

Inductive definitions in logic versus programs of real-time cellular automata

Etienne Grandjean, Théo Grente, Véronique Terrier

▶ **To cite this version:**

Etienne Grandjean, Théo Grente, Véronique Terrier. Inductive definitions in logic versus programs of real-time cellular automata. Theoretical Computer Science, 2024, 987, pp.1-69. hal-02474520

HAL Id: hal-02474520

https://normandie-univ.hal.science/hal-02474520

Submitted on 11 Feb 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Inductive definitions in logic versus programs of real-time cellular automata [1]

Étienne Grandjean, Théo Grente, Véronique Terrier
Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 CAEN, France

Contents

1 Introduction **2**

2 Preliminaries **6**
2.1 Cellular automata and real-time complexity 6
2.2 Our logics 8
2.3 Our main results 10

3 First examples: express problems; translate formulas into automata **10**
3.1 Three examples of problems expressed in our three Horn logics 10
3.2 Normalizing and translating a formula into a cellular automaton: an introduction example 12

4 Normalizing our Horn logics **17**
4.1 Normalizing predecessor Horn logic 18
4.2 Normalizing predecessor Horn logic with diagonal input-output 24
4.3 Normalizing inclusion Horn logic 25

5 Extending our logics with negation and normalizing them **28**
5.1 Proof of Lemma 4: Normalizing predecessor logics with negation 29
5.2 Proof of Lemma 5: Normalizing inclusion logic with negation 31

6 Equivalence between our logics and real-time cellular automata **34**
6.1 Logical characterization of $\texttt{RealTime}_{\texttt{CA}}$ 34
6.2 Logical characterization of $\texttt{RealTime}_{\texttt{IA}}$ 36
6.3 Logical characterization of `Trellis` and linear conjunctive grammars 38

7 Programming some reference problems in our logics **40**
7.1 Defining in predecessor Horn logic the product of integers 40
7.2 Expressing in logic the set of primes by Fischer's algorithm 41
7.3 Dyck languages, inclusion inductive logic and linear conjunctive grammars 43

8 The language of Čulik and the Firing Squad Synchronization Problem **47**
8.1 Construction of the potential tree 49
8.2 An inclusion inductive formula defining the language `Culik` 54

9 Conclusion **59**

Inductive definitions in logic versus programs of real-time cellular automata [1]

Étienne Grandjean, Théo Grente, Véronique Terrier
Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 CAEN, France

Abstract

Descriptive complexity provides intrinsic, that is, *machine-independent*, characterizations of the major complexity classes. On the other hand, logic can be useful for designing programs in a natural declarative way. This is particularly important for parallel computation models such as cellular automata, because designing parallel programs is considered a difficult task.

This paper establishes three logical characterizations of the three classical complexity classes modeling minimal time, called *real-time*, of one-dimensional cellular automata according to their canonical variants: unidirectional or bidirectional communication, input word given in a parallel or sequential way.

Our three logics are natural restrictions of *existential second-order Horn logic* with built-in *successor* and *predecessor* functions. These logics correspond exactly to the three ways of deciding a language on a *square grid circuit* of side n according to one of the three canonical locations of an input word of length n: along a side of the grid, on the diagonal that contains the output cell, or on the diagonal opposite to the output cell.

The key ingredient of our results is a *normalization* method that transforms a formula from one of our three logics into an equivalent normalized formula that faithfully mimics a grid circuit.

Then, we extend our logics by allowing a limited use of *negation* on hypotheses like in *Stratified Datalog*. By revisiting in detail a number of representative classical problems - recognition of the set of primes by Fisher's algorithm, Dyck language recognition, Firing Squad Synchronization problem, etc. - we show that this extension makes easier programming and we prove that it does not change the complexity of our logics in real-time.

Finally, starting from our experience in expressing those representative problems in logic, we argue that our logics are high-level programming languages: they allow to express in a natural, precise and synthetic way the algorithms of literature, based on signals, and to translate them automatically into cellular automata of the same complexity.

Keywords: computational complexity, descriptive complexity, cellular automata, real-time computation, inductive logic, programming

1. Introduction

Descriptive complexity and programming

There are two criteria for a complexity class: it contains a number of "natural" problems that are *complete* in the class; it has *machine-independent* "natural" characterizations, usually in *logic*, i.e., in so-called *descriptive complexity*. The most famous example is Fagin's Theorem [11, 25], which characterizes NP as the class of problems definable in *existential second-order logic* (ESO). Similarly, Immerman-Vardi's Theorem [25, 21] and Grädel's Theorem [13, 14] characterize the class P by *first-order logic plus least fixed-point*, and *second-order logic restricted to Horn formulas*, respectively.

Another interest of descriptive complexity is that it allows to automatically derive from a logical description of a problem a program that solves it. This is particularly interesting for the design of *parallel programs* that is considered a difficult task [24, 30]. A number of algorithmic problems (product of integers, product of matrices, sorting, etc.) are computable in linear time on *cellular automata* (CA), a local and massively parallel model. For each such problem, the literature presents an "ad hoc" parallel and local algorithmic strategy and gives the program of the final CA in an informal way [12, 8]. However, the considered problems can be defined inductively in a natural way.

For instance, the product of two integers in binary notation is simply defined by the usual school method and one may hope to directly derive a parallel program from such an inductive process.

Descriptive complexity and linear time on cellular automata

The present paper is a considerably extended version of the conference paper [18] which is in some sense the sequel of the paper [3] (see also [19]). First, [3] observes that the inductive processes defining the considered problems (product of integers, product of matrices, sorting, etc.) are "local" and are naturally formalized by *Horn formulas*, that is by conjunctions of first-order Horn clauses. Therefore, the computation is nothing else than the classical resolution method on Horn clauses, as in Prolog and Datalog [25, 14, 1]. Moreover, on every concrete problem defined by a Horn formula with $d+1$ first-order variables, this inductive computation by Horn rules can be geometrically modeled as the displacement of a d-dimensional hyperplane along some fixed line in a space of dimension $d+1$. To capture these inductive behaviors, [3] defines a logic denoted $\texttt{monot-ESO-HORN}^d(\forall^{d+1}, \texttt{arity}^{d+1})$ obtained from the logic ESO-HORN tailored by Grädel [14] to characterize P, by restricting both the number of first-order variables and the arity of second-order predicate symbols. Besides, it includes an additional restriction – the "monotonicity condition" – that reflects the geometrical consideration above-mentioned. [3] proves that this logic exactly characterizes the *linear time* complexity class of cellular automata: more precisely, for each integer $d \geq 1$, a set L of d-dimensional pictures can be decided in linear time on a d-dimensional CA – written $L \in \texttt{DLIN}^d_{\texttt{CA}}$ – if and only if it can be expressed in $\texttt{monot-ESO-HORN}^d(\forall^{d+1}, \texttt{arity}^{d+1})$. For short:

$$\texttt{DLIN}^d_{\texttt{CA}} = \texttt{monot-ESO-HORN}^d(\forall^{d+1}, \texttt{arity}^{d+1}).$$

To summarize, expressing a concrete problem in this logic – which seems an *easy* task in practice and is also a *necessary and sufficient* condition according to the above equality – *guarantees* that this problem can be solved in *linear time* on a CA; moreover, the Horn formula that defines the problem can be *automatically* translated into a program of CA that computes it in *linear time*.

Logics for minimal time of cellular automata?

At this point, two natural questions arise:

1. Besides linear time, a robust and very expressive complexity class, what are the other *significant* and *robust* complexity classes of CA?
2. Can we exhibit characterizations of those complexity classes in some naturally (syntactically) defined logics so that any definition of a problem in such a logic can be *automatically* translated into a program of the complexity considered?

Besides *linear time*, the main complexity notion well-studied for a long time in the literature of CA is *real-time*, i.e., *minimal time* [5, 33, 10]. A cellular automaton is said to run in *real-time* if it stops, i.e., gives the answer yes or no, at the *minimal* time sufficient for the output cell (the cell that gives the answer) to have received *each* letter of the input. Real-time complexity appears as a *sensitive/fragile* notion and one generally thinks it is so for CA of dimension 2 or more [35, 15]. However, maybe surprisingly, one knows that real-time complexity is a *robust* notion for *one-dimensional* CA in the following sense: according to the many *natural* variants of the definition of a one-dimensional CA, which essentially rest on the choice of the *neighborhood* of the CA and the *parallel* or *sequential* presentation of its input word, *exactly three* real-time classes of one-dimensional CA[2] have been proved to be distinct [5, 4, 20, 31, 34, 39, 40]:

1. $\texttt{RealTime}_{\texttt{CA}} = \texttt{RealTime}_{\texttt{OIA}}$;
2. $\texttt{RealTime}_{\texttt{IA}}$;
3. $\texttt{Trellis} = \texttt{RealTime}_{\texttt{OCA}}$.

[2] By default, a CA has a *two-way* communication and a *parallel input* mode. Any CA (resp. *one-way* CA or OCA) with *sequential input* mode is also called an *iterative array* or IA (resp. *one-way* IA or OIA).

The final and decisive step to establish this classification is a nice dichotomy of [31] on *admissible* neighborhoods[3] of CA, which can be rephrased as follows: for each neighborhood $\mathcal{N}$ admissible with respect to the first cell as output cell, the real-time complexity class of one-dimensional CA with parallel input mode and neighborhood $\mathcal{N}$,

- either is equal to the real-time class for the neighborhood $\{-1, 0, 1\}$, i.e., $\texttt{RealTime}_{\texttt{CA}}$ (class 1 above),
- or is equal to the real-time class for the neighborhood $\{-1, 0\}$, i.e., `Trellis` (class 3 above).

Further, it is *surprising* to notice that

- the mutual relations between those three real-time classes are *wholly elucidated*: classes `Trellis` and $\texttt{RealTime}_{\texttt{IA}}$ are mutually *incomparable for inclusion* whereas we have the strict inclusion $\texttt{Trellis} \cup \texttt{RealTime}_{\texttt{IA}} \subsetneqq \texttt{RealTime}_{\texttt{CA}}$ [5, 7, 34],
- it is *unknown* whether the trivial inclusion $\texttt{RealTime}_{\texttt{CA}} \subseteq \texttt{DLIN}^1_{\texttt{CA}}$ is strict; worse, whether the inclusion $\texttt{RealTime}_{\texttt{CA}} \subseteq \texttt{LinSpace}$ is strict is an open problem!

Logics and grid circuits for real-time classes

Each of the three real-time classes 1-3 is *robust*, i.e, is not modified for many *variants* of CA (change of neighborhoods, etc.) and has two or three *quite different* equivalent definitions. For example, $\texttt{RealTime}_{\texttt{CA}}$ is equal to the *linear time* class of one-way CA with parallel input mode [4, 39]. Similarly, [28] has proved the surprising result that `Trellis` *is* the class of languages generated by *linear conjunctive grammars* (see also [29]) and [36] has established that a language L is in $\texttt{RealTime}_{\texttt{IA}}$ if and only if its *reverse* language L^R is recognized in *real-time* by a *one-way alternating automaton with one counter*.

Logics have two nice and complementary properties: they are *flexible*, hence *expressive*; they have *normal forms*, so can be tailored for *efficient programming*. The main idea that led us to conceive the *different* logics for real-time classes can be summarized by the following simple question: what are the *different* ways to decide a language on a *square grid circuit*?

For any integer $n \geq 1$ and any input word $w = w_1 \dots w_n$ of length n, let C_n be the grid circuit $n \times n$ where the state $q \in \texttt{S}$ (for finite `S`) of any site (i, j), $1 \leq i, j \leq n$, is determined by the states of its "predecessors" $(i-1, j)$, if it exists $(i > 1)$, and $(i, j-1)$, if it exists $(j > 1)$, and by the letter w_h of the input word if this letter is placed on the site (i, j). The input word w is accepted by the grid circuit C_n if the state of the output cell (n, n) is accepting. Up to symmetries, there are three canonical ways to arrange an input word $w = w_1 \dots w_n$ on the grid C_n, see Figure 1:

1. $\texttt{GRID}_1$: place the input on any *side* (or, equivalently, on both sides[4]) that does (do) not contain the output cell;
2. $\texttt{GRID}_2$: place the input on the *diagonal that contains* the output cell;
3. $\texttt{GRID}_3$: place the input on the *diagonal opposite* to the output cell.

Remark 1. *To be complete, one should say that there is a fourth arrangement: place the input word on a* side containing *the output cell. In this case, the grid circuit behaves like a* finite automaton *(CA of dimension zero).*

Remark 2. *In order to get a natural logic corresponding to* $\texttt{GRID}_3$ *(called* inclusion logic*) it will be convenient to rotate Figure 1 by an anticlockwise rotation of* $90°$*: see Figure 11 below.*

A simple (reversible) deformation transforms a grid circuit of $\texttt{GRID}_i$, $i = 1, 2, 3$, into a time-space diagram of a CA of the real-time class i considered (recall: 1: $\texttt{RealTime}_{\texttt{CA}}$; 2: $\texttt{RealTime}_{\texttt{IA}}$; 3: `Trellis`), and conversely. More precisely, to characterize the three real-time classes, we will define three sub-logics of the Horn logic that characterizes linear time of one-dimensional CA ($\texttt{DLIN}^1_{\texttt{CA}} = \texttt{monot-ESO-HORN}^1(\forall^2, \texttt{arity}^2)$), called respectively `pred-ESO-HORN`, `pred-dio-ESO-HORN` and `incl-ESO-HORN` (defined in the next section), and we will prove the following equalities:

[3] The *neighborhood* of a CA is the finite set of integers $\mathcal{N}$ such that the state of any cell x at any non-initial instant t is determined by the states of the cells $x + d$, for $d \in \mathcal{N}$, at instant $t - 1$. A neighborhood is *admissible* with respect to a fixed output cell (in general the first or the last cell) if it allows to communicate each bit of the input to the output cell.

[4] This equivalence is the consequence of Step 7 (folding the domain) in the proof of Lemma 1 below.

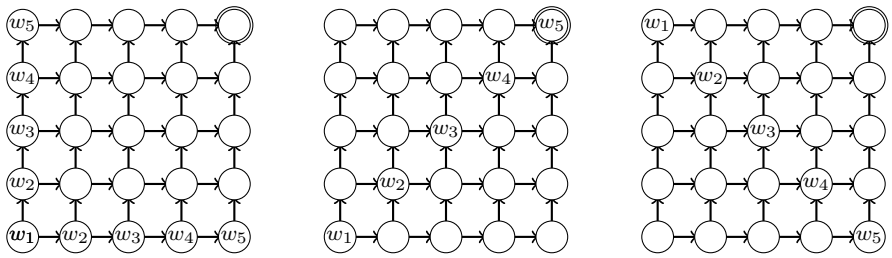


Figure 1: The three ways to arrange the input on the grid

1. pred-ESO-HORN = GRID$_1$ = RealTime$_{\text{CA}}$;
2. pred-dio-ESO-HORN = GRID$_2$ = RealTime$_{\text{IA}}$;
3. incl-ESO-HORN = GRID$_3$ = Trellis.

To establish the double nature of our three logics and deduce the previous equalities 1-3, we present each logic in two forms:

- we try to define it *as large as possible*, showing the extent of its *expressiveness*;
- we prove for it the *most restricted normal form*.

In each case, a formula in normal form can be *translated literally* into a grid program, which is essentially a CA of the considered *real-time* complexity class.

More flexible logics for programming real-time cellular automata

The best argument in favor of our logics is that they allow many properties to be expressed naturally and therefore to be easily programmed on a grid, or equivalent, on a cellular automaton in real time. However, the obligation to formalize everything with Horn clauses seems too strict for a natural expression of some reference languages of the literature, for example, the Dyck language (the set of well-parenthesed expressions, see [28], pp 81-83, or [10]), or the set of prime integers [12, 9, 27], or the Čulik language (the set of words of the form $a^i b^{i+j} a^j$) [20, 6, 39]. Fortunately, we will show in this paper that all these examples are naturally expressed in slightly extended versions of our logics. They now allow a limited use of *negation* on hypothesis computation atoms. This is comparable to the Datalog extension called *Stratified Datalog*, see e.g. [1].

An essential point should be emphasized: even if they allow more flexible expression, our extended logics, called *inductive* logics (they are no longer *Horn* logics), are equivalent to the original logics pred-ESO-HORN, pred-dio-ESO-HORN and incl-ESO-HORN, respectively, because we establish that they still characterize the same complexity classes, RealTime$_{\text{CA}}$, RealTime$_{\text{IA}}$ and Trellis, respectively. Overall, the main new contribution of this paper compared to its conference version [18] is the study of a number of reference problems of the literature that we express in our logics in a natural way. This is made possible thanks to the extensions of our Horn logics we call *inductive logics*.

Structure of the paper: In preliminaries, we recall the classical definitions of one-dimensional cellular automata and of their real-time classes; moreover, after defining our three Horn logics, we succinctly present our main results and, as examples, we give natural expressions of some classical problems in our Horn logics ; in Subsection 3.2, a simplified version of the translation process from a logical formula to some cellular automaton is detailed for one of those examples. Section 4 formally describes the first step of this translation process, i.e., the normalization of our Horn logics, in the general case. In Section 5, we define extensions of our Horn logics with negation — so-called inductive logics — and similarly describe their normalization process. Using those normal forms, we show in Section 6 that our logics exactly characterize the three real-time complexity classes of cellular automata and also – for inclusion logic – the class of linear conjunctive languages of Okhotin [28]. In Sections 7 and 8, some famous problems of the literature are programmed in our logics. Finally, Section 9 gives a conclusion with open problems and suggests some lines of research.

2. Preliminaries

2.1. Cellular automata and real-time complexity

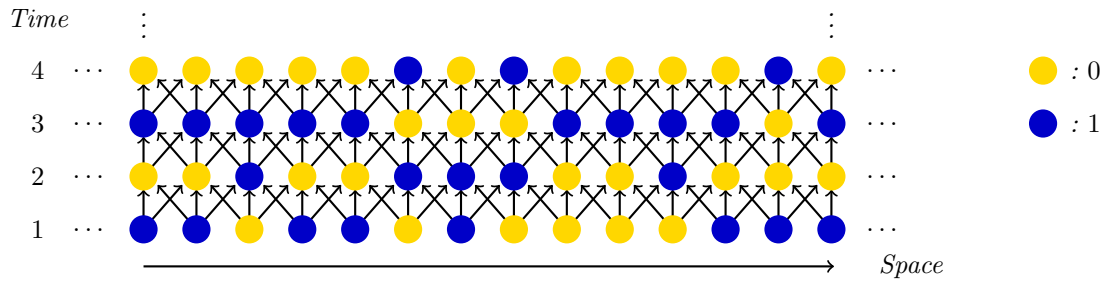
A one-dimensional cellular automaton (CA) is a linear array of identical finite state automata (called cells). Each cell takes its values from a finite *set of states* $\mathtt{S}$ and interact with its adjacent neighbors at discrete time steps. At initial time $t = 1$, each cell receives a state from $\mathtt{S}$. Then the computation is carried out locally: the new state of a cell is the result of a *transition function* $\mathtt{f}$ that depends on the states of its *neighborhood* at the previous time step. This transition function applies synchronously to all cells at each time step. Formally said:

Definition 1. *A cellular automaton (CA) is a triple* $(\mathtt{S}, \mathcal{N}, \mathtt{f})$ *where:*

- $\mathtt{S}$ *is a finite set of states*
- $\mathcal{N} = \{n_1, \ldots, n_{|\mathcal{N}|}\} \subset \mathbb{Z}$ *is the neighborhood*
- $\mathtt{f} : \mathtt{S}^{|\mathcal{N}|} \to \mathtt{S}$ *is the transition function.*

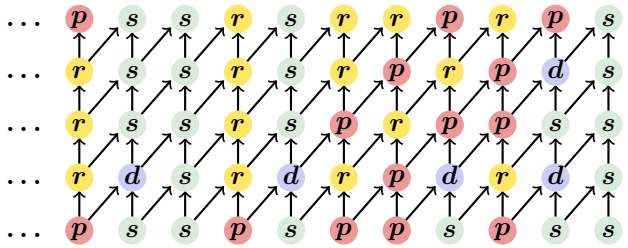
Denoting by $\langle c, t\rangle$ *the state of the cell* c *at time* t*, the state is updated in this way:* $\langle c, t+1\rangle = \mathtt{f}(\langle c + n_1, t\rangle, \ldots, \langle c + n_{|\mathcal{N}|}, t\rangle)$

Example 1. *Consider the cellular automaton* $\mathcal{A} = (\{0,1\}, \{-1,0,1\}, \mathtt{f})$ *where* $\mathtt{f} : \{0,1\}^3 \to \{0,1\}$ *is defined by* $\mathtt{f}(x,y,z) = \max(x,z) \times (1-y) + \min(1-x, 1-z) \times y$*. The first steps of the computation on the initial configuration* $\ldots 11011010000111 \ldots$ *are depicted in the following* space-time diagram.



