Conjunctive grammars, cellular automata and logic

Théo Grente
Joint work with Étienne Grandjean and Véronique Terrier

LACL - Université Paris Est Créteil

December 7, 2021

Overview

Introduction and results

The proof method

Expressing conjunctive grammars in our logic

Over a general alphabet

Conclusion

Conjunctive grammars

A statement by Okhotin :
"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."

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.

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

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

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.

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

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

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

Expressivity

The expressive power of conjunctive grammars is largely unknown, even over a unary alphabet.

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

Over a unary alphabet

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

$$\begin{array}{rcl} A_1 & \rightarrow & A_1A_3 \,\&\, A_2A_2 \mid a \\ A_2 & \rightarrow & A_1A_1 \,\&\, A_2A_{12} \mid aa \\ A_3 & \rightarrow & A_1A_2 \,\&\, A_{12}A_{12} \mid aaa \\ A_{12} & \rightarrow & A_1A_2 \,\&\, A_3A_3 \end{array}$$

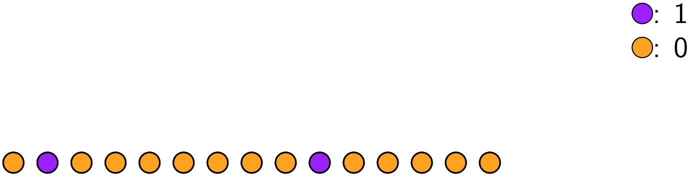
We have a generic method to conceive any unary conjunctive grammar generating the language $\{a^{k^n} \mid n \geq 0\} \subset \{a\}^+$ with $k \geq 4$.

Cellular Automata

- Cellular automaton : ribbon of cells.

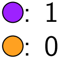
Cellular Automata

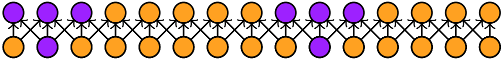
- Cellular automaton : ribbon of cells.
- Each cell is in a given state, evolving over time according to the states of its neighbors.



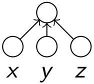
Cellular Automata

- Cellular automaton : ribbon of cells.
- Each cell is in a given state, evolving over time according to the states of its neighbors.
- A local transition function is applied in a synchronous way to each cell and at each time step.
- The computation of the automaton is represented by a space-time diagramm.



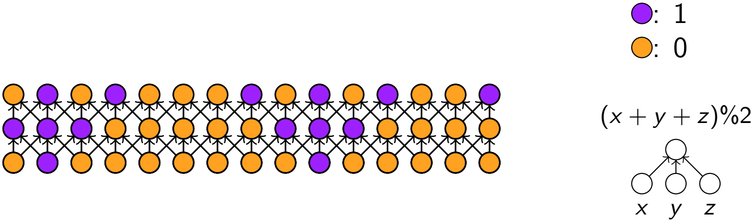


$(x + y + z)\%2$



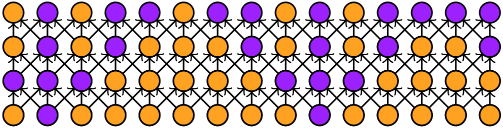
Cellular Automata

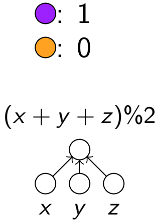
- Cellular automaton : ribbon of cells.
- Each cell is in a given state, evolving over time according to the states of its neighbors.
- A local transition function is applied in a synchronous way to each cell and at each time step.
- The computation of the automaton is represented by a space-time diagramm.



Cellular Automata

- Cellular automaton : ribbon of cells.
- Each cell is in a given state, evolving over time according to the states of its neighbors.
- A local transition function is applied in a synchronous way to each cell and at each time step.
- The computation of the automaton is represented by a space-time diagramm.

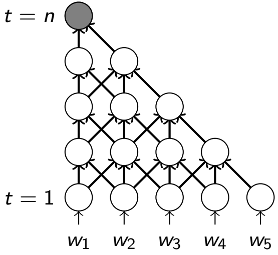




Real time cellular automata as language recognizers

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.

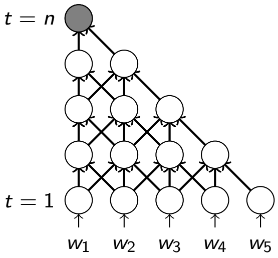


RealTimeCA

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 $\mathtt{Conj}$ a subset of $\mathtt{RealTimeCA}$?

