{A'A2}(x,y)
-min(x) & Sum{A''A2}(x-1,y) --> Sum{A''A2}(x,y)
-min(x) & Sum{A3A3}(x-1,y) --> Sum{A3A3}(x,y)
-min(x) & Sum{A3A6}(x-1,y) --> Sum{A3A6}(x,y)
-min(x) & Sum{A'A3}(x-1,y) --> Sum{A'A3}(x,y)
-min(x) & Sum{A''A3}(x-1,y) --> Sum{A''A3}(x,y)
-min(x) & Sum{A6A6}(x-1,y) --> Sum{A6A6}(x,y)
-min(x) & Sum{A'A6}(x-1,y) --> Sum{A'A6}(x,y)
-:---  a4n.grid       Top L54    (Fundamental)
Overwrite mode disabled in current buffer
```

Computation example

Grammar → Formula→ Normalized formula → **Grid circuit**

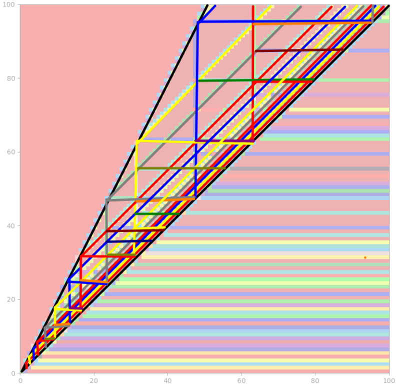
$$
\begin{aligned}
A_1 &\to A_1A_3 \,\&\, A_2A_2 \mid a \\
A_2 &\to A_1A_1 \,\&\, A_2A_{12} \mid A_{1'}A_{1'} \\
A_3 &\to A_1A_2 \,\&\, A_{12}A_{12} \mid A_{1'}A_{2'} \\
A_{12} &\to A_1A_2 \,\&\, A_3A_3 \\
A_{1'} &\to a \\
A_{2'} &\to A_{1'}A_{1'}
\end{aligned}
$$



Computation example

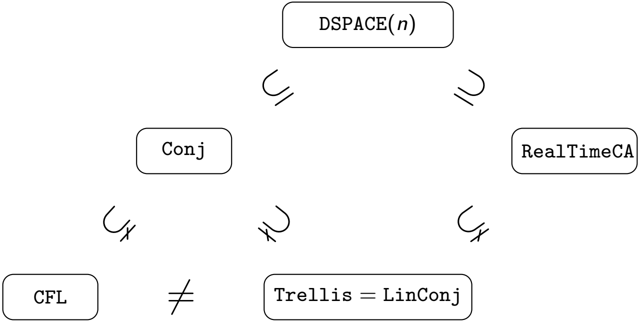
Grammar → Formula→ Normalized formula → Grid circuit→ CA

$$
\begin{aligned}
A_1 &\to A_1 A_3 \,\&\, A_2 A_2 \mid a \\
A_2 &\to A_1 A_1 \,\&\, A_2 A_{12} \mid A_{1'} A_{1'} \\
A_3 &\to A_1 A_2 \,\&\, A_{12} A_{12} \mid A_{1'} A_{2'} \\
A_{12} &\to A_1 A_2 \,\&\, A_3 A_3 \\
A_{1'} &\to a \\
A_{2'} &\to A_{1'} A_{1'}
\end{aligned}
$$

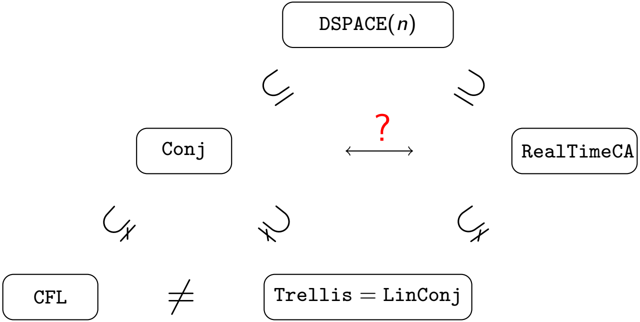


Overview

Context



Context



Our result

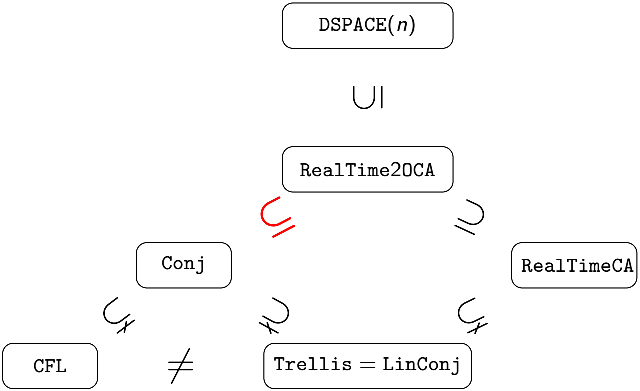
$$\texttt{Conj} \subseteq \texttt{RealTime2OCA}$$

`RealTime2OCA` : real time of 2 dimensional one-way cellular automata

Our result

$$\texttt{Conj} \subseteq \texttt{RealTime2OCA}$$

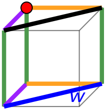
RealTime2OCA : real time of 2 dimensional one-way cellular automata



The method

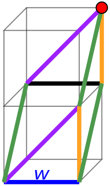
Logic ⟷ Cube-Circuit ⟷ Cellular Automata

incl-pred-ESO-HORN ⇔



Cube

⇔



RealTime2OCA

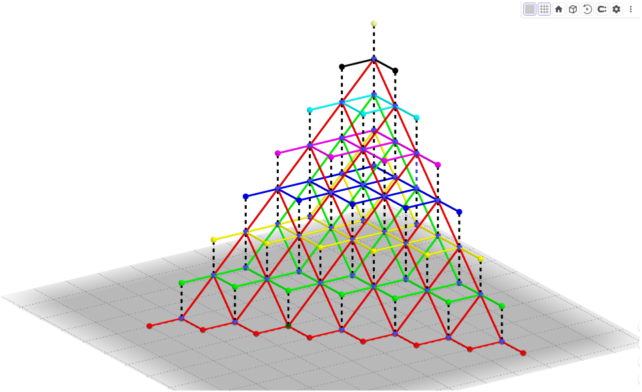
Remarks on the logic

- Conjunction of Horn clauses ;
- 3 variables with asymmetric roles : 2 variables for an induction on intervals, 1 for predecessor induction.

$$([x+a, y-b], z-c) \to ([x, y], z)$$

- Expressing conjunctive grammars : (x, y, z) corresponds to the concatenations $w_x \dots w_{x+z-1} w_{x+z} \dots w_y$ and $w_x \dots w_{y-z} w_{y-z+1} \dots w_y$.

Signals diagram



Overview

Conclusion

- Two inclusions : $\mathtt{Conj}_1 \subseteq \mathtt{RealTimeCA}_1$ and $\mathtt{Conj} \subseteq \mathtt{RealTime2OCA}$.
- The grid : natural way to see the induction of the problem.
- Use of logic to program cellular automata.

Open questions

- The question whether $\texttt{Conj} \subseteq \texttt{RealTimeCA}$ or not is still open.
- Better understanding of the expressive power of conjunctive grammars.
- Exact characterizations of $\texttt{Conj}$? Through logic ? Through computational complexity ?