Let us look at another cellular automaton $\mathcal{T} = (\{d,p,s,r\}, \{-1,0\}, \mathtt{f})$ *where* $\mathtt{f} : \{p,s,d,r\}^2 \to \{p,s,d,r\}$ *is presented in the table below. On the right, the space-time diagram depicts the beginnings of the computation on the input* $\ldots psspsppspss$*.*

	p	s	d	r
p	p	d	p	p
s	r	s	r	r
d	r	s	r	p
r	r	s	s	r



One main difference between these two CA is that the information flow goes in both directions for $\mathcal{A}$ since its neighborhood is $\{-1,0,1\}$, whereas the information flow is restricted from left to right for $\mathcal{T}$ since its neighborhood is $\{-1,0\}$. Below we will refer to automata with bidirectional communication (whose canonic neighborhood is $\{-1,0,1\}$) as *two-way* CA and to those with unidirectional communication (whose canonic neighborhood is $\{-1,0\}$) as *one-way* CA.

In what follows, we will consider CA as a *language recognizer* which operates on input word w and answers either "accept" or "reject". For that, it should be made clear how the input w is given to the CA and how the result of the computation is obtained.

First, we state some conditions on the set of states $\mathtt{S}$.

- The input word letters belong to the set of states: $\Sigma \subset \mathtt{S}$.

- A subset of *accepting states* $\mathtt{S}_{acc} \subset \mathtt{S}$ is identified.
- Two special states are also distinguished: one *permanent state* $\sharp$ and one *quiescent state* λ. A cell in the permanent state $\sharp$ remains in this state forever. A cell in the quiescent state λ remains in this state as long as its neighborhood is quiescent.

Regarding on how to present the input to the array, two modes are usually considered: the *parallel mode* and the *sequential mode*. In parallel mode, the whole input is supplied at initial time: the i-th symbol of the input $w = w_1 \dots w_n$ is given to the cell i at time 1: $\langle i, 1\rangle = w_i$. In sequential mode, for an input of length n, all cells of index in $[1, n]$ are initially in the quiescent state λ and the n input symbols are read one after other by a distinguished cell, the cell 1: w_t is given to the cell 1 at time t. This requires a specific transition function $\mathtt{f}_{input} : \Sigma \times \mathtt{S}^{|\mathcal{N}|} \to \mathtt{S}$ that applies to the distinguished cell 1: $\langle 1, 1\rangle = \mathtt{f}_{input}(w_1, \lambda, \dots, \lambda)$, $\langle 1, t\rangle = \mathtt{f}_{input}(w_t, \langle 1 + n_1, t - 1\rangle, \dots, \langle 1 + n_{|\mathcal{N}|}, t - 1\rangle)$, for $t > 1$. Furthermore, in both modes, the computation is bounded by the length of the input: initialized to the permanent state $\sharp$, the cells of index outside $[1, n]$ remain inactive. In the literature, a CA that reads the input word sequentially is known as *iterative array* (IA).

With outputs being "accept" or "reject", one specific cell, called *output cell*, is enough to indicate the answer of the computation. For the neighborhood $\{-1, 0, 1\}$ that permits two-way communication, the output cell is usually chosen to be the cell 1. For the neighborhood $\{-1, 0\}$ that induces one-way communication, the output cell is the cell n, the single one able to get all information of the input. Now a word w is said to be *accepted* if there is a time t such that the output cell reaches a state in $\mathtt{S}_{acc}$.

In the sequel, we will focus exclusively on minimal computation time. In minimal time, called *real-time*, the computation on an input w is completed as soon as the output cell has received each letter of the input. Then a language is said to be recognized in real-time if each of its words is accepted in minimal time. It gives us four classes of real-time CA languages, according to:

- the input mode is parallel or sequential;
- the communication is two-way with the neighborhood $\{-1, 0, 1\}$ or one-way with the neighborhood $\{-1, 0\}$.

Definition 2 (Real-time complexity classes)**.** *Figure 2 illustrates these classes.*

1. *$\mathcal{L}$ belongs to* $\mathtt{RealTime}_{\mathtt{CA}}$*, the class of languages recognized in real-time by two-way cellular automata with parallel input mode, if $\mathcal{L}$ consists of all the words for which the output cell* 1 *is in an accepting state at time n.*
2. *$\mathcal{L}$ belongs to* $\mathtt{RealTime}_{\mathtt{OIA}}$*, the class of languages recognized in real-time by one-way iterative arrays with sequential input mode, if $\mathcal{L}$ consists of all the words for which the output cell n is in an accepting state at time $2n - 1$.*
3. *$\mathcal{L}$ belongs to* $\mathtt{RealTime}_{\mathtt{IA}}$*, the class of languages recognized in real-time by two-way iterative arrays with sequential input mode, if $\mathcal{L}$ consists of all the words for which the output cell* 1 *is in an accepting state at time n.*
4. *$\mathcal{L}$ belongs to* $\mathtt{RealTime}_{\mathtt{OCA}}$*, the class of languages recognized in real-time by one-way cellular automata with parallel input mode, if $\mathcal{L}$ consists of all the words for which the output cell n is in an accepting state at time n.*

It turns out that $\mathtt{RealTime}_{\mathtt{CA}}$ and $\mathtt{RealTime}_{\mathtt{OIA}}$ characterize the same class of languages (see [4, 20]). Further, as illustrated in Figure 2, the time-space diagram of a $\mathtt{RealTime}_{\mathtt{OCA}}$ is topologically equivalent to what is called a `Trellis`. Just to recapitulate, we have exactly three distinct real-time classes of cellular automata:

1. $\mathtt{RealTime}_{\mathtt{CA}} = \mathtt{RealTime}_{\mathtt{OIA}}$;
2. $\mathtt{RealTime}_{\mathtt{IA}}$;
3. $\mathtt{Trellis} = \mathtt{RealTime}_{\mathtt{OCA}}$.

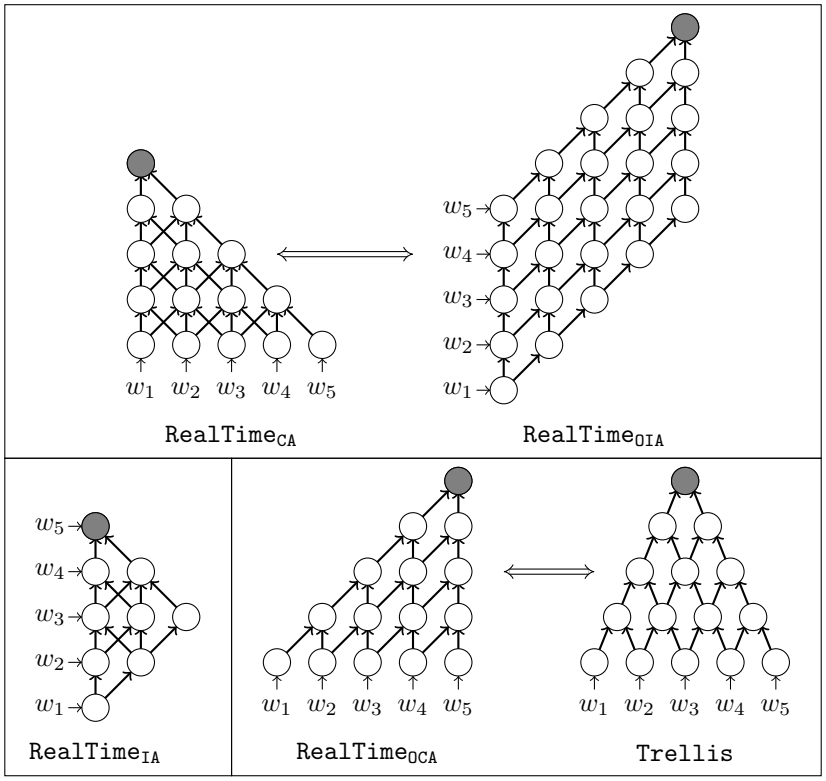


Figure 2: The space-time diagrams of the three natural real-time classes

2.2. Our logics

The "local" nature of our logics requires that the underlying structure encoding an input word $w = w_1 \dots w_n$ on its index interval $[1, n]$ only uses the *successor* and *predecessor* functions and the monadic predicates `min` and `max` as its *only* arithmetic functions/predicates:

Definition 3 (structure encoding a word)**.** *Each nonempty word $w = w_1 \dots w_n \in \Sigma^n$ on a fixed finite alphabet Σ is represented by the first-order structure*

$$\langle w \rangle := ([1, n]; (Q_s)_{s \in \Sigma}, \mathtt{min}, \mathtt{max}, \mathtt{suc}, \mathtt{pred})$$

of domain $[1, n]$, monadic predicates Q_s, $s \in \Sigma$, `min` and `max`, such that $Q_s(i) \iff w_i = s$, $\mathtt{min}(i) \iff i = 1$, and $\mathtt{max}(i) \iff i = n$, and unary functions `suc` and `pred` such that $\mathtt{suc}(i) = i + 1$ for $i < n$ and $\mathtt{suc}(n) = n$, $\mathtt{pred}(i) = i - 1$ for $i > 1$ and $\mathtt{pred}(1) = 1$. Let $\mathcal{S}_\Sigma$ denote the signature $\{(Q_s)_{s \in \Sigma}, \mathtt{min}, \mathtt{max}, \mathtt{suc}, \mathtt{pred}\}$ of structure $\langle w \rangle$. The monadic predicates Q_s, $s \in \Sigma$, `min`, and `max` of $\mathcal{S}_\Sigma$ are called input predicates*.*

Notation. *Let $x + k$ and $x - k$ abbreviate the terms $\mathtt{suc}^k(x)$ and $\mathtt{pred}^k(x)$, for a fixed integer $k \geq 0$.*

Let us now define two of our logics:

Definition 4 (predecessor logics)**.** *A* predecessor Horn formula *(resp.* predecessor Horn formula with diagonal input-output*) is a formula of the form $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x, y)$ where $\mathbf{R}$ is a set of binary predicates called* computation predicates *and ψ is a conjunction of Horn clauses on the variables x, y, of signature $\mathcal{S}_\Sigma \cup \mathbf{R}$ (resp. $\mathcal{S}_\Sigma \cup \mathbf{R} \cup \{=\}$), of the form $\delta_1 \wedge \dots \wedge \delta_r \to \delta_0$ where the conclusion δ_0 is either a* computation atom *$R(x, y)$ with $R \in \mathbf{R}$, or $\bot$ (*False*) and each hypothesis δ_i is*

1. *either an* input literal *(resp.* input conjunction*) of one of the forms:*
   - *$Q_s(x - a)$, $Q_s(y - a)$ (resp. $Q_s(x - a) \wedge x = y$), for $s \in \Sigma$ and an integer $a \geq 0$,*
   - *$U(x - a)$, $\neg U(x - a)$, $U(y - a)$ or $\neg U(y - a)$, for $U \in \{\mathtt{min}, \mathtt{max}\}$ and an integer $a \geq 0$,*
2. *or a* computation atom *of the form $S(x - a, y - b)$ or $S(y - b, x - a)$, for $S \in \mathbf{R}$ and some integers $a, b \geq 0$.*

Let `pred-ESO-HORN` (resp. `pred-dio-ESO-HORN`) denote the class of *predecessor Horn formulas* (resp. *predecessor Horn formulas with diagonal input-output*) and, by abuse of notation, the class of languages they define.

The formulas of the "predecessor" logics defined above use the *predecessor* function but *not* the *successor* function: both logics inductively define problems in *increasing* both coordinates x and y. The inductive principle of our last logic is seemingly different: it lies on *inclusions* of intervals $[x, y]$.

Definition 5 (inclusion logic)**.** *An* inclusion Horn formula *is a formula of the form* $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x, y)$ *where* $\mathbf{R}$ *is a set of binary predicates called* computation predicates *and* ψ *is a conjunction of Horn clauses of signature* $\mathcal{S}_\Sigma \cup \mathbf{R} \cup \{=, \leq, <\}$*, of the form* $x \leq y \wedge \delta_1 \wedge \ldots \wedge \delta_r \rightarrow \delta_0$ *where the conclusion* δ_0 *is either a* computation atom $R(x, y)$ *with* $R \in \mathbf{R}$*, or the atom* $\perp$ *(*False*), and each hypothesis* δ_i *is*

1. *either an* input literal *of the form*[5] $U(x + a)$, $\neg U(x + a)$, $U(y + a)$ *or* $\neg U(y + a)$*, for* $U \in \{(Q_s)_{s\in\Sigma}, \mathtt{min}, \mathtt{max}\}$ *and an integer* $a \in \mathbb{Z}$*,*
2. *or an (in)equality* $x = y$ *or* $x < y$ [6]*,*
3. *or a conjunction of the form*

$$S(x + a, y - b) \wedge x + a \leq y - b$$

for $S \in \mathbf{R}$ *and some integers* $a, b \geq 0$*.*

Let `incl-ESO-HORN` denote the class of *inclusion Horn formulas* and, also, the class of languages they define.

Note that the "inclusion" meaning of logic `incl-ESO-HORN` is given by the hypotheses $x \leq y$ and $x + a \leq y - b$. It means that the inductive computation of each value $R(x, y)$, for $x \leq y$ and $R \in \mathbf{R}$, only use values of the form $S(x + a, y - b)$, for $S \in \mathbf{R}$ and an *included* interval $[x + a, y - b] \subseteq [x, y]$.

Notation. *We will freely use the intuitive abbreviations* $x > a$*,* $x = a$*, for a constant integer* $a \geq 1$*, and* $x \leq n - a$*,* $x < n - a$*,* $x = n - a$*, for a constant integer* $a \geq 0$*, and similarly for* y*. For example,* $x > 3$ *is written in place of* $\neg\mathtt{min}(x - 2)$ *and* $y \leq n - 2$ *is written in place of* $\neg\mathtt{max}(y + 1)$*.*

Remark 3. *Without loss of generality, we can suppose that each clause having a hypothesis atom of the form* $S(x - a, y - b)$ *or* $S(y - b, x - a)$*, for* $a, b \geq 0$*, has* also *the hypotheses* $x > a$ *(if* $a > 0$*) and* $y > b$ *(if* $b > 0$*). The same for each hypothesis atom of the form* $Q_s(x - a)$ *or* $Q_s(y - b)$*, for* $a, b > 0$*. Similarly, we assume that each clause with a hypothesis of the form* $Q_s(x + a)$ *(resp.* $Q_s(y + a)$*), with* $a > 0$*,* also *contains the hypothesis* $x \leq n - a$ *(resp.* $y \leq n - a$*). Similarly, for each atom* $S(x + a, y - b)$*, for* $a, b \geq 0$*.*

Remark 4. *The presentation of the input is* more restrictive *in Definition 4 of predecessor logics than in that of inclusion logic (Definition 5) because we have forbidden the use of the successor function for uniformity/aesthetics. However, allowing the* largest *set of input literals* $(\neg)U(x + a)$*,* $(\neg)U(y + a)$*, for* $U \in \{(Q_s)_{s\in\Sigma}, \mathtt{min}, \mathtt{max}\}$ *and* $a \in \mathbb{Z}$*,* does not modify *the expressive power of predecessor logics: Steps 5 and 6 of the normalization of inclusion logic in Section 4 can be easily adapted to predecessor logics.*

Remark 5. *As usual, it is convenient to adopt the semantics of the* minimal model *of Horn formulas. That means that if we have* $\langle w \rangle \models \Phi$ *for a formula* $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x, y)$ *in any of our logics, then there is a model* $(\langle w \rangle, \mathbf{R}) \models \forall x \forall y \psi(x, y)$ *for which each* $R \in \mathbf{R}$ *is minimal. We will say that the formula* $\forall x \forall y \psi(x, y)$ defines *the (minimal) predicates* $R \in \mathbf{R}$ *on* $\langle w \rangle$*.*

[5] Without loss of generality, we assume that there is no negation on a predicate Q_s.

[6] Then, the hypothesis $x \leq y$ is redundant.

2.3. Our main results

Much of the paper will be devoted to show that our three logics characterize the three real-time complexity classes of CA. Precisely, we will prove the following statements.

The languages accepted in real-time by two-way CA's with input fed in a parallel way and output read on the first cell are exactly the languages defined by the predecessor logic:

Theorem 1. $\texttt{RealTime}_{\texttt{CA}} = \texttt{pred-ESO-HORN}$.

The languages accepted in real-time by IA are exactly the languages defined by the predecessor logic with diagonal input-output:

Theorem 2. $\texttt{RealTime}_{\texttt{IA}} = \texttt{pred-dio-ESO-HORN}$.

The languages accepted by Trellis are exactly the languages defined by the inclusion logic:

Theorem 3. $\texttt{Trellis} = \texttt{incl-ESO-HORN}$.

In order to facilitate the programming (= expression) of the problems, we will also extend our Horn logics by a limited use of negation on hypothesis computation atoms (like in *Stratified Datalog*) while preserving their computational complexity, i.e., extending Theorems 1, 2 and 3 to those extended logics.

3. First examples: express problems; translate formulas into automata

In this section, we give an illustration of our logics and their transformation techniques about three classical problems.

3.1. Three examples of problems expressed in our three Horn logics

Our logics make it possible to express problems in a natural way.

Example 2. *The language* $\texttt{Palindrome} = \{w \in \Sigma^+ \mid w = w^R\}$.

The language Palindrome is defined by the formula $\exists \texttt{notPal} \forall x \forall y \psi$ of incl-ESO-HORN where ψ is the conjunction of the following clauses in which the inductively defined predicate $\texttt{notPal}(x, y)$ means that the factor $w_x \dots w_y$ is not a palindrome:

- $x < y \wedge Q_s(x) \wedge Q_t(y) \rightarrow \texttt{notPal}(x, y)$, for $s, t \in \Sigma$, $s \neq t$;
- $x < y \wedge \texttt{notPal}(x + 1, y - 1) \rightarrow \texttt{notPal}(x, y)$;
- $x \leq y \wedge \texttt{min}(x) \wedge \texttt{max}(y) \wedge \texttt{notPal}(x, y) \rightarrow \bot$.

As a consequence of Theorem 3, Palindrome belongs to Trellis (see [10], ex. 4, pp 267-268).