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

Two results

We have proved two weakened versions of this question.

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

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

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

$\texttt{RealTime2OCA}$: real time of 2 dimensional one-way cellular automata

Overview

Introduction and results

The proof method

Expressing conjunctive grammars in our logic

Over a general alphabet

Conclusion

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 $\exists \mathrm{R} \forall x \forall y \psi(x, y)$ where:

- R is a finite set of binary predicates;
- ψ is a conjunction of Horn clauses of the form $\delta_1 \wedge \ldots \wedge \delta_r \rightarrow \delta_0$

where δ_0 is either an atom $R(x, y)$ or $\perp$ and where each δ_i is:

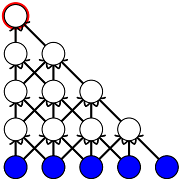
- either an *input litteral* of one of the forms:
  - $Q_s(x-a)$, $Q_s(y-a)$ pour $s \in \Sigma$,
  - $(\neg)U(x-a)$ ou $(\neg)U(y-a)$, pour $U \in \{\texttt{min}, \texttt{max}\}$,
- either a *computation atom* or a *computation conjunction* :
  - $S(x, y)$;
  - $S(x-a, y-b) \wedge x > a \wedge y > b$.

Equivalence of our logic with real time CA

Logic

Cellular Automata

pred-ESO-HORN



RealTimeCA

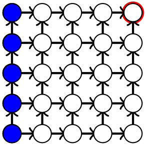
Equivalence of our logic with real time CA

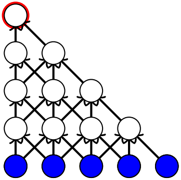
Logic

Grid-Circuit

Cellular Automata

pred-ESO-HORN



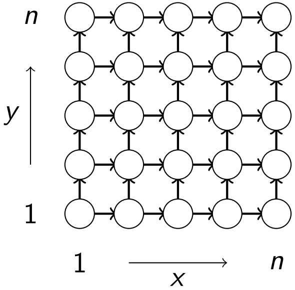


RealTimeCA

The grid-circuit

For an input word of size n:

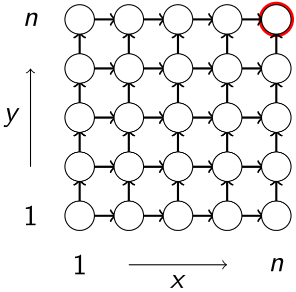
- Grid of $n \times n$ cells, each being in a given **state**.
- The state of the cell (x, y) only depends of the states of the cells $(x-1, y)$ and $(x, y-1)$.



The grid-circuit

For an input word of size n:

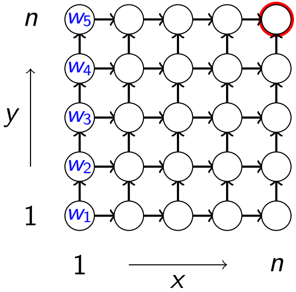
- Grid of $n \times n$ cells, each being in a given **state**.
- The state of the cell (x, y) only depends of the states of the cells $(x-1, y)$ and $(x, y-1)$.
- The output is read on the cell (n, n).

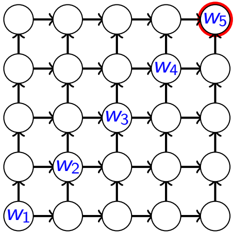


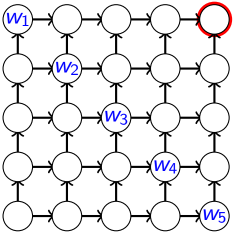
The grid-circuit

For an input word of size n:

- Grid of $n \times n$ cells, each being in a given **state**.
- The state of the cell (x, y) only depends of the states of the cells $(x-1, y)$ and $(x, y-1)$.
- The output is read on the cell (n, n).
- 3 natural ways to fed the input.



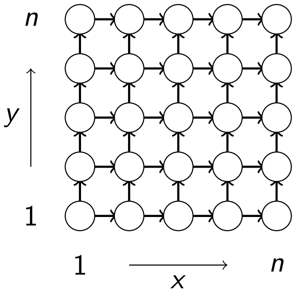




A logic for the grid-circuit ?