Example 3. *The language* Unbordered *is the set of words* $w \in \Sigma^+$ *with no proper prefix of length at least 2 equal to a suffix. Equivalently,*

$$\texttt{Unbordered} \coloneqq \Sigma^+ \setminus \{uvu \mid u, v \in \Sigma^* \wedge |u| \geq 2\}.$$

The language Unbordered, which is the complement of the language L studied in [34], can be defined by the formula $\Phi_{\texttt{Unbordered}} \coloneqq \exists \texttt{Border} \forall x \forall y \psi$ of pred-ESO-HORN, where ψ is the conjunction of the following clauses involving the binary predicate Border whose intuitive meaning is given by the equivalence $\texttt{Border}(x, y) \iff 2 \leq x < y \wedge w_1 \dots w_x = w_{y-x+1} \dots w_y$, as expressed by clauses 1 and 2:

1. $\neg\texttt{min}(x) \wedge \texttt{min}(x - 1) \wedge \neg\texttt{min}(y - 1) \wedge Q_s(x - 1) \wedge Q_s(y - 1) \wedge Q_t(x) \wedge Q_t(y) \rightarrow \texttt{Border}(x, y)$, for all $s, t \in \Sigma$ (meaning: $2 = x < y \wedge w_1 w_2 = w_{y-1} w_y \rightarrow \texttt{Border}(2, y)$);
2. $\neg\texttt{min}(x) \wedge \neg\texttt{min}(y) \wedge \texttt{Border}(x - 1, y - 1) \wedge Q_s(x) \wedge Q_s(y) \rightarrow \texttt{Border}(x, y)$, for $s \in \Sigma$ (meaning: $2 \leq x - 1 < y - 1 \wedge w_1 \dots w_{x-1} = w_{y-x+1} \dots w_{y-1} \wedge w_x = w_y \rightarrow (3 \leq x < y \wedge w_1 \dots w_x = w_{y-x+1} \dots w_y)$);

3. $\texttt{max}(y) \wedge \texttt{Border}(x,y) \to \bot$ (meaning: $\forall x\, \neg\texttt{Border}(x,n)$).

Justification: We have the equivalence:
$\exists x\ \texttt{Border}(x,n) \iff \exists x(2 \leq x < n \wedge w_1 \dots w_x = w_{n-x+1} \dots w_n) \iff w \notin \texttt{Unbordered}$.

A geometric view of the "algorithm" is given in Figure 3 where the usual x and y axes are implicit.

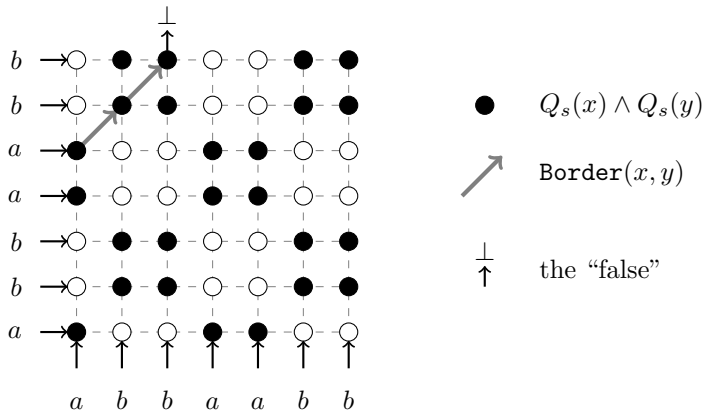


Figure 3: Computation of $\Phi_{\texttt{Unbordered}}$ on the word **abb***a***abb**

So, as a consequence of Theorem 1, Unbordered belongs to $\texttt{RealTime}_{\texttt{CA}}$. In fact, we know much more: [34] has proved $\texttt{Unbordered} \in \texttt{RealTime}_{\texttt{CA}} \setminus (\texttt{Trellis} \cup \texttt{RealTime}_{\texttt{IA}})$.

The next example uses geometric constructions defined inductively in logic.

Example 4. *The language* Disj *is the set of words* $w = w_1 \dots w_n \in \{0,1\}^+$ *of even length* $n = 2k$, $w = x_1 \dots x_k y_1 \dots y_k$, *such that* $x_i y_i \neq 11$, *for all* $i \in [1,k]$.

The language Disj (well known in Communication Complexity [23] and studied in [40]) can be defined by the formula $\Phi_{\texttt{Disj}} \coloneqq \exists\{D, I^x, I^y, H, T\}\forall x \forall y \psi$ of pred-dio-ESO-HORN where D, I^x, I^y, H, T are binary predicates such that

- $D(x,y)$ (intuitively) means $y = 2x$,
- $I^x(x,y)$ means $Q_1(x) \wedge y \geq x$,
- $I^y(x,y)$ means $Q_1(y) \wedge x \geq y$,
- $H(x,y)$ means y is even, $y \leq 2x$ and $Q_1(y/2)$,
- $T(x,y)$ means x is even, $x/2 < y \leq x$ and $Q_1(y - x/2)$,

and ψ is the following conjunction of clauses:

- for (inductively) defining D: $x = 1 \wedge y = 2 \to D(x,y)$, and $x > 1 \wedge y > 2 \wedge D(x-1, y-2) \to D(x,y)$;
- for defining I^x: $x = y \wedge Q_1(x) \to I^x(x,y)$, and $y > 1 \wedge I^x(x, y-1) \to I^x(x,y)$, and similarly for I^y;
- for defining H: 1) $y = 2x \wedge I^x(x,y) \to H(x,y)$, and 2) $x > 1 \wedge H(x-1,y) \to H(x,y)$;
- for defining T: 3) $x = y \wedge H(x,y) \to T(x,y)$, and 4) $x > 2 \wedge y > 1 \wedge T(x-2, y-1) \to T(x,y)$.

Justification: The definitions of D, I^x and I^y are obvious. Let us explain why the definitions of H and T are correct. The hypotheses of clause 1 mean $D(x,y) \wedge Q_1(x) \wedge y \geq x$ which implies y even, $y \leq 2x$, and $Q_1(y/2)$; the hypotheses of clause 3 mean $x = y$, y even, $y \leq 2x$, and $Q_1(y/2)$, which imply together x even, $x/2 < y \leq x$, $y - x/2 = y/2$, and then $Q_1(y - x/2)$; clause 4 is justified by the trivial implication $(x-2)/2 < y - 1 \leq x - 2 \to x/2 < y \leq x$ and the identity $(y-1) - (x-2)/2 = y - x/2$.

Here are the last two clauses of ψ. First, in order to reject all words of odd length n, we use the clause $y = n-1 \wedge D(x,y) \to \bot$. Second, for words of even length n, the comparison between w_y and $w_{y-n/2}$ with $y > n/2$ is handled by the clause $x = n \wedge I^y(x,y) \wedge T(x,y) \to \bot$, which expresses the falsity of the conjunction $I^y(n,y) \wedge T(n,y)$, equivalent to $w_y = 1 \wedge w_{y-n/2} = 1$ by the previous clauses; hence, this clause means $w \notin \texttt{Disj}$. See Figure 4 where the contradiction is expressed by the "signal line" in bold $I^x \to H \to T \to I^y$, which connects two points of the diagonal of ordinates $y - n/2$ and y with $w_{y-n/2} = w_y = 1$.

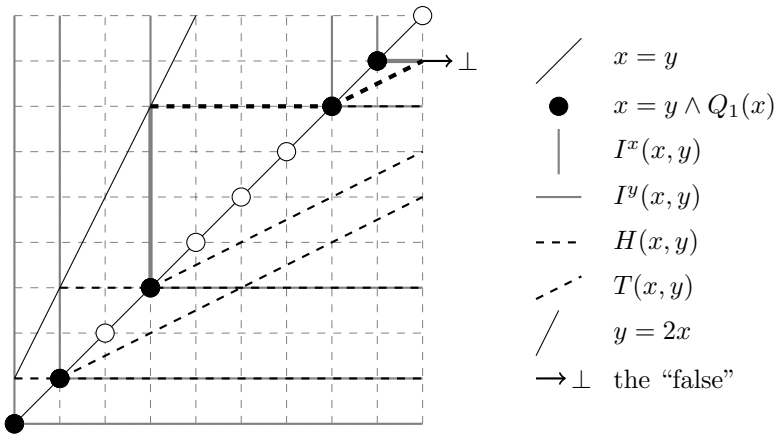


Figure 4: Computation of $\Phi_{\texttt{Disj}}$ on the word 110**1**0001**1**0 $\notin$ `Disj`

So, Theorem 2 implies $\texttt{Disj} \in \texttt{RealTime}_{\texttt{IA}}$. The language `Disj` is also known not to belong to `Trellis` as it is proved in [40].

3.2. Normalizing and translating a formula into a cellular automaton: an introduction example

In this subsection, we give an illustration, on an example, of the translation processes which will be presented rigorously and in their general form in the following sections. We take as example the formula $\Phi_{\texttt{Unbordered}} \in$ `pred-ESO-HORN` constructed in Subsection 3.1 (to describe the language `Unbordered`). Our first goal is to turn the formula into a normalized formula mimicking a grid circuit. Intuitively, a normalized formula meets the following contraints:

- it should only have one clause with conclusion $\bot$ and this clause should have the hypothesis $(x,y) = (n,n)$ to mimic the output of the grid circuit at vertex (n,n);
- in order to mimic an input on the grid circuit, all clauses using information about the input word should have the hypothesis $x = 1$;
- the only computation atoms in the hypotheses of computation clauses should be of the form $R(x-1,y)$ or $R(x,y-1)$.

Our second goal will be to deduce from this normalized formula a cellular automaton recognizing the language `Unbordered`.

Normalizing the formula defining `Unbordered`*:*

Let us detail the *normalization* process, i.e., the successive steps turning $\Phi_{\texttt{Unbordered}}$ into a formula in normal form (= mimicking a grid circuit). First, let us define what this means.

Definition 6 (normal form)**.** *A formula* $\Phi := \exists \mathbf{R} \forall x \forall y \psi$ *of* `pred-ESO-HORN` *is in* normal form *if each clause of* ψ *is of one of the following forms:*

- *an* input clause *of the form, for* $s \in \Sigma$ *and* $R \in \mathbf{R}$*:*
  $\texttt{min}(x) \wedge \texttt{min}(y) \wedge Q_s(y) \to R(x,y)$*, or* $\texttt{min}(x) \wedge \neg\texttt{min}(y) \wedge Q_s(y) \to R(x,y)$*;*
- *the* contradiction clause*, for a fixed* $R_\bot \in \mathbf{R}$*:* $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\bot(x,y) \to \bot$*;*

- *a* computation clause *of the form* $\delta_1 \wedge \ldots \wedge \delta_r \rightarrow R(x,y)$*, for* $R \in \mathbf{R}$*, where each hypothesis* δ_i *is a conjunction of the form* $S(x-1,y) \wedge \neg\mathtt{min}(x)$ *or* $S(x,y-1) \wedge \neg\mathtt{min}(y)$*, for* $S \in \mathbf{R}$*.*

Remark 6. *It will also be convenient to allow input clauses of the form* $\mathtt{min}(x) \wedge \mathtt{min}(y) \rightarrow R(x,y)$ *(resp.* $\mathtt{min}(x) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$*). Indeed, such a clause is obviously equivalent to the conjunction of input clauses* $\bigwedge_{s\in\Sigma}(\mathtt{min}(x) \wedge \mathtt{min}(y) \wedge Q_s(y) \rightarrow R(x,y))$ *(resp.* $\bigwedge_{s\in\Sigma}(\mathtt{min}(x) \wedge \neg\mathtt{min}(y) \wedge Q_s(y) \rightarrow R(x,y))$*).*

The main difficulty encountered in the normalization process will be to maintain or restore the constraint that no computation atom of the form $R(x,y)$ occurs as a hypothesis of a computation clause. This will be performed in an "ad hoc" way on our example. However, in Step 10 of Subsection 4.1, we will describe and justify a general procedure to eliminate the computation atoms $R(x,y)$ in the hypotheses of Horn clauses.

0: Before normalization. Recall that $\Phi_{\mathtt{Unbordered}}$ is the formula $\exists\mathtt{Border}\forall x\forall y\psi$ where ψ is the conjunction of the following clauses:

0.a: $\neg\mathtt{min}(x) \wedge \mathtt{min}(x-1) \wedge \neg\mathtt{min}(y-1) \wedge Q_s(x-1) \wedge Q_s(y-1) \wedge Q_t(x) \wedge Q_t(y) \rightarrow \mathtt{Border}(x,y)$, for all $s,t \in \Sigma$;

0.b: $\neg\mathtt{min}(x) \wedge \neg\mathtt{min}(y) \wedge \mathtt{Border}(x-1,y-1) \wedge Q_s(x) \wedge Q_s(y) \rightarrow \mathtt{Border}(x,y)$, for $s \in \Sigma$;

0.c: $\mathtt{max}(y) \wedge \mathtt{Border}(x,y) \rightarrow \bot$.

Remark 7. *Since* $\mathtt{Border}(x,y)$ *implies* $2 \leq x < y \leq n$*, hence* $x < n$*, clause (0.c) can be replaced by the equivalent clause*

0.c': $\mathtt{max}(y) \wedge \mathtt{Border}(x-1,y) \rightarrow \bot$.

1: Processing the contradiction clause. In order to push the contradiction to the vertex (n,n), we introduce the binary predicate $R_\bot^{\mathtt{max}(y)}$ of intuitive meaning: $R_\bot^{\mathtt{max}(y)}(x,y) \iff (\mathtt{max}(y) \rightarrow \bot)$. For this purpose, we replace clause (0.c') by the following three clauses:

1.a: $\neg\mathtt{min}(x) \wedge \mathtt{Border}(x-1,y) \rightarrow R_\bot^{\mathtt{max}(y)}(x,y)$ (of intuitive meaning: $\neg\mathtt{min}(x) \wedge \mathtt{Border}(x-1,y) \rightarrow (\mathtt{max}(y) \rightarrow \bot)$, equivalent to clause (0.c'));

1.b: $\neg\mathtt{min}(x) \wedge R_\bot^{\mathtt{max}(y)}(x-1,y) \rightarrow R_\bot^{\mathtt{max}(y)}(x,y)$ (clause transporting $R_\bot^{\mathtt{max}(y)}$ to the side $x = n$);

1.c: $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge R_\bot^{\mathtt{max}(y)}(x,y) \rightarrow \bot$ (clause giving the meaning of $R_\bot^{\mathtt{max}(y)}$).

2: Processing the input. For each $s \in \Sigma$, we introduce two new binary predicates W_s^x and W_s^y (with intuitive meaning $W_s^x(x,y) \iff Q_s(x)$ and $W_s^y(x,y) \iff Q_s(y)$) to replace the unary predicates Q_s when x or y is greater than 1. W_s^x and W_s^y are defined by the following clauses:

2.a: $\mathtt{min}(y) \wedge Q_s(x) \rightarrow W_s^x(x,y)$;

2.b: $\neg\mathtt{min}(y) \wedge W_s^x(x,y-1) \rightarrow W_s^x(x,y)$;

2.c: $\mathtt{min}(x) \wedge Q_s(y) \rightarrow W_s^y(x,y)$;

2.d: $\neg\mathtt{min}(x) \wedge W_s^y(x-1,y) \rightarrow W_s^y(x,y)$.

This justifies replacing, for any $s \in \Sigma$, atoms $Q_s(x-1)$ and $Q_s(y-1)$ by $W_s^x(x-1,y)$ and $W_s^y(x,y-1)$, respectively, and replacing atoms $Q_s(x)$ and $Q_s(y)$ by $W_s^x(x,y-1)$ and $W_s^y(x-1,y)$, respectively. So, clauses (0.a) and (0.b) become respectively:

2.e: $\neg\mathtt{min}(x) \wedge \mathtt{min}(x-1) \wedge \neg\mathtt{min}(y-1) \wedge W_s^x(x-1,y) \wedge W_s^y(x,y-1) \wedge W_t^x(x,y-1) \wedge W_t^y(x-1,y) \rightarrow \mathtt{Border}(x,y)$, for $s,t \in \Sigma$;

2.f: $\neg\mathtt{min}(x) \wedge \neg\mathtt{min}(y) \wedge \mathtt{Border}(x-1,y-1) \wedge W_s^x(x,y-1) \wedge W_s^y(x-1,y) \rightarrow \mathtt{Border}(x,y)$, for $s \in \Sigma$.

Remark 8. *Our substitutions respect the constraint that hypotheses of the form* $R(x,y)$ *are forbidden in the computation clauses of a normalized formula (Definition 6).*

3: Restriction of computation atoms to the forms $R(x-1,y)$ and $R(x,y-1)$. A new binary predicate $\texttt{Border}^{\texttt{x}-1}$ is introduced with the intuitive meaning $\texttt{Border}^{\texttt{x}-1}(x,y) \iff (x>1 \wedge \texttt{Border}(x-1,y))$. This predicate is defined by the following clause:

3.a: $\neg\texttt{min}(x) \wedge \texttt{Border}(x-1,y) \rightarrow \texttt{Border}^{\texttt{x}-1}(x,y)$.

In this way, the clause (2.f) $\neg\texttt{min}(x) \wedge \neg\texttt{min}(y) \wedge \texttt{Border}(x-1,y-1) \wedge W_s^x(x,y-1) \wedge W_s^y(x-1,y) \rightarrow \texttt{Border}(x,y)$, for $s \in \Sigma$, is replaced by the following clause:

3.b: $\neg\texttt{min}(x) \wedge \neg\texttt{min}(y) \wedge \texttt{Border}^{\texttt{x}-1}(x,y-1) \wedge W_s^x(x,y-1) \wedge W_s^y(x-1,y) \rightarrow \texttt{Border}(x,y)$, for $s \in \Sigma$.

4: Processing of min *and* max. In order to get rid of the atoms $\texttt{min}(x-1)$ and $\neg\texttt{min}(y-1)$ in clause (2.e) $\neg\texttt{min}(x) \wedge \texttt{min}(x-1) \wedge \neg\texttt{min}(y-1) \wedge W_s^x(x-1,y) \wedge W_s^y(x,y-1) \wedge W_t^x(x,y-1) \wedge W_t^y(x-1,y) \rightarrow \texttt{Border}(x,y)$, for $s,t \in \Sigma$, we introduce two binary predicates $R_{\texttt{min}(x)}$ and $R_{\neg\texttt{min}(y)}$ (with intuitive meaning $R_{\texttt{min}(x)}(x,y) \iff \texttt{min}(x)$ and $R_{\neg\texttt{min}(y)}(x,y) \iff \neg\texttt{min}(y)$) defined by the following clauses:

4.a: $\texttt{min}(x) \wedge \texttt{min}(y) \rightarrow R_{\texttt{min}(x)}(x,y)$;

4.b: $\neg\texttt{min}(y) \wedge R_{\texttt{min}(x)}(x,y-1) \rightarrow R_{\texttt{min}(x)}(x,y)$;

4.c: $\texttt{min}(x) \wedge \neg\texttt{min}(y) \rightarrow R_{\neg\texttt{min}(y)}(x,y)$;

4.d: $\neg\texttt{min}(x) \wedge R_{\neg\texttt{min}(y)}(x-1,y) \rightarrow R_{\neg\texttt{min}(y)}(x,y)$.

After substitution, clause (2.e) becomes, for $s,t \in \Sigma$:

4.e: $\neg\texttt{min}(x) \wedge R_{\texttt{min}(x)}(x-1,y) \wedge \neg\texttt{min}(y) \wedge R_{\neg\texttt{min}(y)}(x,y-1) \wedge W_s^x(x-1,y) \wedge W_s^y(x,y-1) \wedge W_t^x(x,y-1) \wedge W_t^y(x-1,y) \rightarrow \texttt{Border}(x,y)$.

Remark 9. *We have added the hypothesis atom $\neg\texttt{min}(y)$ in clause (4.e) in order that this clause fulfills the conditions of a* computation clause*, as given in Definition 6.*

5: Defining equality and inequalities. In the next part of the normalization process, we will fold the square domain $[1,n]^2$ along the diagonal $x=y$ on the over-diagonal triangle where $x \leq y$. Therefore, we need predicates representing the equality $x=y$ ($R_=$) and the inequalities $x<y$ ($R_<$) and $x \leq y$ ($R_\leq$). $R_=$ is defined jointly with the predicate $R_{\texttt{pred}}$, of intuitive meaning $R_{\texttt{pred}}(x,y) \iff x = y-1$, by the following clauses:

5.a: $\texttt{min}(x) \wedge \texttt{min}(y) \rightarrow R_=(x,y)$;

5.b: $\neg\texttt{min}(y) \wedge R_=(x,y-1) \rightarrow R_{\texttt{pred}}(x,y)$;

5.c: $\neg\texttt{min}(x) \wedge R_{\texttt{pred}}(x-1,y) \rightarrow R_=(x,y)$.

The predicates $R_<$ and $R_\leq$ are defined by the next clauses:

5.d: $\neg\texttt{min}(y) \wedge R_=(x,y-1) \rightarrow R_<(x,y)$;

5.e: $\neg\texttt{min}(y) \wedge R_<(x,y-1) \rightarrow R_<(x,y)$;

5.f: $\texttt{min}(x) \wedge \texttt{min}(y) \rightarrow R_\leq(x,y)$;

5.g: $\texttt{min}(x) \wedge \neg\texttt{min}(y) \rightarrow R_\leq(x,y)$;

5.h: $\neg\texttt{min}(x) \wedge R_<(x-1,y) \rightarrow R_\leq(x,y)$.

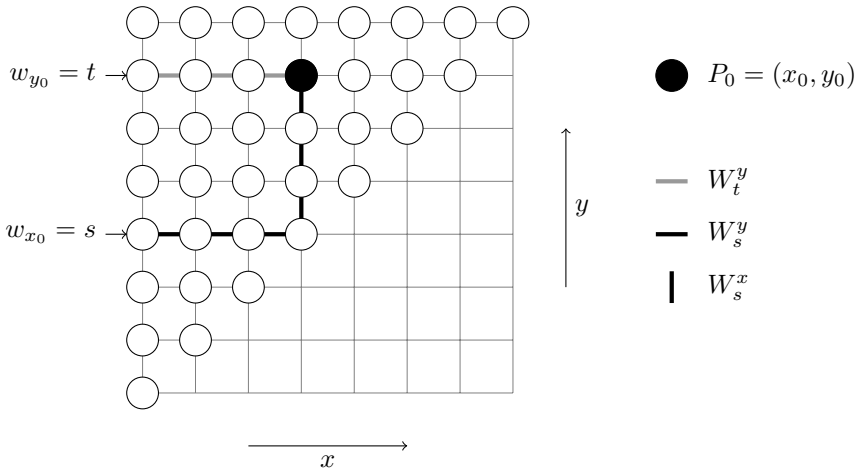


Figure 5: Behaviour of W_s^x and W_s^y after the folding

6: Folding the domain. We want to restrict the access to the input to the axis $x = 1$ with the only input clauses (i.e., the only clauses involving the input predicates Q_s) $\mathtt{min}(x) \wedge Q_s(y) \to W_s^y(x, y)$. In other words, we want to get rid of the clauses $\mathtt{min}(y) \wedge Q_s(x) \to W_s^x(x, y)$. For this purpose, we fold the square domain $[1, n]^2$ along the diagonal $x = y$ on the over-diagonal triangle $T_n = \{(x, y) \in [1, n]^2 \mid 1 \leq x \leq y \leq n\}$ by the transformation that maps any point (y, x) such that $x \leq y$ to the point (x, y). Observe that the computation predicates $\mathtt{Border}$ and $\mathtt{Border}^{\mathtt{x-1}}$ are included in T_n (see Figure 5) so that the only computation predicates really acting on (x, y) sites such that $x > y$ are the (in)equality predicates $R_=, R_{\mathtt{pred}}, R_<, R_\leq$, and the "transport" predicates W_s^x and W_s^y of the input, which are the "folding" of each other. The "folded" versions on T_n of the clauses defining W_s^x and W_s^y are, for all $s \in \Sigma$, the following clauses defining W_s^y:

- $\mathtt{min}(x) \wedge Q_s(y) \to W_s^y(x, y)$, that will be replaced by the equivalent conjunction of the next clauses (6.a) and (6.b), which are of the allowed forms of input clauses of Definition 6:

  6.a: $\mathtt{min}(x) \wedge \mathtt{min}(y) \wedge Q_s(y) \to W_s^y(x, y)$,

  6.b: $\mathtt{min}(x) \wedge \neg\mathtt{min}(y) \wedge Q_s(y) \to W_s^y(x, y)$, and

- 6.c:$\neg\mathtt{min}(x) \wedge W_s^y(x - 1, y) \wedge R_<(x - 1, y) \to W_s^y(x, y)$ (where $R_<(x - 1, y)$ means $x \leq y$),

and the following clauses defining W_s^x:

- $x = y \wedge W_s^y(x, y) \to W_s^x(x, y)$ ("rebound" of the horizontal signal W_s^y on the diagonal as the vertical signal W_s^x), which can be equivalently rewritten as the conjunction of the following two clauses:
  6.d: $\mathtt{min}(x) \wedge \mathtt{min}(y) \wedge Q_s(y) \to W_s^x(x, y)$,
  6.e: $\neg\mathtt{min}(x) \wedge R_{\mathtt{pred}}(x - 1, y) \wedge W_s^y(x - 1, y) \to W_s^x(x, y)$ (where $R_{\mathtt{pred}}(x - 1, y)$ means $x = y$), and

- $x < y \wedge \neg\mathtt{min}(y) \wedge W_s^x(x, y - 1) \to W_s^x(x, y)$, which can be rewritten as
  6.f: $\neg\mathtt{min}(y) \wedge R_\leq(x, y - 1) \wedge W_s^x(x, y - 1) \to W_s^x(x, y)$.

Figure 5 depicts the new behaviour of W_s^x and W_s^y. Note that after the folding, the computation predicates W_s^x and W_s^y are included in the upper-diagonal triangle T_n.

A normalized formula describing the language $\mathtt{Unbordered}$. We recapitulate the final list of clauses obtained after steps 1-6:

• the clauses of steps 4 and 5 defining the "arithmetic" computation predicates $R_{\mathtt{min}(x)}$, $R_{\neg\mathtt{min}(y)}$, $R_=$, $R_{\mathtt{pred}}$, $R_<$ and $R_\leq$;

• the clauses defining the "input" computation predicates W_s^x and W_s^y, for $s \in \Sigma$:

6.a: $\mathtt{min}(x) \wedge \mathtt{min}(y) \wedge Q_s(y) \to W_s^y(x, y)$;

6.b: $\texttt{min}(x) \wedge \neg\texttt{min}(y) \wedge Q_s(y) \to W_s^y(x,y)$;

6.c: $\neg\texttt{min}(x) \wedge W_s^y(x-1,y) \wedge R_<(x-1,y) \to W_s^y(x,y)$;

6.d: $\texttt{min}(x) \wedge \texttt{min}(y) \wedge Q_s(y) \to W_s^x(x,y)$;

6.e: $\neg\texttt{min}(x) \wedge R_{\texttt{pred}}(x-1,y) \wedge W_s^y(x-1,y) \to W_s^x(x,y)$;

6.f: $\neg\texttt{min}(y) \wedge R_\leq(x,y-1) \wedge W_s^x(x,y-1) \to W_s^x(x,y)$;

• the clauses defining the "main" computation predicates $\texttt{Border}$, $\texttt{Border}^{\texttt{x}-1}$ and $R_\perp^{\texttt{max}(y)}$, i.e, describing the main computation and its output:

4.e: $\neg\texttt{min}(x) \wedge R_{\texttt{min}(x)}(x-1,y) \wedge \neg\texttt{min}(y) \wedge R_{\neg\texttt{min}(y)}(x,y-1) \wedge W_s^x(x-1,y) \wedge W_s^y(x,y-1) \wedge W_t^x(x,y-1) \wedge W_t^y(x-1,y) \to \texttt{Border}(x,y)$ for $s,t \in \Sigma$;

3.a: $\neg\texttt{min}(x) \wedge \texttt{Border}(x-1,y) \to \texttt{Border}^{\texttt{x}-1}(x,y)$;

3.b: $\neg\texttt{min}(y) \wedge \neg\texttt{min}(x) \wedge \texttt{Border}^{\texttt{x}-1}(x,y-1) \wedge W_s^x(x,y-1) \wedge W_s^y(x-1,y) \to \texttt{Border}(x,y)$, for $s \in \Sigma$;

1.a: $\neg\texttt{min}(\texttt{x}) \wedge \texttt{Border}(x-1,y) \to R_\perp^{\texttt{max}(y)}(x,y)$;

1.b: $\neg\texttt{min}(\texttt{x}) \wedge R_\perp^{\texttt{max}(y)}(x-1,y) \to R_\perp^{\texttt{max}(y)}(x,y)$;

1.c: $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\perp^{\texttt{max}(y)}(x,y) \to \perp$.

We observe that each of those clauses fulfills the conditions of Definition 6:

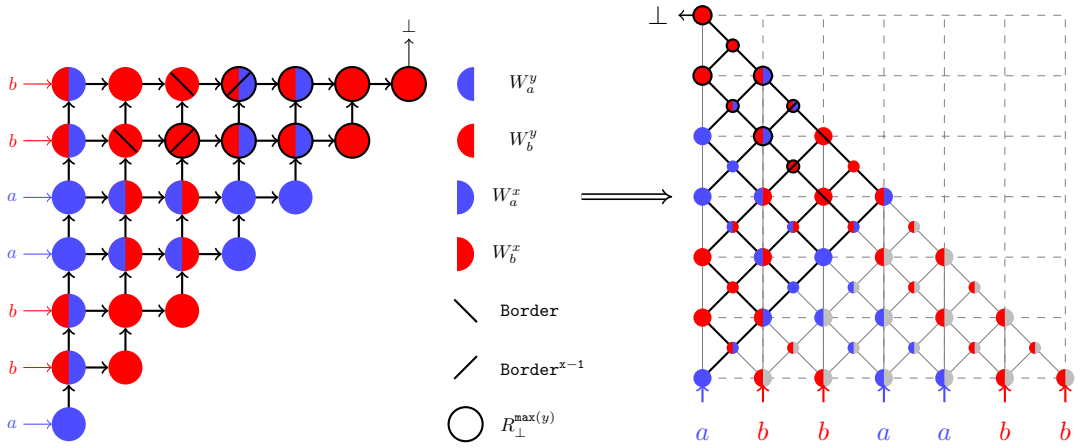
- clauses (4.a), (4.c), (5.a), (5.f), (5,g), (6,a), (6,b) and (6,d) are input clauses;
- the clause (1.c) $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\perp(x,y) \to \perp$ where $R_\perp \coloneqq R_\perp^{\texttt{max}(y)}$ is the contradiction clause;
- the other clauses are computation clauses.

Let ψ' be the conjunction of those clauses and let $\mathbf{R}$ be the set of computation predicates, which are $R_{\texttt{min}(x)}, R_{\neg\texttt{min}(y)}, R_=, R_{\texttt{pred}}, R_<, R_\leq$, W_s^y, W_s^y, for $s \in \Sigma$, $\texttt{Border}$, $\texttt{Border}^{\texttt{x}-1}$ and $R_\perp^{\texttt{max}(y)}$. The formula obtained $\exists\mathbf{R}\forall x\forall y\psi'$ is the equivalent normal form of the formula $\Phi_{\texttt{Unbordered}}$ in $\texttt{pred-ESO-HORN}$.

The reader may object that although we have tried to present the final list of (normalized) clauses in the least artificial way possible, it is much longer and less readable than the original (short) list of clauses (0,a), (0,b) and (0,c). It means that while the expression of a problem in $\texttt{pred-ESO-HORN}$ (a high-level language) is natural and synthetic, the only interest of its normal form lies in its ability to be translated into an automaton in a straightforward way, as formally established in the next sections.

A real real-timetime CA recognizing $\texttt{Unbordered}$*:*

Now that the formula $\Phi_{\texttt{Unbordered}}$ has been normalized, we can easily deduce from it a grid circuit recognizing the language $\texttt{Unbordered}$. A computation example on this grid circuit and its transformation into a computation of a real-time CA is shown in Figure 6.



(a) On a grid circuit

(b) On a real-time CA

Figure 6: Computation of the word $abbaabb \notin$ `Unbordered`

After this introduction example of the normalization of a specific formula (with its final transformation into an automaton), let us describe and justify in the next Sections 4 and 5 how to normalize our logics in the general case.

4. Normalizing our Horn logics

The most difficult and main parts of the proofs of our descriptive complexity results, i.e., equalities 1-3 of Subsection 2.3, are the following *normalization lemmas*.

Lemma 1 (normalization of predecessor logics)**.** *Each formula* $\Phi \in$ `pred-ESO-HORN` *(resp.* $\Phi \in$ `pred-dio-ESO-HORN`*) is equivalent to a formula* $\Phi' \in$ `pred-ESO-HORN` *(resp.* $\Phi' \in$ `pred-dio-ESO-HORN`*) where each clause is of one of the following forms:*

- input clause *of the form, for* $s \in \Sigma$ *and* $R \in \mathbf{R}$*:*
  $\texttt{min}(x) \wedge \texttt{min}(y) \wedge Q_s(y) \to R(x,y)$*, or* $\texttt{min}(x) \wedge \neg\texttt{min}(y) \wedge Q_s(y) \to R(x,y)$
  (resp. $x = y \wedge \texttt{min}(x) \wedge Q_s(x) \to R(x,y)$*, or* $x = y \wedge \neg\texttt{min}(x) \wedge Q_s(x) \to R(x,y)$*);*
- *the* contradiction clause*, for a fixed* $R_\perp \in \mathbf{R}$*:* $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\perp(x,y) \to \perp$*;*
- computation clause *of the form* $\delta_1 \wedge \ldots \wedge \delta_r \to R(x,y)$*, for* $R \in \mathbf{R}$*, where each hypothesis* δ_i *is a conjunction of the form* $S(x-1,y) \wedge \neg\texttt{min}(x)$ *or* $S(x,y-1) \wedge \neg\texttt{min}(y)$*, for* $S \in \mathbf{R}$*.*

Let `normal-pred-ESO-HORN` (resp. `normal-pred-dio-ESO-HORN`) denote the class of formulas (languages) so defined.

Lemma 2 (normalization of inclusion logic)**.** *Each formula* $\Phi \in$ `incl-ESO-HORN` *is equivalent to a formula* $\Phi' \in$ `incl-ESO-HORN` *where each clause is of one of the following forms:*

- input clause *of the form* $x = y \wedge Q_s(x) \to R(x,y)$*, for* $s \in \Sigma$ *and* $R \in \mathbf{R}$*;*
- *the* contradiction clause*, for a fixed* $R_\perp \in \mathbf{R}$*,* $\texttt{min}(x) \wedge \texttt{max}(y) \wedge R_\perp(x,y) \to \perp$*;*
- computation clause *of the form*[7] $x < y \wedge \delta_1 \wedge \ldots \wedge \delta_r \to R(x,y)$*, where* $R \in \mathbf{R}$ *and where each hypothesis* δ_i *is a computation atom of either form* $S(x+1,y)$ *or* $S(x,y-1)$*, for* $S \in \mathbf{R}$*.*

Let `normal-incl-ESO-HORN` denote the class of formulas (languages) so defined.

The normalization processes of our three logics are quite similar to each other; further, some steps are exactly the same. Therefore, we choose to present here below the successive normalization steps for one logic: `pred-ESO-HORN`. Afterwards, we will succinctly describe how those steps should be adapted for the two other logics.

[7] Note that the hypothesis $x < y$ is equivalent to the expected inequality $x + 1 \leq y$ or $x \leq y - 1$.

4.1. Normalizing predecessor Horn logic

Let a formula $\Phi \in$ `pred-ESO-HORN`. For simplicity of notation, we first assume that the only computation atoms of Φ are of the form $R(x-a, y-b)$, $a, b \geq 0$ (no atom of the form $R(y-b, x-a)$). We will show at the end of the proof how to manage the general case. Φ will be transformed into an equivalent normalized form Φ' by a sequence of 10 steps:

1. Processing the contradiction clauses;
2. Processing the input;
3. Restriction of computation atoms to $R(x-1, y)$, $R(x, y-1)$, and $R(x, y)$;
4. Elimination of atoms $x > a$, $x = a$, $y > a$, $y = a$;
5. Processing of `min` and `max`;
6. Defining equality and inequalities;
7. Folding the domain;
8. Deleting `max` in the initialization clauses;
9. From initialization clauses to input clauses;
10. Elimination of atoms $R(x, y)$ as hypotheses.

In each of those 10 steps, we will introduce *new* (binary) *computation predicates*, to be added to the set $\mathbf{R}$ of existentially quantified predicates, and new clauses to define them.

1. Processing the contradiction clauses. Without loss of generality, one can assume there is *only* the contradiction clause $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\perp(x, y) \rightarrow \perp$. Indeed, each contradiction clause $\ell_1 \wedge \ldots \wedge \ell_k \rightarrow \perp$ can be equivalently replaced by the conjunction of computation clauses $\ell_1 \wedge \ldots \wedge \ell_k \rightarrow R_\perp(x, y)$ with the clause $R_\perp(x, y) \rightarrow \perp$ where $R_\perp$ is a new computation predicate (intuitively, always false). However, in place of the previous clause, we "delay" the contradiction, by propagating predicate $R_\perp$ till point (n, n), thanks to the conjunction of the "transport" clauses $R_\perp(x-1, y) \wedge \neg\texttt{min}(x) \rightarrow R_\perp(x, y)$ and $R_\perp(x, y-1) \wedge \neg\texttt{min}(y) \rightarrow R_\perp(x, y)$ and of the unique contradiction clause $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\perp(x, y) \rightarrow \perp$ required by the normal form.

2. Processing the input. The idea is to make available the letters of the input word *only* on the sides $x = 1$ and $y = 1$ of the square $\{(x, y) \in [1, n]^2\}$, this by carrying out their transport thanks to new "transport" predicates W_s^x and W_s^y, for $s \in \Sigma$, inductively defined by the following clauses:

initialization clauses: $Q_s(x) \wedge \texttt{min}(y) \rightarrow W_s^x(x, y)$ and $Q_s(y) \wedge \texttt{min}(x) \rightarrow W_s^y(x, y)$;

transport clauses: $W_s^x(x, y-1) \wedge \neg\texttt{min}(y) \rightarrow W_s^x(x, y)$ and $W_s^y(x-1, y) \wedge \neg\texttt{min}(x) \rightarrow W_s^y(x, y)$.

By transitivity, these clauses imply clauses $Q_s(x) \rightarrow W_s^x(x, y)$ and $Q_s(y) \rightarrow W_s^y(x, y)$. In other words, the minimal model of the conjunction of those clauses that expands structure $\langle w \rangle$ satisfies equivalences $\forall x \forall y\ (W_s^x(x, y) \iff Q_s(x))$ and $\forall x \forall y\ (W_s^y(x, y) \iff Q_s(y))$. This justifies the replacement of the input atoms of form $Q_s(x-a)$ and $Q_s(y-b)$ by the respective atoms $W_s^x(x-a, y)$ and $W_s^y(x, y-b)$ in all the clauses, except in the initialization clauses.

3. Restriction of computation atoms to $R(x-1, y)$, $R(x, y-1)$, $R(x, y)$. The idea is to introduce new "shift" predicates R^{x-a}, R^{y-b} and $R^{x-a, y-b}$, for fixed integers $a, b > 0$ and $R \in \mathbf{R}$. We define the predicate $R^{x-a, y-b}$ that intuitively satisfies the equivalence $R^{x-a, y-b}(x, y) \iff R(x-a, y-b)$.[8] Let us explain the method by an example. Assume we have initially the Horn clause

$$x > 3 \wedge y > 2 \wedge R(x-2, y-1) \wedge S(x-3, y-2) \rightarrow T(x, y).$$

[8] As Remark 5 says, we use the semantics of the *minimal model* of Horn formulas. Here, we assert that the equivalence $R^{x-a, y-b}(x, y) \iff R(x-a, y-b)$ holds in the minimal model of the Horn formula we construct.