Horn formula on two variables, with each clause being:

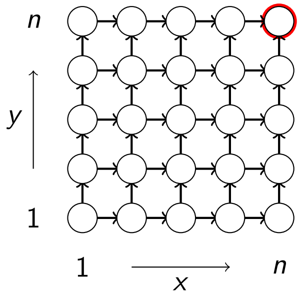
- either a computation clause: $\delta_1 \wedge \cdots \wedge \delta_h \to R(x, y)$ where each hypotheses δ_i is a conjunction $x > 1 \wedge T(x-1, y)$ or $y > 1 \wedge S(x, y-1)$;



A logic for the grid-circuit ?

Horn formula on two variables, with each clause being:

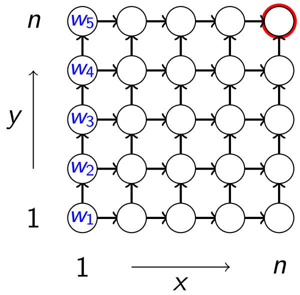
- either a computation clause: $\delta_1 \wedge \cdots \wedge \delta_h \rightarrow R(x,y)$ where each hypotheses δ_i is a conjunction $x > 1 \wedge T(x-1,y)$ or $y > 1 \wedge S(x,y-1)$;
- either a contradiction clause: $x = n \wedge y = n \wedge R(x,y) \rightarrow \perp$;



A logic for the grid-circuit ?

Horn formula on two variables, with each clause being:

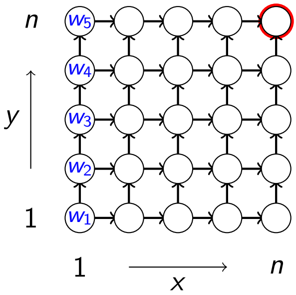
- either a computation clause: $\delta_1 \wedge \cdots \wedge \delta_h \to R(x,y)$ where each hypotheses δ_i is a conjunction $x > 1 \wedge T(x-1,y)$ or $y > 1 \wedge S(x,y-1)$;
- either a contradiction clause: $x = n \wedge y = n \wedge R(x,y) \to \bot$;
- either an input clause: $x = 1 \wedge Q_s(y) \to R(x,y)$.



A logic for the grid-circuit ?

Horn formula on two variables, with each clause being:

- either a computation clause: $\delta_1 \wedge \cdots \wedge \delta_h \to R(x, y)$ where each hypotheses δ_i is a conjunction $x > 1 \wedge T(x-1, y)$ or $y > 1 \wedge S(x, y-1)$;
- either a contradiction clause: $x = n \wedge y = n \wedge R(x, y) \to \perp$;
- either an input clause: $x = 1 \wedge Q_s(y) \to R(x, y)$.

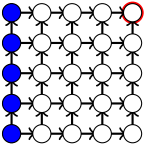


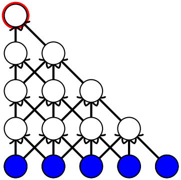
state of site (x, y) = set of binary predicates true on (x, y)

Equivalence of our logic with real time CA

Logic | Grid-Circuit | Cellular Automata

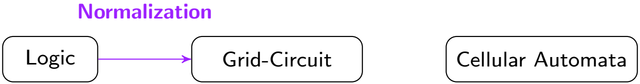
pred-ESO-HORN



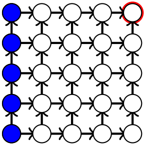


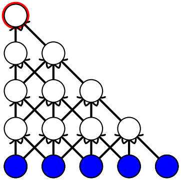
RealTimeCA

Equivalence of our logic with real time CA



pred-ESO-HORN $\Rightarrow$

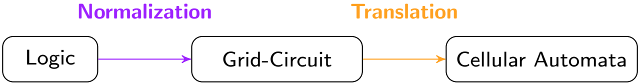




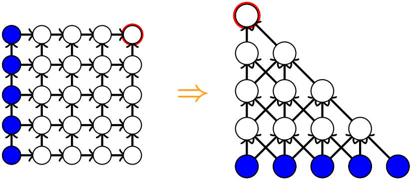
RealTimeCA

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

Equivalence of our logic with real time CA



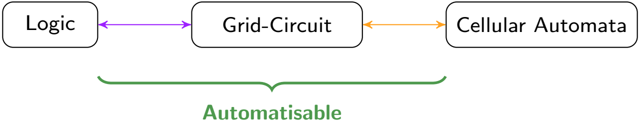
pred-ESO-HORN $\Rightarrow$



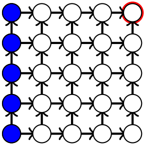
RealTimeCA

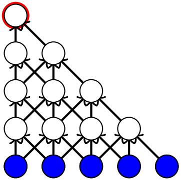
- The logic of the grid-circuit corresponds to a normalized version of our starting logic.
- The computation of the grid-circuit is local and uniform as for a CA $\Rightarrow$ direct translation of the grid-circuit into a CA.