This clause is replaced by the clause

$$x > 3 \wedge y > 2 \wedge R^{x-2}(x, y-1) \wedge S^{x-2,y-2}(x-1, y) \rightarrow T(x, y),$$

for which the predicates R^{x-1} and R^{x-2} are defined by the clauses $x > 1 \wedge R(x-1, y) \rightarrow R^{x-1}(x, y)$ and $x > 2 \wedge R^{x-1}(x-1, y) \rightarrow R^{x-2}(x, y)$ which imply $x > 2 \wedge R(x-2, y) \rightarrow R^{x-2}(x, y)$ and then $x > 2 \wedge y > 1 \wedge R(x-2, y-1) \rightarrow R^{x-2}(x, y-1)$, and the predicates S^{x-1}, S^{x-2}, $S^{x-2,y-1}$ and $S^{x-2,y-2}$ defined by the respective clauses: $x > 1 \wedge S(x-1, y) \rightarrow S^{x-1}(x, y)$, $x > 2 \wedge S^{x-1}(x-1, y) \rightarrow S^{x-2}(x, y)$, $x > 2 \wedge y > 1 \wedge S^{x-2}(x, y-1) \rightarrow S^{x-2,y-1}(x, y)$, and $x > 2 \wedge y > 2 \wedge S^{x-2,y-1}(x, y-1) \rightarrow S^{x-2,y-2}(x, y)$, which imply together the clause $x > 2 \wedge y > 2 \wedge S(x-2, y-2) \rightarrow S^{x-2,y-2}(x, y)$ and then also $x > 3 \wedge y > 2 \wedge S(x-3, y-2) \rightarrow S^{x-2,y-2}(x-1, y)$.

Remark 10. *Atoms on* `min` *and* x *are of the forms* $\texttt{min}(x - a)$ *or* $\neg\texttt{min}(x - a)$*, for* $a \geq 0$*, or, equivalently,* $x \leq a + 1$ *or* $x > a + 1$*. Besides, for each integer* $a \geq 1$*, the atom* $\texttt{max}(x - a)$ *is false. Therefore, one may consider that the only literals on* x *involving* `min` *or* `max` *are of the form* $\texttt{min}(x)$*,* $\neg\texttt{min}(x)$*,* $\texttt{max}(x)$*,* $\neg\texttt{max}(x)$*,* $x \leq a$*,* $x > a$*, for an integer* $a > 1$*, and similarly, for* y*.*

4. Elimination of atoms $x > a$*,* $x \leq a$*,* $y > a$*,* $y \leq a$*.* By recurrence on integer $a \geq 1$, let us define the *binary* predicates $R^{x>a}$ (and, similarly, $R^{x\leq a}$, $R^{y>a}$, $R^{y\leq a}$) whose intuitive meaning is $x > a$ (resp. $x \leq a$, $y > a$, $y \leq a$). The predicate $R^{x>1}$ is defined by the clause $\neg\texttt{min}(x) \rightarrow R^{x>1}(x, y)$. For $a > 1$, let us define $R^{x>a}$ from $R^{x>a-1}$ by the clause $R^{x>a-1}(x-1, y) \wedge \neg\texttt{min}(x) \rightarrow R^{x>a}(x, y)$. By recurrence on integer $a \geq 1$, these clauses imply $x > a \rightarrow R^{x>a}(x, y)$. This justifies the replacement of the atoms $x > a$ and $x \leq a$, for $a > 1$, by $R^{x>a}(x, y)$ and $R^{x\leq a}(x, y)$, respectively, and similarly for y in place of x.

After Step 4, the only literals involving `min` or `max` are $(\neg)\texttt{min}(x)$, $(\neg)\texttt{max}(x)$, $(\neg)\texttt{min}(y)$, $(\neg)\texttt{max}(y)$.

5. Processing of `min` *and* `max`*.* To each literal $\eta(x)$ of the form $\texttt{min}(x)$, $\neg\texttt{min}(x)$, $\texttt{max}(x)$ or $\neg\texttt{max}(x)$, associate the new binary predicate $R^{\eta(x)}$ defined by the conjunction of the *initialization clause* $\eta(x) \wedge \texttt{min}(y) \rightarrow R^{\eta(x)}(x, y)$ and of the *transport clause* $R^{\eta(x)}(x, y-1) \wedge \neg\texttt{min}(y) \rightarrow R^{\eta(x)}(x, y)$. Do similarly for the literals $\eta(y) \in \{(\neg)\texttt{min}(y), (\neg)\texttt{max}(y)\}$. This justifies the replacement of each such literal $\eta(x)$ (resp. $\eta(y)$) by the "equivalent" atom $R^{\eta(x)}(x, y)$ (resp. $R^{\eta(y)}(x, y)$) in all the clauses, except in the above initialization clauses and in the contradiction clause or in case $\eta(x)$ (resp. $\eta(y)$) is $\neg\texttt{min}(x)$ (resp. $\neg\texttt{min}(y)$) and is joined to a hypothesis of the form $R(x-1, y)$ (resp. $R(x, y-1)$).

Recapitulation. After Step 5 each clause is of one of the following forms:

1. an *initialization clause* of one of the two forms:
   - $\texttt{min}(x) \wedge \eta(y) \rightarrow R(x, y)$, with $\eta(y) \in \{(Q_s(y))_{s\in\Sigma}, (\neg)\texttt{min}(y), (\neg)\texttt{max}(y)\}$;
   - $\texttt{min}(y) \wedge \eta(x) \rightarrow R(x, y)$, with $\eta(x) \in \{(Q_s(x))_{s\in\Sigma}, (\neg)\texttt{min}(x), (\neg)\texttt{max}(x)\}$;
2. "the" *contradiction clause* $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\perp(x, y) \rightarrow \perp$;
3. a *computation clause* of the form $\delta_1(x, y) \wedge \ldots \wedge \delta_r(x, y) \rightarrow R(x, y)$, where each hypothesis δ_i is of one of the three forms $R(x, y)$, $R(x-1, y) \wedge \neg\texttt{min}(x)$, $R(x, y-1) \wedge \neg\texttt{min}(y)$. In fact, without loss of generality, we can *assume* that each computation clause is of one of the following forms:
   (a) $S(x-1, y) \wedge \neg\texttt{min}(x) \rightarrow R(x, y)$;
   (b) $S(x, y-1) \wedge \neg\texttt{min}(y) \rightarrow R(x, y)$;
   (c) $S(x, y) \wedge T(x, y) \rightarrow R(x, y)$.

Justification of the assumption: "Decompose" each computation clause into clauses of forms (a,b,c) by introducing new intermediate predicates. For example, the computation clause $R_1(x-1, y) \wedge \neg\texttt{min}(x) \wedge R_2(x, y-1) \wedge \neg\texttt{min}(y) \wedge R_3(x, y) \rightarrow R_4(x, y)$ is "equivalent" to the conjunction of the following clauses using new predicates R_5, R_6, R_7: $R_1(x-1, y) \wedge \neg\texttt{min}(x) \rightarrow R_5(x, y)$; $R_2(x, y-1) \wedge \neg\texttt{min}(y) \rightarrow R_6(x, y)$; $R_5(x, y) \wedge R_6(x, y) \rightarrow R_7(x, y)$; $R_7(x, y) \wedge R_3(x, y) \rightarrow R_4(x, y)$.

We now plan to fold the square domain $\{(x, y) \in [1, n]^2\}$ along the diagonal $x = y$ on the *overdiagonal triangle* $T_n = \{(x, y) \in [1, n]^2 |\ x \leq y\}$. This requires to first define equality and inequalities.

6. Defining equality and inequalities. Let us jointly define the predicates $R_=$ and $R_{\mathtt{pred}}$ of intuitive meaning $R_=(x,y) \iff x = y$ and $R_{\mathtt{pred}}(x,y) \iff x - 1 = y$ by the following clauses: $\mathtt{min}(x) \wedge \mathtt{min}(y) \rightarrow R_=(x,y)$; $\neg\mathtt{min}(x) \wedge R_=(x-1,y) \rightarrow R_{\mathtt{pred}}(x,y)$; $\neg\mathtt{min}(y) \wedge R_{\mathtt{pred}}(x,y-1) \rightarrow R_=(x,y)$. Then define the predicate $R_<$ such that $R_<(x,y) \iff x < y$ with the two clauses $\neg\mathtt{min}(y) \wedge R_=(x,y-1) \rightarrow R_<(x,y)$ and $\neg\mathtt{min}(y) \wedge R_<(x,y-1) \rightarrow R_<(x,y)$. Define similarly the predicate $R_\leq$ such that $R_\leq(x,y) \iff x \leq y$ with the two clauses $\mathtt{min}(x) \wedge \mathtt{min}(y) \rightarrow R_\leq(x,y)$ and $\neg\mathtt{min}(x) \wedge R_<(x-1,y) \rightarrow R_\leq(x,y)$.

For easy reading, we will freely write $x = y$, $x < y$ and $x \leq y$ in place of the atoms $R_=(x,y)$, $R_<(x,y)$ and $R_\leq(x,y)$, respectively.

7. Folding the domain. Let us fold the square domain $\{(x,y) \in [1,n]^2\}$ along the diagonal $x = y$ on the overdiagonal triangle $T_n = \{(x,y) \in [1,n]^2 |\ x \leq y\}$ so that each point (y,x) such that $x \leq y$ is sent to its symmetrical point $(x,y) \in T_n$. For that purpose, let us associate to each predicate $R \in \mathbf{R}$ a new (inverse) predicate $R^{\mathtt{inv}}$ whose intuitive meaning is the following: for each $x \leq y$, we have $R^{\mathtt{inv}}(x,y) \iff R(y,x)$. So, each clause C will be replaced by two clauses: the first one is the restriction of C to the triangle T_n; the second one is the *folding* on T_n of the restriction of C to the *subdiagonal triangle* using predicates $R^{\mathtt{inv}}$. Finally, we will express that each $R \in \mathbf{R}$ coincides with its inverse $R^{\mathtt{inv}}$ on the fold $x = y$.

Folding the initialization clauses: Each initialization clause of the form $\mathtt{min}(x) \wedge \eta(y) \rightarrow R(x,y)$ (with $\eta(y) \in \{Q_s(y) | s \in \Sigma\} \cup \{(\neg)\mathtt{min}(y), (\neg)\mathtt{max}(y)\}$) applies to the line $x = 1$ which is included in the triangle T_n and consequently it should be *unchanged* in the folding; in contrast, each initialization clause of the form $\mathtt{min}(y) \wedge \eta(x) \rightarrow R(x,y)$ (with $\eta(x) \in \{Q_s(x) | s \in \Sigma\} \cup \{(\neg)\mathtt{min}(x), (\neg)\mathtt{max}(x)\}$) is *replaced* by its *folded* version $\mathtt{min}(x) \wedge \eta(y) \rightarrow R^{\mathtt{inv}}(x,y)$.

Folding the computation clauses: Let us describe how to fold the clauses (a) and (b) (to fold clauses (c) is easy):

- *Folding clauses (a)*: A clause of the form $S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R(x,y)$ is equivalent to the conjunction of clauses i) $x \leq y \wedge S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R(x,y)$ and ii) $x > y \wedge S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R(x,y)$. Notice that clause (i) applies to the triangle T_n since $x \leq y$ implies $x - 1 < y$: therefore, clause (i) should be left *unchanged*. Clause (ii) is equivalent (by exchanging variables x and y) to the clause $y > x \wedge S(y-1,x) \wedge \neg\mathtt{min}(y) \rightarrow R(y,x)$ whose folded (equivalent) form on T_n is $x < y \wedge S^{\mathtt{inv}}(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R^{\mathtt{inv}}(x,y)$ since $x < y$ implies $x \leq y - 1$.
- *Folding clauses (b)*: Similarly, a clause of the form $S(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$ is equivalent to the conjunction of clauses $x < y \wedge S(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$ and $x \leq y \wedge S^{\mathtt{inv}}(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R^{\mathtt{inv}}(x,y)$.

Folding the contradiction clause: Clearly, it is harmless to confuse the (contradiction) predicate $R_\perp$ and its inverse $(R_\perp)^{\mathtt{inv}}$; consequently, the contradiction clause itself $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge R_\perp(x,y) \rightarrow \perp$ is its own folded version.

The diagonal fold: Finally, for each $R \in \mathbf{R}$, the following two clauses mean that R coincides with its inverse $R^{\mathtt{inv}}$ on the diagonal: $x = y \wedge R(x,y) \rightarrow R^{\mathtt{inv}}(x,y)$; $x = y \wedge R^{\mathtt{inv}}(x,y) \rightarrow R(x,y)$.

Recapitulation. By a careful examination of the set of clauses obtained after Steps 1-7, we can check that each of them is of one of the following forms:

1. an *initialization clause* of the form $\mathtt{min}(x) \wedge \eta(y) \rightarrow R(x,y)$ with $\eta(y) \in \{Q_s(y) | s \in \Sigma\} \cup \{(\neg)\mathtt{min}(y), (\neg)\mathtt{max}(y)\}$;
2. "the" *contradiction clause* $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge R_\perp(x,y) \rightarrow \perp$;
3. a *computation clause* of one of the following forms:
   (a) $x \leq y \wedge S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R(x,y)$;
   (b) $x < y \wedge S(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$;
   (c) $x \leq y \wedge S(x,y) \wedge T(x,y) \rightarrow R(x,y)$;
   (d) $x = y \wedge S(x,y) \rightarrow R(x,y)$.

8. Deleting $\mathtt{max}$ *in the initialization clauses.* The idea is to consider in parallel for each point (x, y) the case where the hypothesis $\mathtt{max}(y)$ holds and the opposite case where the negation $\neg\mathtt{max}(y)$ holds. For that purpose, we duplicate each computation predicate R in two new predicates called $R^y_{\leftarrow\mathtt{max}}$ and $R^y_{\leftarrow\neg\mathtt{max}}$. Intuitively, the atom $R^y_{\leftarrow\mathtt{max}}(x, y)$ (resp. $R^y_{\leftarrow\neg\mathtt{max}}(x, y)$) expresses the implication $\mathtt{max}(y) \rightarrow R(x, y)$ (resp. $\neg\mathtt{max}(y) \rightarrow R(x, y)$).

Transforming the initialization clauses: According to the desired semantics of $R^y_{\leftarrow\mathtt{max}}$ and $R^y_{\leftarrow\neg\mathtt{max}}$, each initialization clause of the form $\mathtt{min}(x) \wedge \mathtt{max}(y) \rightarrow R(x, y)$ (resp. $\mathtt{min}(x) \wedge \neg\mathtt{max}(y) \rightarrow R(x, y)$) should be rewritten as $\mathtt{min}(x) \rightarrow R^y_{\leftarrow\mathtt{max}}(x, y)$ (resp. $\mathtt{min}(x) \rightarrow R^y_{\leftarrow\neg\mathtt{max}}(x, y)$). Similarly, each initialization clause of the form $\mathtt{min}(x) \wedge \eta(y) \rightarrow R(x, y)$, for $\eta(y) \in \{Q_s(y)|s \in \Sigma\} \cup \{(\neg)\mathtt{min}(y)\}$ should be replaced by the conjunction of the following two clauses: $\mathtt{min}(x) \wedge \eta(y) \rightarrow R^y_{\leftarrow\mathtt{max}}(x, y)$ and $\mathtt{min}(x) \wedge \eta(y) \rightarrow R^y_{\leftarrow\neg\mathtt{max}}(x, y)$.

Transforming the computation clauses: We describe it for each above form (a-d).

- Each clause (a) $x \leq y \wedge S(x-1, y) \wedge \neg\mathtt{min}(x) \rightarrow R(x, y)$ is replaced by the "equivalent" conjunction of the following two clauses $x \leq y \wedge S^y_{\leftarrow\mathtt{max}}(x-1, y) \wedge \neg\mathtt{min}(x) \rightarrow R^y_{\leftarrow\mathtt{max}}(x, y)$ and $x \leq y \wedge S^y_{\leftarrow\neg\mathtt{max}}(x-1, y) \wedge \neg\mathtt{min}(x) \rightarrow R^y_{\leftarrow\neg\mathtt{max}}(x, y)$.
- Each clause (b) $x < y \wedge S(x, y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x, y)$ is "equivalent" to $x < y \wedge S^y_{\leftarrow\neg\mathtt{max}}(x, y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x, y)$ since the hypothesis $\neg\mathtt{max}(y-1)$ always holds. Consequently, clause (b) should be replaced by the "equivalent" conjunction of the following two clauses:

  $x < y \wedge S^y_{\leftarrow\neg\mathtt{max}}(x, y-1) \wedge \neg\mathtt{min}(y) \rightarrow R^y_{\leftarrow\mathtt{max}}(x, y)$, and
  $x < y \wedge S^y_{\leftarrow\neg\mathtt{max}}(x, y-1) \wedge \neg\mathtt{min}(y) \rightarrow R^y_{\leftarrow\neg\mathtt{max}}(x, y)$.
- Each clause (c) $x \leq y \wedge S(x, y) \wedge T(x, y) \rightarrow R(x, y)$ is replaced by the "equivalent" conjunction of the following two clauses: $x \leq y \wedge S^y_{\leftarrow\mathtt{max}}(x, y) \wedge T^y_{\leftarrow\mathtt{max}}(x, y) \rightarrow R^y_{\leftarrow\mathtt{max}}(x, y)$ and $x \leq y \wedge S^y_{\leftarrow\neg\mathtt{max}}(x, y) \wedge T^y_{\leftarrow\neg\mathtt{max}}(x, y) \rightarrow R^y_{\leftarrow\neg\mathtt{max}}(x, y)$.
- Perform a similar substitution for each above clause (d).

Processing the contradiction clause: Obviously, the contradiction clause $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge R_\perp(x, y) \rightarrow \perp$ is equivalent to the formula $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge (\mathtt{max}(y) \rightarrow R_\perp(x, y)) \rightarrow \perp$ and should be rewritten as $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge (R_\perp)^y_{\leftarrow\mathtt{max}}(x, y) \rightarrow \perp$, which is of the required form if the predicate $(R_\perp)^y_{\leftarrow\mathtt{max}}$ is renamed $R_\perp$.

9. From initialization clauses to input clauses. The *initialization clauses* are now of the form $\mathtt{min}(x) \rightarrow R(x, y)$ or $\mathtt{min}(x) \wedge \eta(y) \rightarrow R(x, y)$, for $\eta(y) \in \{Q_s(y)|s \in \Sigma\} \cup \{(\neg)\mathtt{min}(y)\}$. By a case analysis, it is easy to transform each of these clauses into an equivalent conjunction of *input clauses* of the required (normalized) forms: $\mathtt{min}(x) \wedge \mathtt{min}(y) \wedge Q_s(y) \rightarrow R(x, y)$; $\mathtt{min}(x) \wedge \neg\mathtt{min}(y) \wedge Q_s(y) \rightarrow R(x, y)$.

After Step 9, the formula obtained is of the claimed normal form, *except* that some computation clauses may have atoms $R(x, y)$ as hypotheses. Our last step is to eliminate such hypotheses.

10. Elimination of atoms $R(x, y)$ *as hypotheses.* This is the most technical step but the general ideas are rather simple. The first idea is to group together in each computation clause the hypotheses of the clause of the form $R(x, y)$ and its conclusion. Accordingly, the formula can be rewritten in the form

$$\Phi := \exists\mathbf{R}\forall x\forall y \left[\bigwedge_i C_i(x, y) \wedge \bigwedge_{i\in[1,k]} (\alpha_i(x, y) \rightarrow \theta_i(x, y))\right]$$

where the C_i's are the input clauses and the contradiction clause, and each computation clause is written in the form $\alpha_i(x, y) \rightarrow \theta_i(x, y)$ where

- $\alpha_i(x, y)$ is a conjunction of formulas of the only forms $R(x-1, y) \wedge \neg\mathtt{min}(x)$, $R(x, y-1) \wedge \neg\mathtt{min}(y)$, *but not* $R(x, y)$,
- and $\theta_i(x, y)$ is a Horn clause whose *all* atoms are of the form $R(x, y)$.