Equivalence of our logic with real time CA



pred-ESO-HORN

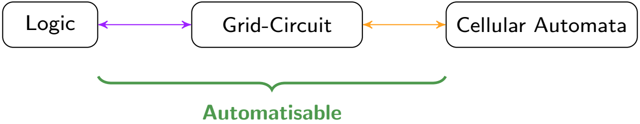




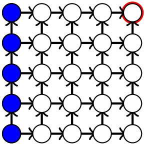
RealTimeCA

- 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.

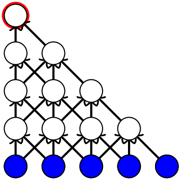
Equivalence of our logic with real time CA



`pred-ESO-HORN` $\Leftrightarrow$



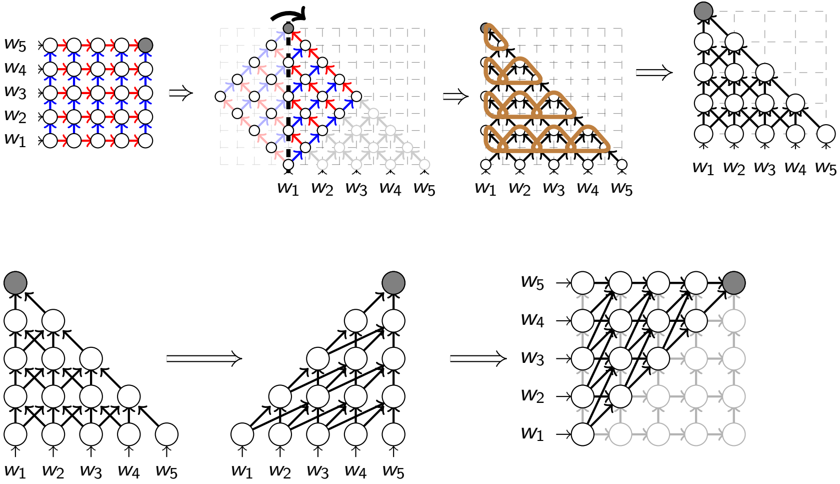
$\Leftrightarrow$



`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.

Equivalence between Grid-circuit and CA



Overview

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

Example of a binary normal form

Binary normal form of the grammar generating the language $\{a^{4^n} \mid n \geq 0\} \subset \{a\}^+$:

$$
\begin{array}{rcl}
A_1 & \rightarrow & A_1 A_3 \,\&\, A_2 A_2 \mid a \\
A_2 & \rightarrow & A_1 A_1 \,\&\, A_2 A_{12} \mid A_{1'} A_{1'} \\
A_3 & \rightarrow & A_1 A_2 \,\&\, A_{12} A_{12} \mid A_{1'} A_{2'} \\
A_{12} & \rightarrow & A_1 A_2 \,\&\, A_3 A_3 \\
A_{1'} & \rightarrow & a \\
A_{2'} & \rightarrow & A_{1'} A_{1'}
\end{array}
$$

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:

- $\mathtt{Maj}_A(x, y) \iff \lceil \frac{y}{2} \rceil \leq x \leq y$ and $a^x \in L(A)$;
- $\mathtt{Min}_A(x, y) \iff \lceil \frac{y}{2} \rceil \leq x < y$ and $a^{y-x} \in L(A)$;
- $\mathtt{Sum}_{\mathtt{BC}}(x, y) \iff$ there is some x' with $\lceil \frac{y}{2} \rceil \leq x' \leq x$ 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:

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

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

Sample of clauses

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

Our result

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

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

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

$$\rotatebox{90}{$\subseteq$}$$

$$\boxed{\mathtt{RealTimeCA}_1}$$

$$\rotatebox{90}{$\subseteq$}$$

$$\boxed{\mathtt{Conj}_1}$$

$$\rotatebox{90}{$\subsetneq$}$$

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

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 $\rightarrow$ **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