The second idea is to "solve" the Horn clauses θ_i (containing *only* atoms of the form $R(x,y)$) according to the input clauses and *all the possible* conjunctions of hypotheses α_i that may be true. Notice the two following facts: the hypotheses of the input clauses are input literals and the conjuncts of the α_i's have the only forms $R(x-1,y) \wedge \neg\mathtt{min}(x)$ and $R(x,y-1) \wedge \neg\mathtt{min}(y)$. Roughly expressed, we will prove by induction on the sum values $x+y \in [2,2n]$ that the obtained formula Φ' which is a conjunction of clauses (whose hypotheses no longer include any atom of the form $R(x,y)$) is equivalent to formula Φ.

We now give the transformation process in detail with its proof. Let us number $R_1, \ldots, R_m$ the computation predicates of $\mathbf{R}$. To each subset $J \subseteq [1,k]$ of the family of implications $(\alpha_i(x,y) \to \theta_i(x,y))_{i\in[1,k]}$ let us associate the set

$$K_J := \{h \in [1,m] \mid \textstyle\bigwedge_{i\in J} \theta_i(x,y) \to R_h(x,y) \text{ is a tautology}\}.$$

Note that the notion of *tautology* used in the definition of K_J is purely "propositional" because all the atoms involved are of the form $R_i(x,y)$, i.e., refer to the same pair of variables (x,y). Also, note that the function $J \mapsto K_J$ is *monotonous*: for $J' \subseteq J$, we have $K_{J'} \subseteq K_J$ because $\bigwedge_{i\in J'} \theta_i(x,y) \to R_h(x,y)$ implies $\bigwedge_{i\in J} \theta_i(x,y) \to R_h(x,y)$.

Clearly, it is enough to prove the following claim:

Claim 1. *The formula* Φ *is equivalent to the* normalized *formula*

$$\Phi' := \exists\mathbf{R}\forall x\forall y \left[\bigwedge_i C_i(x,y) \wedge \bigwedge_{J\subseteq[1,k]} \bigwedge_{h\in K_J} \left(\bigwedge_{i\in J} \alpha_i(x,y) \to R_h(x,y)\right)\right].$$

Notation. *The formula between brackets can be rewritten*

$$\bigwedge_i C_i(x,y) \wedge \bigwedge_{J\subseteq[1,k]} \left(\bigwedge_{i\in J} \alpha_i(x,y) \to \bigwedge_{h\in K_J} R_h(x,y)\right).$$

Proof of the implication $\Phi \Rightarrow \Phi'$. It is enough to prove the implication

$$\left[\bigwedge_{i\in[1,k]} (\alpha_i(x,y) \to \theta_i(x,y))\right] \to \left[\bigwedge_{i\in J} \alpha_i(x,y) \to \bigwedge_{h\in K_J} R_h(x,y)\right]$$

for all set $J \subseteq [1,k]$ and all $h \in K_J$.
The implication to be proved can be equivalently written:

$$\left[\bigwedge_{i\in J} \alpha_i(x,y) \wedge \bigwedge_{i\in[1,k]} (\alpha_i(x,y) \to \theta_i(x,y))\right] \to \bigwedge_{h\in K_J} R_h(x,y).$$

This implication is a straightforward consequence of the two following facts:

- The sub-formula between brackets above implies the conjunction $\bigwedge_{i\in J} \theta_i(x,y)$.
- As the implication $\bigwedge_{i\in J} \theta_i(x,y) \to \bigwedge_{h\in K_J} R_h(x,y)$ is a tautology (by definition of K_J), the implication to be proved is a tautology too.

□

The converse implication $\Phi' \Rightarrow \Phi$ is more difficult to prove. It uses a folklore property of propositional Horn formulas easy to prove:

Lemma 3 (Horn property [22])**.** *Let* F *be a* strict Horn formula *of propositional calculus, that is a conjunction of clauses of the form* $p_1 \wedge \ldots \wedge p_k \to p_0$ *where* $k \geq 0$ *and the* p_i*'s are propositional variables. Let* F' *be the conjunction of propositional variables* q *such that the implication* $F \to q$ *is a tautology.* F *has the same minimal model*[9] *as* F'.

[9]For example, for $F := p_1 \wedge p_3 \wedge (p_1 \wedge p_3 \to p_5) \wedge (p_1 \wedge p_2 \to p_4)$, we have $F' := p_1 \wedge p_3 \wedge p_5$ which has the same minimal model I as F; this model is given by $I(p_1) = I(p_3) = I(p_5) = 1$ and $I(p_2) = I(p_4) = 0$.

Proof of the implication $\Phi' \Rightarrow \Phi$. Let $\langle w \rangle$ be a model of Φ' and let $(\langle w \rangle, \mathbf{R})$ be the minimal model of the Horn formula

$$\varphi' := \forall x \forall y \left[\bigwedge_i C_i(x,y) \wedge \bigwedge_{J \subseteq [1,k]} \bigwedge_{h \in K_J} \left(\bigwedge_{i \in J} \alpha_i(x,y) \rightarrow R_h(x,y) \right) \right].$$

It is enough to show that $(\langle w \rangle, \mathbf{R})$ also satisfies the formula

$$\varphi := \forall x \forall y \left[\bigwedge_i C_i(x,y) \wedge \bigwedge_{i \in [1,k]} (\alpha_i(x,y) \rightarrow \theta_i(x,y)) \right]$$

As each α_i is a conjunction of formulas of the form $R(x-1,y) \wedge \neg \mathtt{min}(x)$, or $R(x,y-1) \wedge \neg \mathtt{min}(y)$, all the computation atoms of the conjunctions α_i are of the form $R(x-1,y)$ (with also the conjunct $\neg \mathtt{min}(x)$) or $R(x,y-1)$ (with also the conjunct $\neg \mathtt{min}(y)$) we make an induction on the domain $\{(a,b) \in [1,n]^2 \mid a+b \leq t\}$, for $t \in [1,2n]$. More precisely, we are going to prove, by recurrence on the integer $t \in [1,2n]$, that the minimal model $(\langle w \rangle, \mathbf{R})$ of φ' satisfies the "relativized" formula φ_t of the formula φ defined by

$$\varphi_t := \forall x \forall y \left[x+y \leq t \rightarrow \left[\bigwedge_i C_i(x,y) \wedge \bigwedge_{i \in [1,k]} (\alpha_i(x,y) \rightarrow \theta_i(x,y)) \right] \right]$$

As the hypothesis $x+y \leq 2n$ holds for all x,y in the domain $[1,n]$, φ_{2n} is equivalent to φ on the structure $(\langle w \rangle, \mathbf{R})$.

Basis case: For $t=1$ the set $\{(a,b) \in [1,n]^2 \mid a+b \leq t\}$ is empty so that the "relativized" formula φ_1 is trivially true in the minimal model $(\langle w \rangle, \mathbf{R})$ of φ'.

Recurrence step: Suppose $(\langle w \rangle, \mathbf{R}) \models \varphi_{t-1}$, for an integer $t \in [2,2n]$. It is enough to show that, for each couple $(a,b) \in [1,n]^2$ such that $a+b=t$, we have $(\langle w \rangle, \mathbf{R}) \models \bigwedge_{i \in [1,k]} (\alpha_i(a,b) \rightarrow \theta_i(a,b))$. Let $J_{a,b}$ be the set of indices $i \in [1,k]$ such that the couple (a,b) satisfies α_i:

$$J_{a,b} := \{i \in [1,k] \mid (\langle w \rangle, \mathbf{R}) \models \alpha_i(a,b)\}.$$

Recall that each $\alpha_i(a,b)$ is a (possibly empty) conjunction of atoms $R(a',b')$ with $(a',b') = (a-1,b)$ or $(a',b') = (a,b-1)$, therefore such that $a'+b' = t-1$. This is the basis of the inductive construction of the minimal model $(\langle w \rangle, \mathbf{R})$ of the Horn formula φ_t and of the proof by recurrence. Let a set $J \subseteq [1,k]$. Let us examine the two possible cases:

(1) $J \subseteq J_{a,b}$: then the conjunction $\bigwedge_{i \in J} \alpha_i(a,b)$ holds in $(\langle w \rangle, \mathbf{R})$; hence, in $(\langle w \rangle, \mathbf{R})$, the conjunction $\bigwedge_{h \in K_J} \left(\bigwedge_{i \in J} \alpha_i(a,b) \rightarrow R_h(a,b) \right)$ is equivalent to $\bigwedge_{h \in K_J} R_h(a,b)$;

(2) $J \setminus J_{a,b} \neq \emptyset$: then the conjunction $\bigwedge_{i \in J} \alpha_i(a,b)$ is false in $(\langle w \rangle, \mathbf{R})$; hence, the conjunction $\bigwedge_{h \in K_J} (\bigwedge_{i \in J} \alpha_i(a,b) \rightarrow R_h(a,b))$ holds in $(\langle w \rangle, \mathbf{R})$.

From (1) and (2), we deduce that in $(\langle w \rangle, \mathbf{R})$ the conjunction

$$\bigwedge_{J \subseteq [1,k]} \bigwedge_{h \in K_J} \left(\bigwedge_{i \in J} \alpha_i(a,b) \rightarrow R_h(a,b) \right)$$

is equivalent to the conjunction $\bigwedge_{J \subseteq J_{a,b}} \bigwedge_{h \in K_J} R_h(a,b)$, which can be simplified as $\bigwedge_{h \in K_{J_{a,b}}} R_h(a,b)$ because $J \subseteq J_{a,b}$ implies $K_J \subseteq K_{J_{a,b}}$. Consequently, for all $h \in [1,m]$, the minimal model $(\langle w \rangle, \mathbf{R})$ of the Horn formula φ' satisfies the atom $R_h(a,b)$ *iff* h belongs to $K_{J_{a,b}}$. By definition,

$$K_{J_{a,b}} := \{h \in [1,m] \mid \textstyle\bigwedge_{i \in J_{a,b}} \theta_i(x,y) \rightarrow R_h(x,y) \text{ is a tautology}\}.$$

As a consequence of Lemma 3, the two conjunctions $\bigwedge_{i\in J_{a,b}} \theta_i(a,b)$ and $\bigwedge_{h\in K_{J_{a,b}}} R_h(a,b)$ have the same minimal model, which is also the restriction of the minimal model $(\langle w\rangle, \mathbf{R})$ of φ' to the set of atoms $R_h(a,b)$, for $h \in [1,m]$. Therefore, if $i \in J_{a,b}$, then $(\langle w\rangle, \mathbf{R}) \models \theta_i(a,b)$. If $i \in [1,k] \setminus J_{a,b}$, then we have $(\langle w\rangle, \mathbf{R}) \models \neg\alpha_i(a,b)$, by the definition of $J_{a,b}$. Therefore, for all $i \in [1,k]$, we get $(\langle w\rangle, \mathbf{R}) \models \neg\alpha_i(a,b) \vee \theta_i(a,b)$. In other words, for all (a,b) such that $a+b=t$:

$$(\langle w\rangle, \mathbf{R}) \models \bigwedge_{i\in[1,k]} (\alpha_i(a,b) \to \theta_i(a,b))$$

and then $(\langle w\rangle, \mathbf{R}) \models \varphi_t$.

This concludes the inductive proof that $(\langle w\rangle, \mathbf{R}) \models \varphi_t$, for all $t \in [1,2n]$, and then $\langle w\rangle \models \Phi$. This proves the implication $\Phi' \Rightarrow \Phi$ and completes the proof of Claim 1, hence the justification of Step 10. □

General case of Lemma 1. Steps 1-6 are easy to adapt in the general case where the initial formula may contain hypotheses of the form $R(y-b, x-a)$. The new points are the following: Step 3 restricts the computation atoms to four forms: $R(x,y)$, $R(y,x)$, $R(x-1,y)$ and $R(x,y-1)$; the key point is the adaptation of Step 7 (folding the domain) so that it eliminates the atoms of the form $R(y,x)$. Without loss of generality, assume that each computation clause is of one of the following forms:

(a) $S(x-1,y) \wedge \neg\mathtt{min}(x) \to R(x,y)$;

(b) $S(x,y-1) \wedge \neg\mathtt{min}(y) \to R(x,y)$;

(c) $S(x,y) \wedge T(x,y) \to R(x,y)$;

(d) $S(y,x) \to R(x,y)$.

Here again, this is obtained by "decomposing" each computation clause into an "equivalent" conjunction of clauses using new intermediate predicates. For instance, the computation clause $R_1(x-1,y) \wedge \neg\mathtt{min}(x) \wedge R_2(x,y-1) \wedge \neg\mathtt{min}(y) \wedge R_3(y,x) \to R_4(x,y)$ is "equivalent" to the conjunction of the following clauses using the new predicates R_5, R_6, R_7, R_8: $R_1(x-1,y) \wedge \neg\mathtt{min}(x) \to R_5(x,y)$; $R_2(x,y-1) \wedge \neg\mathtt{min}(y) \to R_6(x,y)$; $R_5(x,y) \wedge R_6(x,y) \to R_7(x,y)$; $R_3(y,x) \to R_8(x,y)$; $R_7(x,y) \wedge R_8(x,y) \to R_4(x,y)$.

The folding of clauses (a-c) is not modified. Let us describe how to fold the (new) clauses (d): $S(y,x) \to R(x,y)$. Obviously, such a clause is equivalent to the conjunction of the two clauses

(i) $x \le y \wedge S(y,x) \to R(x,y)$ and

(ii) $y \le x \wedge S(y,x) \to R(x,y)$.

The equivalent "folded" form of clause (i) is $x \le y \wedge S^{\mathtt{inv}}(x,y) \to R(x,y)$. Clause (ii) is equivalent to the clause $x \le y \wedge S(x,y) \to R(y,x)$ whose equivalent "folded" form is $x \le y \wedge S(x,y) \to R^{\mathtt{inv}}(x,y)$. Finally, Steps 8-10 are not modified. So the equality `pred-ESO-HORN` = `normal-pred-ESO-HORN` of Lemma 1 is proved.

4.2. Normalizing predecessor Horn logic with diagonal input-output

The purpose of this section is the normalization of the predecessor logic with diagonal input-output (second part of Lemma 1). It consists in transforming each formula $\Phi \in$ `pred-dio-ESO-HORN` into a formula where each clause is of one of the following forms:

- *input clause* of the form, for $s \in \Sigma$ and $R \in \mathbf{R}$: $x = y \wedge \mathtt{min}(x) \wedge Q_s(x) \to R(x,y)$, or $x = y \wedge \neg\mathtt{min}(x) \wedge Q_s(x) \to R(x,y)$;
- the *contradiction clause*, for a fixed $R_\perp \in \mathbf{R}$: $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge R_\perp(x,y) \to \perp$;
- *computation clause* of the form $\delta_1 \wedge \ldots \wedge \delta_r \to R(x,y)$, for $R \in \mathbf{R}$, where each hypothesis δ_i is a conjunction of the form $S(x-1,y) \wedge \neg\mathtt{min}(x)$ or $S(x,y-1) \wedge \neg\mathtt{min}(y)$, for $S \in \mathbf{R}$.

To this end, we adapt the steps of the previous normalization.

Step 1 is not modified. In Step 2, the initialization clauses are now $x = y \wedge Q_s(x) \to W_s^x(x,y)$ and $x = y \wedge Q_s(y) \to W_s^y(x,y)$ whereas the transport clauses are not modified.

Steps 3 to 5 and the recapitulation after Step 5 are not modified either, except that now each initialization clause is of one of the three forms:

1) $x = y \wedge Q_s(x) \to R(x,y)$;

2) $\mathtt{min}(x) \wedge \eta(y) \to R(x,y)$, with $\eta(y) \in \{(\neg)\mathtt{min}(y), (\neg)\mathtt{max}(y)\}$;

3) $\mathtt{min}(y) \wedge \eta(x) \to R(x,y)$, with $\eta(x) \in \{(\neg)\mathtt{min}(x), (\neg)\mathtt{max}(x)\}$.

Steps 6 and 7 (folding the domain) and the recapitulation after Step 7 are not modified either, except that now each initialization clause has one of the only forms 1 and 2 above.

Step 8 (deleting `max` in the initialization clauses) can be easily adapted according to those initialization clauses whose forms after Step 8 are now restricted to

1) $x = y \wedge Q_s(x) \to R(x,y)$;

2) $\mathtt{min}(x) \wedge \mathtt{min}(y) \to R(x,y)$;

3) $\mathtt{min}(x) \wedge \neg\mathtt{min}(y) \to R(x,y)$.

Clause 2 can be replaced by the equivalent clause 2') $x = y \wedge \mathtt{min}(x) \to R(x,y)$.

Step 9 (from initialization clauses to input clauses) is modified as follows. Define the predicates $R^{\mathtt{min}(x)}$, $R^{\mathtt{min}(y)}$ and $R^{\neg\mathtt{min}(y)}$ by the initialization clauses

4) $x = y \wedge \mathtt{min}(x) \to R^{\mathtt{min}(x)}(x,y)$ and

5) $x = y \wedge \mathtt{min}(x) \to R^{\mathtt{min}(y)}(x,y)$,

and the computation clauses $\neg\mathtt{min}(y) \wedge R^{\mathtt{min}(x)}(x,y-1) \to R^{\mathtt{min}(x)}(x,y)$, $\neg\mathtt{min}(y) \wedge R^{\mathtt{min}(y)}(x,y-1) \to R^{\neg\mathtt{min}(y)}(x,y)$, and $\neg\mathtt{min}(y) \wedge R^{\neg\mathtt{min}(y)}(x,y-1) \to R^{\neg\mathtt{min}(y)}(x,y)$. This allows to replace the initialization clause 3 by the computation clause $R^{\mathtt{min}(x)}(x,y) \wedge R^{\neg\mathtt{min}(y)}(x,y) \to R(x,y)$. After those transformations all the initialization clauses are of the form $x = y \wedge Q_s(x) \to R(x,y)$ (clause 1 above) or $x = y \wedge \mathtt{min}(x) \to R(x,y)$ (clauses 2', 4 and 5 above). By a case analysis, it is easy to transform each of these clauses into an equivalent conjunction of *input clauses* of the required (normalized) forms: $x = y \wedge \mathtt{min}(x) \wedge Q_s(x) \to R(x,y)$, or $x = y \wedge \neg\mathtt{min}(x) \wedge Q_s(x) \to R(x,y)$.

Step 10 (Elimination of atoms $R(x,y)$ as hypotheses) and the end of the proof are the same as those for `pred-ESO-HORN`.

This achieves the proof of the equality `pred-dio-ESO-HORN` = `normal-pred-dio-ESO-HORN` and of Lemma 1. □

4.3. Normalizing inclusion Horn logic

Now, we present the normalization of inclusion logic (Lemma 2), i.e., we show how to transform each formula $\Phi \in$ `incl-ESO-HORN` into a formula of `normal-incl-ESO-HORN` where each clause is of one of the following forms:

- *input clause* of the form $x = y \wedge Q_s(x) \to R(x,y)$, for $s \in \Sigma$ and $R \in \mathbf{R}$;
- the *contradiction clause*, for a fixed $R_\perp \in \mathbf{R}$, $\mathtt{min}(x) \wedge \mathtt{max}(y) \wedge R_\perp(x,y) \to \perp$;
- *computation clause* of the form $x < y \wedge \delta_1 \wedge \ldots \wedge \delta_r \to R(x,y)$, where $R \in \mathbf{R}$ and where each hypothesis δ_i is a computation atom of either form $S(x+1,y)$ or $S(x,y-1)$, for $S \in \mathbf{R}$.

It divides into seven steps.