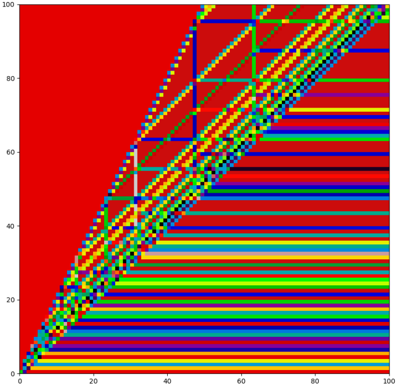
Grammar $\rightarrow$ Formula$\rightarrow$ **Normalized formula**

```
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 $\to$ Formula$\to$ Normalized formula $\to$ **Grid circuit**

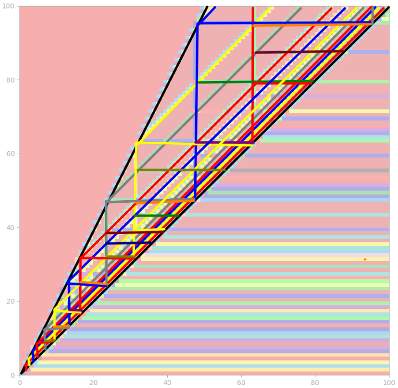
$$
\begin{array}{lcl}
A_1 & \to & A_1 A_3 \,\&\, A_2 A_2 \mid a \\
A_2 & \to & A_1 A_1 \,\&\, A_2 A_{12} \mid A_{1'} A_{1'} \\
A_3 & \to & A_1 A_2 \,\&\, A_{12} A_{12} \mid A_{1'} A_{2'} \\
A_{12} & \to & A_1 A_2 \,\&\, A_3 A_3 \\
A_{1'} & \to & a \\
A_{2'} & \to & A_{1'} A_{1'}
\end{array}
$$



Computation example

Grammar $\to$ Formula$\to$ Normalized formula $\to$ Grid circuit$\to$ CA

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

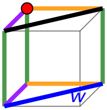


Overview

The method

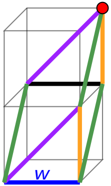
Logic ⟷ Cube-Circuit ⟷ Cellular Automata

`incl-pred-ESO-HORN` $\Leftrightarrow$



`Cube`

$\Leftrightarrow$



`RealTime2OCA`

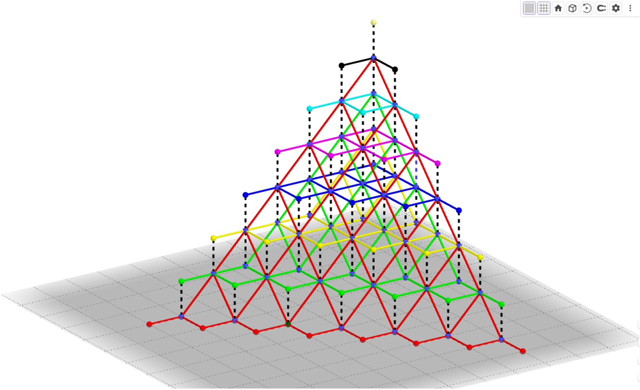
Remarks on the logic

- 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



Our result

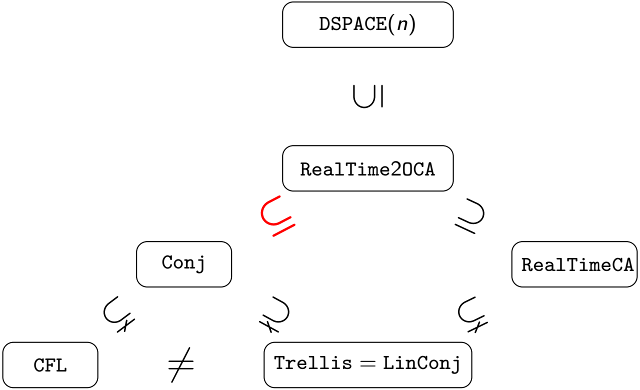
$$\mathtt{Conj} \subseteq \mathtt{RealTime2OCA}$$

`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



Overview

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 ?

Take home message

- Conjunctive grammars seem very interesting by their link to logic but their expressive power is still largely unknown.
- 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.
- Method of proof: use of logic to program cellular automata.