1. Processing the contradiction clauses. Here again, we delay the contradiction and propagate the predicate $R_\perp$ till the output site $(1,n)$ by the conjunction of the computation clauses $x < y \wedge R_\perp(x+1,y) \to R_\perp(x,y)$ and $x < y \wedge R_\perp(x,y-1) \to R_\perp(x,y)$ with the unique *contradiction clause* $\mathtt{min}(x) \wedge \mathtt{max}(y) \wedge R_\perp(x,y) \to \perp$.

2. Processing the input. We make available the letters of the input word only on the diagonal $x = y$ and in the forms $Q_s(x - a)$ and $Q_s(y + b)$ only (for $a, b \geq 0$), by introducing new computation predicates W_s^{x+a} and W_s^{y+a}, for $a \in \mathbb{Z}$, inductively defined and whose intuitive meaning is: $W_s^{x+a}(x, y) \iff Q_s(x + a) \wedge 1 \leq x + a \leq n$ (resp. $W_s^{y+a}(x, y) \iff Q_s(y + a) \wedge 1 \leq y + a \leq n$). They are inductively defined by the following clauses:

Initialization clauses (on the diagonal): for $s \in \Sigma$ and $a \geq 1$,
$x = y \wedge Q_s(x) \to W_s^x(x, y)$; $x = y \wedge Q_s(x) \to W_s^y(x, y)$;
$x = y \wedge Q_s(x - a) \wedge x > a \to W_s^{x-a}(x, y)$, $x = y \wedge Q_s(x - a) \wedge x > a \to W_s^{y-a}(x, y)$,
$x = y \wedge Q_s(y + a) \wedge y \leq n - a \to W_s^{x+a}(x, y)$, $x = y \wedge Q_s(y + a) \wedge y \leq n - a \to W_s^{y+a}(x, y)$;

Transport clauses: for $a \in \mathbb{Z}$,
$x < y \wedge W_s^{x+a}(x, y - 1) \to W_s^{x+a}(x, y)$, and $x < y \wedge W_s^{y+a}(x + 1, y) \to W_s^{y+a}(x, y)$.

This allows to replace each input atom $Q_s(x + a)$ (resp. $Q_s(y + a)$), $a \in \mathbb{Z}$, by the computation atom $W_s^{x+a}(x, y)$ (resp. $W_s^{y+a}(x, y)$), in all the clauses, except in the initialization clauses.

Note that after Step 2 the atoms on input predicates Q_s occur always jointly with $x = y$ and in the only three forms $Q_s(x)$, $Q_s(x - a)$ (jointly with $x > a$), or $Q_s(y + a)$ (jointly with $y \leq n - a$), for $a \geq 1$ (see the initialization clauses above).

3. Processing the `min`/`max` *literals.* One may consider that the only literals on x involving `min` or `max` are of the forms $x = a$, $x > a$, for an integer $a \geq 1$, or $x = n - a$, $x < n - a$, for $a \geq 0$, and similarly for y. By a similar process as we have done for the Q_s, we can manage to make available the information about `min` and `max`, i.e., about extrema, on the diagonal $x = y$ only and in one of the following forms only: $x = a$ or $x > a$, for some $a \geq 1$, or $y = n - b$ or $y < n - b$, for some $b \geq 0$. We introduce for that new computation predicates defined inductively: $R^{x=a}$, $R^{x>a}$, for $a \geq 1$, and $R^{x=n-a}$, $R^{x<n-a}$, for $a \geq 0$, and similarly for y, with obvious intuitive meaning: for instance, $R^{x>a}(x, y) \iff x > a$. For example, define the predicate $R^{x=a}$ (resp. $R^{y=a}$) by the two clauses $x = y \wedge x = a \to R^{x=a}(x, y)$ and $x < y \wedge R^{x=a}(x, y - 1) \to R^{x=a}(x, y)$ (resp. $x = y \wedge x = a \to R^{y=a}(x, y)$ and $x < y \wedge R^{y=a}(x + 1, y) \to R^{y=a}(x, y)$). As another example, define $R^{x<n-a}$ by the clauses $x = y \wedge y < n - a \to R^{x<n-a}(x, y)$ and $x < y \wedge R^{x<n-a}(x, y - 1) \to R^{x<n-a}(x, y)$.

This allows to replace the "extremum" atoms $x = a$, $x > a$, $x = n-a$, $x < n-a$ by the respective computation atoms $R^{x=a}(x, y)$, $R^{x>a}(x, y)$, $R^{x=n-a}(x, y)$, $R^{x<n-a}(x, y)$, in all the clauses, except in the initialization clauses and in the contradiction clause. And similarly for y. The important fact is that after Step 3, the predicate `min` (resp. `max`) only occurs in the form $x = a$ or $x > a$ (resp. in the form $y = n - a$ or $y < n - a$) and *always occurs jointly with* $x = y$, i.e., is only used on the diagonal.

4. Restriction of computation atoms to $R(x + 1, y)$, $R(x, y - 1)$, *and* $R(x, y)$. This is a variant of the similar step in the normalization of predecessor logics (Step 3). We introduce new "shift" predicates R^{x+a}, R^{y-b} and $R^{x+a,y-b}$, for fixed integers $a, b > 0$ and $R \in \mathbf{R}$, with easy interpretation and definitions. In particular, the intuitive interpretation of the predicate $R^{x+a,y-b}$ is: $R^{x+a,y-b}(x, y) \iff x + a \leq y - b \wedge R(x + a, y - b)$. As an example, the "normalized" clause $x < y \wedge S^{x+2,y-2}(x + 1, y) \to S^{x+3,y-2}(x, y)$ defines the predicate $S^{x+3,y-2}$ from the predicate $S^{x+2,y-2}$.

Recapitulation. After Step 4, one may consider that each clause is of one of the following forms (1-3):

1. an *initialization clause* $x = y \wedge \delta \to R(x, y)$, where δ is
   - either an input atom $Q_s(x)$,
   - or an equality $x = a$, for a fixed $a \geq 1$, or $y = n - b$, for a fixed $b \geq 0$,
   - or a conjunction $Q_s(x - a) \wedge x > a$, or $Q_s(y + b) \wedge y \leq n - b$, for $a, b \geq 1$;
2. a *computation clause* of one of the forms (a,b,c):
   (a) $x < y \wedge S(x + 1, y) \to R(x, y)$;
   (b) $x < y \wedge S(x, y - 1) \to R(x, y)$;

(c) $x \preceq y \wedge S(x,y) \wedge T(x,y) \rightarrow R(x,y)$, where $\preceq \in \{<,=\}$;

3. "the" *contradiction clause* $\texttt{min}(x) \wedge \texttt{max}(y) \wedge R_\perp(x,y) \rightarrow \perp$, which can be rephrased $x = 1 \wedge y = n \wedge R_\perp(x,y) \rightarrow \perp$.

Justification for initialization clauses: By a case analysis, one easily obtains the above three forms of initialization clauses.

Justification for computation clauses: Here again, "break down" each computation clause into clauses of forms (a,b,c) above by introducing new intermediate predicates. For example, the computation clause $x < y \wedge R_1(x+1,y) \wedge R_2(x,y-1) \rightarrow R_3(x,y)$ is "equivalent" to the conjunction of the following clauses that use the new predicates R_4, R_5: $x < y \wedge R_1(x+1,y) \rightarrow R_4(x,y)$; $x < y \wedge R_2(x,y-1) \rightarrow R_5(x,y)$; $x < y \wedge R_4(x,y) \wedge R_5(x,y) \rightarrow R_3(x,y)$.

Steps 5 and 6 that follow rest on a generalization of the method used in Step 8 of the normalization of predecessor logics above (eliminating `max` in the initialization clauses). Roughly expressed, for any computation predicate $R \in \mathbf{R}$ and any hypothesis η, we introduce a new predicate $R^\eta_\leftarrow$ whose intuitive meaning is:

$$R^\eta_\leftarrow(x,y) \iff (\eta \rightarrow R(x,y)).$$

5. Elimination of equalities $x = a$ ($a \geq 1$) and $y = n - b$ ($b \geq 0$), in the initialization clauses. Let A (resp. B) be the maximum of the integers a (resp. b) that occur in the equalities $x = a$ (resp. $y = n - b$) of the clauses. For each $R \in \mathbf{R}$, we introduce the new predicates $R^{x=a}_\leftarrow$, $R^{y=n-b}_\leftarrow$ and $R^{x=a,y=n-b}_\leftarrow$, for all $a \in [1,A]$ and $b \in [0,B]$, whose intuitive meaning has been announced. For example, we should have $R^{x=a,y=n-b}_\leftarrow(x,y) \iff (x = a \wedge y = n - b \rightarrow R(x,y))$.

Transforming the initialization clauses: Each initialization clause $x = y \wedge x = a \rightarrow R(x,y)$ (resp. $x = y \wedge y = n - b \rightarrow R(x,y)$) is transformed into the clause $x = y \rightarrow R^{x=a}_\leftarrow(x,y)$ (resp. $x = y \rightarrow R^{y=n-b}_\leftarrow(x,y)$).

Transforming the computation clauses: For each clause (a) $x < y \wedge S(x+1,y) \rightarrow R(x,y)$, add the clauses $x < y \wedge S^{x=a,y=n-b}_\leftarrow(x+1,y) \rightarrow R^{x=a-1,y=n-b}_\leftarrow(x,y)$, for all $a \in [2,A]$ and $b \in [0,B]$ (*justification*: the hypothesis $x+1 = a$ is equivalent to $x = a-1$). Similarly, for each clause (b) $x < y \wedge S(x,y-1) \rightarrow R(x,y)$, add the clauses $x < y \wedge S^{x=a,y=n-b}_\leftarrow(x,y-1) \rightarrow R^{x=a,y=n-(b-1)}_\leftarrow(x,y)$, for all $a \in [1,A]$ and $b \in [1,B]$. Also add for clauses (a,b) the similar (simpler) clauses with only one (instead of two) equality hypothesis. For example, for clause (a) we add the clauses $x < y \wedge S^{x=a}_\leftarrow(x+1,y) \rightarrow R^{x=a-1}_\leftarrow(x,y)$, for all $a \in [2,A]$, and $x < y \wedge S^{y=n-b}_\leftarrow(x+1,y) \rightarrow R^{y=n-b}_\leftarrow(x,y)$, for all $b \in [0,B]$.

For each clause (c) $x \preceq y \wedge S(x,y) \wedge T(x,y) \rightarrow R(x,y)$, where $\preceq \in \{<,=\}$, add clauses that *cumulate* the hypotheses as long as they are *compatible*. More precisely, for all $a \in [1,A]$ and $b \in [0,B]$ and any two compatible (possibly empty) subsets η, θ of the set of two hypotheses $\{x = a, y = n - b\}$, we have the clause $x \preceq y \wedge S^\eta_\leftarrow(x,y) \wedge T^\theta_\leftarrow(x,y) \rightarrow R^{\eta \cup \theta}_\leftarrow(x,y)$. For example, $x \preceq y \wedge S^{x=a}_\leftarrow(x,y) \wedge T^{y=n-b}_\leftarrow(x,y) \rightarrow R^{x=a,y=n-b}_\leftarrow(x,y)$ and $x \preceq y \wedge S^{y=n-b}_\leftarrow(x,y) \wedge T^{x=a,y=n-b}_\leftarrow(x,y) \rightarrow R^{x=a,y=n-b}_\leftarrow(x,y)$.

Processing the contradiction clause: The contradiction clause is equivalent to $x = 1 \wedge y = n \wedge (x = 1 \wedge y = n \rightarrow R_\perp(x,y)) \rightarrow \perp$. Consequently, it should be replaced by the clause $x = 1 \wedge y = n \wedge (R_\perp)^{x=1,y=n}_\leftarrow(x,y) \rightarrow \perp$, which is the contradiction clause required if the predicate $(R_\perp)^{x=1,y=n}_\leftarrow$ is renamed $R_\perp$.

6. Elimination of atoms $Q_s(x-a)$, $Q_s(y+b)$, for $a, b > 0$. This step is quite similar to Step 5. For each $R \in \mathbf{R}$, we introduce new predicates:

$$R^{x-a_1,\ldots,x-a_l,y+b_1,\ldots,y+b_m}_{\leftarrow s_1,\ldots,s_l,t_1,\ldots,t_m}$$

with $l, m \geq 0$, the $s_i, t_j \in \Sigma$, $0 \leq a_1 < a_2 \ldots < a_l \leq A$ and $0 \leq b_1 < b_2 \ldots < b_m \leq B$, where A (resp. B) is the maximal a in atoms $Q_s(x-a)$ (resp. maximal b in atoms $Q_s(y+b)$). Their intuitive meaning is as follows:

$R^{x-a_1,\ldots,x-a_l,y+b_1,\ldots,y+b_m}_{\leftarrow s_1,\ldots,s_l,t_1,\ldots,t_m}(x,y) \iff$

$[\,[\bigwedge_{i=1,\ldots,l}(Q_{s_i}(x-a_i) \wedge x > a_i) \wedge \bigwedge_{j=1,\ldots,m}(Q_{t_j}(y+b_j) \wedge y \leq n - b_j)] \rightarrow R(x,y)\,]$.

Transforming the initialization clauses: Each initialization clause $x = y \wedge Q_s(x-a) \wedge x > a \rightarrow R(x,y)$, $a \geq 1$, (resp. $x = y \wedge Q_s(y+b) \wedge y \leq n-b \rightarrow R(x,y)$, $b \geq 1$) is transformed into the clause $x = y \wedge R^{x-a}_{\leftarrow,s}(x,y) \rightarrow R(x,y)$ (resp. $x = y \wedge R^{y+b}_{\leftarrow,s}(x,y) \rightarrow R(x,y)$).

Transforming the computation clauses: To each clause (a) $x < y \wedge S(x+1,y) \rightarrow R(x,y)$ add the following clauses justified by the identity $x+1-a_i = x-(a_i-1)$:

$$x < y \wedge S^{x-a_1,\ldots,x-a_l,y+b_1,\ldots,y+b_m}_{\leftarrow s_1,\ldots,s_l,t_1,\ldots,t_m}(x+1,y) \rightarrow R^{x-(a_1-1),\ldots,x-(a_l-1),y+b_1,\ldots,y+b_m}_{\leftarrow s_1,\ldots,s_l,t_1,\ldots,t_m}(x,y).$$

Similarly, to each clause (b) $x < y \wedge S(x,y-1) \rightarrow R(x,y)$ add the clauses

$$x < y \wedge S^{x-a_1,\ldots,x-a_l,y+b_1,\ldots,y+b_m}_{\leftarrow s_1,\ldots,s_l,t_1,\ldots,t_m}(x,y-1) \rightarrow R^{x-a_1,\ldots,x-a_l,y+(b_1-1),\ldots,y+(b_m-1)}_{\leftarrow s_1,\ldots,s_l,t_1,\ldots,t_m}(x,y).$$

Moreover, add for $a_1 = 0$ and each $s_1 \in \Sigma$, the following "verification" clauses, which intuitively delete the hypothesis $Q_{s_1}(x)$ after verifying that it is satisfied because of the equivalence $W^x_{s_1}(x,y) \iff Q_{s_1}(x)$:

$$x < y \wedge S^{x,x-a_2,\ldots,x-a_l,y+b_1,\ldots,y+b_m}_{\leftarrow s_1,s_2,\ldots,s_l,t_1,\ldots,t_m}(x,y) \wedge W^x_{s_1}(x,y) \rightarrow R^{x-a_2,\ldots,x-a_l,y+b_1,\ldots,y+b_m}_{\leftarrow s_2,\ldots,s_l,t_1,\ldots,t_m}(x,y).$$

Similarly, add for $b_1 = 0$ and each $t_1 \in \Sigma$, the "verification" clauses (justified by $W^y_{t_1}(x,y) \iff Q_{t_1}(y)$):

$$x < y \wedge S^{x-a_1,\ldots,x-a_l,y,y+b_2,\ldots,y+b_m}_{\leftarrow s_1,\ldots,s_l,t_1,t_2,\ldots,t_m}(x,y) \wedge W^y_{t_1}(x,y) \rightarrow R^{x-a_1,\ldots,x-a_l,y+b_2,\ldots,y+b_m}_{\leftarrow s_1,\ldots,s_l,t_2\ldots,t_m}(x,y).$$

For each clause (c) $x \preceq y \wedge S(x,y) \wedge T(x,y) \rightarrow R(x,y)$, where $\preceq \in \{<,=\}$, add similar clauses that *cumulate* the hypotheses provided they are *compatible*: for example, the clause

$$x \preceq y \wedge S^{x-1,x-3,y+2}_{s_1,s_2,t_1}(x,y) \wedge T^{x-1,y+2,y+4}_{s_1,t_1,t_2}(x,y) \rightarrow R^{x-1,x-3,y+2,y+4}_{s_1,s_2,t_1,t_2}(x,y).$$

Recapitulation. After Step 6, all the initialization clauses are of the form $x = y \wedge Q_s(x) \rightarrow R(x,y)$ as required[10].

7. Elimination of atoms $R(x,y)$ as hypotheses. This step is absolutely similar to the corresponding Step 10 of normalization of predecessor logics.

This completes the proof of the equality `incl-ESO-HORN` = `normal-incl-ESO-HORN`, i.e., Lemma 2. □

5. Extending our logics with negation and normalizing them

In this section, we will extend our logics by allowing a limited use of negation. The main interest of this extension is to make easier the natural expression of problems within the logics without changing the complexity classes involved. More precisely, we now allow the negations of computation atoms $\neg\alpha$ as hypotheses of the clauses provided α is not in the form $R(x,y)$ nor $R(y,x)$. We will see that such an extension does not increase the computation power (complexity) of the logics because it does not modify the nature of their inductive process.

Let us give the precise definitions of our formulas with negation called *inductive formulas* since they are no longer Horn formulas.

Definition 7 (predecessor logics)**.** *A* predecessor inductive formula *(resp.* predecessor inductive formula with diagonal input-output*) is a formula of the form $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x,y)$ where $\mathbf{R}$ is a set of binary predicates called* computation predicates *and ψ is a conjunction of clauses on the variables x, y, of signature $\mathcal{S}_\Sigma \cup \mathbf{R}$ (resp. $\mathcal{S}_\Sigma \cup \mathbf{R} \cup \{=\}$), of the form $\delta_1 \wedge \ldots \wedge \delta_r \rightarrow \delta_0$ where the conclusion δ_0 is either a* computation atom $R(x,y)$ *with $R \in \mathbf{R}$, or $\bot$, and each hypothesis δ_i is*

1. *either an* input literal *(resp.* input conjunction*) of one of the forms:*

[10] Note that an initialization clause of the form $x = y \rightarrow R(x,y)$ can be rewritten $\bigwedge_{s\in\Sigma}(x = y \wedge Q_s(x) \rightarrow R(x,y))$ (case analysis).

- $Q_s(x-a)$, $Q_s(y-a)$ *(resp.* $Q_s(x-a) \wedge x = y$*), for* $s \in \Sigma$ *and* $a \geq 0$,
- $(\neg)U(x-a)$ *or* $(\neg)U(y-a)$*, for* $U \in \{\texttt{min}, \texttt{max}\}$ *and* $a \geq 0$,

2. *or a* computation literal *of the form* $S(x,y)$, $S(y,x)$, $(\neg)S(x-a, y-b)$ *or* $(\neg)S(y-b, x-a)$*, for* $S \in \mathbf{R}$ *and* $a, b \geq 0$ *such that* $a + b > 0$.

Let `pred-ESO-IND` (resp. `pred-dio-ESO-IND`) denote the class of *predecessor inductive formulas* (resp. *predecessor inductive formulas with diagonal input-output*) and, also, the class of languages they define.

Definition 8 (inclusion logic)**.** *An* inclusion inductive formula *is a formula of the form* $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x,y)$ *where* $\mathbf{R}$ *is a set of binary predicates called* computation predicates *and* ψ *is a conjunction of clauses of signature* $\mathcal{S}_\Sigma \cup \mathbf{R} \cup \{=, \leq, <\}$*, of the form* $x \leq y \wedge \delta_1 \wedge \ldots \wedge \delta_r \rightarrow \delta_0$ *where the conclusion* δ_0 *is either a* computation atom $R(x,y)$ *with* $R \in \mathbf{R}$*, or* $\bot$*, and each hypothesis* δ_i *is*

1. *either an* input literal *of the form* $(\neg)U(x+a)$ *or* $(\neg)U(y+a)$*, for* $a \in \mathbb{Z}$ *and* $U \in \{(Q_s)_{s\in\Sigma}, \texttt{min}, \texttt{max}\}$,
2. *or an (in)equality* $x = y$ *or* $x < y$,
3. *or a conjunction of the form* $S(x,y) \wedge x \leq y$ *or* $(\neg)S(x+a, y-b) \wedge x + a \leq y - b$*, for* $S \in \mathbf{R}$ *and* $a, b \geq 0$ *such that* $a + b > 0$.

Let `incl-ESO-IND` denote the class of *inclusion inductive formulas* and, also, the class of languages they define. Here again, it is convenient to normalize the logics.

Normalizing logics with negation

Lemma 4 (normalization of predecessor logics)**.** *Each formula* $\Phi \in$ `pred-ESO-IND` *(resp.* $\Phi \in$ `pred-dio-ESO-IND`*) is equivalent to a formula* $\Phi' \in$ `pred-ESO-IND` *(resp.* $\Phi' \in$ `pred-dio-ESO-IND`*) where each clause is of one of the following forms:*

- *an* input clause *of the form* $\texttt{min}(x) \wedge \texttt{min}(y) \wedge Q_s(y) \rightarrow R(x,y)$, *or* $\texttt{min}(x) \wedge \neg\texttt{min}(y) \wedge Q_s(y) \rightarrow R(x,y)$ *(resp.* $x = y \wedge \texttt{min}(x) \wedge Q_s(x) \rightarrow R(x,y)$, *or* $x = y \wedge \neg\texttt{min}(x) \wedge Q_s(x) \rightarrow R(x,y)$*), for* $s \in \Sigma$ *and* $R \in \mathbf{R}$*;*
- *the* contradiction clause $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\bot(x,y) \rightarrow \bot$*, for a fixed* $R_\bot \in \mathbf{R}$*;*
- *a* computation clause *of the form* $\delta_1 \wedge \ldots \wedge \delta_r \rightarrow R(x,y)$*, for* $R \in \mathbf{R}$*, where each hypothesis* δ_i *is a conjunction of the form* $(\neg)S(x-1,y) \wedge \neg\texttt{min}(x)$ *or* $(\neg)S(x,y-1) \wedge \neg\texttt{min}(y)$*, for* $S \in \mathbf{R}$.

Let `normal-pred-ESO-IND` (resp. `normal-pred-dio-ESO-IND`) denote the class of formulas (languages) so defined.

Lemma 5 (normalization of inclusion logic)**.** *Each formula* $\Phi \in$ `incl-ESO-IND` *is equivalent to a formula* $\Phi' \in$ `incl-ESO-IND` *where each clause is of one of the following forms:*

- *an* input clause *of the form* $x = y \wedge Q_s(x) \rightarrow R(x,y)$*, for* $s \in \Sigma$ *and* $R \in \mathbf{R}$*;*
- *the* contradiction clause*, for a fixed* $R_\bot \in \mathbf{R}$*,* $\texttt{min}(x) \wedge \texttt{max}(y) \wedge R_\bot(x,y) \rightarrow \bot$*;*
- *a* computation clause *of the form* $x < y \wedge \delta_1 \wedge \ldots \wedge \delta_r \rightarrow R(x,y)$*, where* $R \in \mathbf{R}$ *and each hypothesis* δ_i *is a computation atom of either forms* $(\neg)S(x+1,y)$ *or* $(\neg)S(x,y-1)$*, for* $S \in \mathbf{R}$.

Let `normal-incl-ESO-IND` denote the class of formulas (languages) so defined.

5.1. Proof of Lemma 4: Normalizing predecessor logics with negation

The transformation still divides into the same 10 steps of the normalization of predecessor Horn logics (proof of Lemma 1). Steps 1-2, 4-6 and 9-10 are exactly the same or can be trivially adapted. Let us explain how to adapt Steps 3, 7 and 8 for formulas in `pred-ESO-HORN` (the case `pred-dio-ESO-HORN` is similar).

Step 3: Restriction of the computation literals to $(\neg)R(x-1,y)$, $(\neg)R(x,y-1)$ *and* $R(x,y)$. In addition to the "shift" predicates R^{x-a}, R^{y-b}, and $R^{x-a,y-b}$, we define "shift negative" computation predicates of the forms $R^{\neg,x-a}$, $R^{\neg,y-b}$, $R^{\neg,x-a,y-b}$, for fixed integers $a,b>0$, with intuitive meaning: $R^{\neg,x-a}(x,y) \iff \neg R(x-a,y)$; $R^{\neg,y-b}(x,y) \iff \neg R(x,y-b)$; $R^{\neg,x-a,y-b}(x,y) \iff \neg R(x-a,y-b)$. Let us describe the transformation on an example. The computation clause

$$y>2 \wedge x>3 \wedge S(x,y-2) \wedge \neg T(x-3,y-1) \rightarrow R(x,y) \tag{1}$$

is transformed into the clause without negation

$$y>2 \wedge x>3 \wedge S^{y-1}(x,y-1) \wedge T^{\neg,x-2,y-1}(x-1,y) \rightarrow R(x,y). \tag{2}$$

The predicate S^{y-1} is defined by the implication $y>1 \wedge S(x,y-1) \rightarrow S^{y-1}(x,y)$ which implies

$$y>2 \wedge S(x,y-2) \rightarrow S^{y-1}(x,y-1). \tag{3}$$

The predicates $T^{\neg,x-1}$, $T^{\neg,x-2}$ and $T^{\neg,x-2,y-1}$ are defined successively by the implications $x>1 \wedge \neg T(x-1,y) \rightarrow T^{\neg,x-1}(x,y)$, $x>2 \wedge T^{\neg,x-1}(x-1,y) \rightarrow T^{\neg,x-2}(x,y)$, and $x>2 \wedge y>1 \wedge T^{\neg,x-2}(x,y-1) \rightarrow T^{\neg,x-2,y-1}(x,y)$, which imply by transitivity $x>2 \wedge y>1 \wedge \neg T(x-2,y-1) \rightarrow T^{\neg,x-2,y-1}(x,y)$ and then

$$x>3 \wedge y>1 \wedge \neg T(x-3,y-1) \rightarrow T^{\neg,x-2,y-1}(x-1,y). \tag{4}$$

Justification: Since the clauses defining the "shift" predicates S^{y-1}, $T^{\neg,x-1}$, $T^{\neg,x-2}$ and $T^{\neg,x-2,y-1}$ imply the clauses 3 and 4, and since the conjunction of the clauses 3, 4 and 2 implies clause 1 by transitivity, the conjunction of clause 2 with the clauses defining the "shift" predicates also implies clause 1 as required. Besides, one observes that after the transformation, all the clauses obtained, which in our example are clause 2 and the clauses defining the "shift" predicates, are of the forms required $(\neg)R(x-1,y)$, $(\neg)R(x,y-1)$, and $R(x,y)$.

Moreover, note that after Step 3 the only clauses involving negation are of the forms $x>1 \wedge \neg S(x-1,y) \rightarrow R(x,y)$ or $y>1 \wedge \neg S(x,y-1) \rightarrow R(x,y)$.

Recapitulation. We can *assume* that after Steps 1-6 each clause is of one of the following forms:

1. an *initialization clause* of one of the two forms:
   $\mathtt{min}(x) \wedge \eta(y) \rightarrow R(x,y)$ with $\eta(y) \in \{(Q_s(y))_{s\in\Sigma}, (\neg)\mathtt{min}(y), (\neg)\mathtt{max}(y)\}$;
   $\mathtt{min}(y) \wedge \eta(x) \rightarrow R(x,y)$ with $\eta(x) \in \{(Q_s(x))_{s\in\Sigma}(\neg)\mathtt{min}(x), (\neg)\mathtt{max}(x)\}$;
2. "the" *contradiction clause* $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge R_\perp(x,y) \rightarrow \perp$;
3. a *computation clause* of one of the following forms:
   (a) $(\neg)S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R(x,y)$;
   (b) $(\neg)S(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$;
   (c) $S(x,y) \wedge T(x,y) \rightarrow R(x,y)$.

Step 7: Folding the domain. Recall briefly this transformation by adding the slight extensions due to the negation of hypotheses.

Folding the initialization clauses: Each clause of the form $\mathtt{min}(x) \wedge \eta(y) \rightarrow R(x,y)$ (with $\eta(y) \in \{Q_s(y) | s \in \Sigma\} \cup \{(\neg)\mathtt{min}(y), (\neg)\mathtt{max}(y)\}$) is *unchanged* ; each clause of the form $\mathtt{min}(y) \wedge \eta(x) \rightarrow R(x,y)$ (with $\eta(x) \in \{Q_s(x) | s \in \Sigma\} \cup \{(\neg)\mathtt{min}(x), (\neg)\mathtt{max}(x)\}$) is *replaced* by its *folded* version $\mathtt{min}(x) \wedge \eta(y) \rightarrow R^{\mathtt{inv}}(x,y)$.

Folding the computation clauses:
- Clause (a) $(\neg)S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R(x,y)$ is replaced by the equivalent conjunction of the clause $x \leq y \wedge (\neg)S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R(x,y)$ with the folded clause $x<y \wedge (\neg)S^{\mathtt{inv}}(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R^{\mathtt{inv}}(x,y)$.
- Clause (b) $(\neg)S(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$ is replaced by the equivalent conjunction of the clause $x<y \wedge (\neg)S(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$ with the folded clause $x \leq y \wedge (\neg)S^{\mathtt{inv}}(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R^{\mathtt{inv}}(x,y)$.

Folding the contradiction clause: The clause $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge R_\perp(x,y) \rightarrow \perp$ is not modified.

The diagonal fold: Add for each $R \in \mathbf{R}$ the clauses $x=y \wedge R(x,y) \rightarrow R^{\mathtt{inv}}(x,y)$ and $x=y \wedge R^{\mathtt{inv}}(x,y) \rightarrow R(x,y)$.

Recapitulation. After Steps 1-7, each clause is of one of the following forms:

1. an *initialization clause* of the form $\mathtt{min}(x) \wedge \eta(y) \rightarrow R(x,y)$ with $\eta(y) \in \{Q_s(y) | s \in \Sigma\} \cup \{(\neg)\mathtt{min}(y), (\neg)\mathtt{max}(y)\}$;
2. "the" *contradiction clause* $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge R_\perp(x,y) \rightarrow \perp$;
3. a *computation clause* of one of the following forms:
   (a) $x \leq y \wedge (\neg)S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R(x,y)$;
   (b) $x < y \wedge (\neg)S(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$;
   (c) $x \leq y \wedge S(x,y) \wedge T(x,y) \rightarrow R(x,y)$;
   (d) $x = y \wedge S(x,y) \rightarrow R(x,y)$.

Step 8: Deleting max *in the initialization clauses.* It is sufficient to describe the transformation of the clauses that *contain a negation* (the transformation of the other clauses is not modified). First, note that any clause of the form

$$\text{(a) } x \leq y \wedge \neg S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R(x,y)$$

is equivalent to the conjunction of the two following clauses:

- $x \leq y \wedge \mathtt{max}(y) \wedge \neg S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow (\mathtt{max}(y) \rightarrow R(x,y))$, and
- $x \leq y \wedge \neg\mathtt{max}(y) \wedge \neg S(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow (\neg\mathtt{max}(y) \rightarrow R(x,y))$,

which are equivalent to

1) $x \leq y \wedge \neg(\mathtt{max}(y) \rightarrow S(x-1,y)) \wedge \neg\mathtt{min}(x) \rightarrow (\mathtt{max}(y) \rightarrow R(x,y))$, and

2) $x \leq y \wedge \neg(\neg\mathtt{max}(y) \rightarrow S(x-1,y)) \wedge \neg\mathtt{min}(x) \rightarrow (\neg\mathtt{max}(y) \rightarrow R(x,y))$, respectively.

Using the equivalence $S^y_{\leftarrow\mathtt{max}}(x,y) \iff (\mathtt{max}(y) \rightarrow S(x,y))$ and the similar equivalences for $R^y_{\leftarrow\mathtt{max}}$, $S^y_{\leftarrow\neg\mathtt{max}}$ and $R^y_{\leftarrow\neg\mathtt{max}}$ we can transform clauses 1 and 2 into the "equivalent" clauses

1') $x \leq y \wedge \neg S^y_{\leftarrow\mathtt{max}}(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R^y_{\leftarrow\mathtt{max}}(x,y)$, and

2') $x \leq y \wedge \neg S^y_{\leftarrow\neg\mathtt{max}}(x-1,y) \wedge \neg\mathtt{min}(x) \rightarrow R^y_{\leftarrow\neg\mathtt{max}}(x,y)$, respectively.

This justifies to replace clause (a) above by the conjunctions of clauses (1') and (2').

Similarly, any clause (b) $x < y \wedge \neg S(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$ is equivalent to $x < y \wedge \neg\mathtt{max}(y-1) \wedge \neg S(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$ and therefore to $x < y \wedge \neg S^y_{\leftarrow\neg\mathtt{max}}(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R(x,y)$. This justifies the replacement of clause (b) by the conjunction of clauses $x < y \wedge \neg S^y_{\leftarrow\neg\mathtt{max}}(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R^y_{\leftarrow\mathtt{max}}(x,y)$ and $x < y \wedge \neg S^y_{\leftarrow\neg\mathtt{max}}(x,y-1) \wedge \neg\mathtt{min}(y) \rightarrow R^y_{\leftarrow\neg\mathtt{max}}(x,y)$.

This concludes the proof of Lemma 4. □

5.2. Proof of Lemma 5: Normalizing inclusion logic with negation

It is a variant of the seven steps of the normalization of inclusion Horn formulas (see the proof of Lemma 2). Steps 1-3 are not modified.

Step 4: Restriction of computation atoms to $(\neg)R(x+1,y)$, $(\neg)R(x,y-1)$, *and* $R(x,y)$. As this is a straightforward variant of the similar step (Step 3) of the normalization of inductive predecessor logics (proof of Lemma 4 above) we let the reader complete the details.

Recapitulation. After Step 4, one may consider that each clause is of one of the following forms (1-3):

1. an *initialization clause* $x = y \wedge \delta \to R(x, y)$, where δ is
   - either an input atom $Q_s(x)$,
   - or an equality $x = a$, for a fixed $a \geq 1$, or $y = n - b$, for a fixed $b \geq 0$,
   - or a conjunction $Q_s(x - a) \wedge x > a$, or $Q_s(y + b) \wedge y \leq n - b$, for $a, b \geq 1$;
2. a *computation clause* of one of the forms (a,b,c):
   (a) $x < y \wedge (\neg)S(x + 1, y) \to R(x, y)$;
   (b) $x < y \wedge (\neg)S(x, y - 1) \to R(x, y)$;
   (c) $x \preceq y \wedge S(x, y) \wedge T(x, y) \to R(x, y)$, where $\preceq \in \{<, =\}$;
3. "the" *contradiction clause* $x = 1 \wedge y = n \wedge R_\perp(x, y) \to \perp$.

Steps 5 and 6 are deeply modified. They use the following notions. Let A (resp. B) be the maximum of integers a (resp. b) that occurs in equalities $x = a$ (resp. $y = n - b$) and conjunctions $Q_s(x-a) \wedge x > a$ (resp. $Q_s(y+b) \wedge y \leq n-b$) of the initialization clauses. We call an *x-description* any conjunction of the form $x = a \wedge \bigwedge_{i=1}^{a-1} Q_{s_i}(x - i)$, for $a \in [1, A]$ and $(s_1, \ldots, s_a) \in \Sigma^a$, or $x > A \wedge \bigwedge_{i=1}^{A} Q_{s_i}(x - i)$, for $(s_1, \ldots, s_A) \in \Sigma^A$. The first form is called an *x-equal-description*. The second form is called an *x-sup-description*. Similarly, a *y-description* is any conjunction of the form $y = n - b \wedge \bigwedge_{j=1}^{b} Q_{t_j}(y + j)$, for $b \in [0, B - 1]$ and $(t_1, \ldots, t_b) \in \Sigma^b$ (*y-equal-description*), or $y \leq n - B \wedge \bigwedge_{j=1}^{B} Q_{t_j}(y + j)$, for $(t_1, \ldots, t_B) \in \Sigma^B$ (*y-inf-description*). Let H denote the set of x-descriptions and Θ the set of y-descriptions. Notice as special cases the x-description $x = 1$ and the y-description $y = n$. Here are the key conditions satisfied by x-descriptions and y-descriptions:

- $x = a$ (resp. $Q_s(x-a) \wedge x > a$), for $a \in [1, A]$, is equivalent to the disjunction of the set $H^{x=a}$ (resp. H_s^{x-a}) of x-descriptions that contain $x = a$ (resp. $Q_s(x - a)$) as a conjunct;
- similarly, $y = n - b$, for $b \in [0, B]$ (resp. $Q_t(y + b) \wedge y \leq n - b$, for $b \in [1, B]$), is equivalent to the disjunction of the set $\Theta^{y=n-b}$ (resp. Θ_t^{y+b}) of y-descriptions that contain $y = n - b$ (resp. $Q_t(y + b)$) as a conjunct;
- the x-descriptions (resp. y-descriptions) cover all possible cases, i.e., the disjunctions $\bigvee_{\eta \in H} \eta$ and $\bigvee_{\theta \in \Theta} \theta$ are tautologies.

Step 5: Processing the initialization clauses.. Replace each initialization clause of the form $x = y \wedge \delta \to R(x, y)$ where δ is $x = a$ (resp. δ is $Q_s(x - a) \wedge x > a$) by the equivalent conjunction of clauses $x = y \wedge \eta \wedge \theta \to R(x, y)$ where $\eta \in H^{x=a}$ (resp. $\eta \in H_s^{x-a}$) and $\theta \in \Theta$. Similarly, replace each initialization clause of the form $x = y \wedge \delta \to R(x, y)$ where δ is $y = n - b$ (resp. δ is $y \leq n - b \wedge Q_t(y + b)$) by the equivalent conjunction of clauses $x = y \wedge \eta \wedge \theta \to R(x, y)$ where $\eta \in H$ and $\theta \in \Theta^{y=n-b}$ (resp. $\theta \in \Theta_t^{y+b}$). Notice that each initialization clause is now of the form $x = y \wedge Q_s(x) \to R(x, y)$ or $x = y \wedge \eta \wedge \theta \to R(x, y)$ where $\eta \in H$ and $\theta \in \Theta$.

Step 6: Processing the clauses with conditional hypotheses. Let us introduce the new computation predicates $R_{\leftarrow}^{\eta,\theta}$, for each $R \in \mathbf{R}$, each x-description η and each y-description θ, with the following intuitive meaning: $R_{\leftarrow}^{\eta,\theta}(x, y) \iff (\eta \wedge \theta \to R(x, y))$. We now transform all the clauses by using the predicates $R_{\leftarrow}^{\eta,\theta}$ instead of the initial computation predicates R.

Transforming the initialization clauses: Replace each clause $x = y \wedge Q_s(x) \to R(x, y)$ by the "equivalent" conjunction of clauses $\bigwedge_{(\eta,\theta) \in H \times \Theta}(x = y \wedge Q_s(x) \to R_{\leftarrow}^{\eta,\theta}(x, y))$, and replace each clause of the form $x = y \wedge \eta \wedge \theta \to R(x, y)$ which is "equivalent" to the clause $x = y \ \to R_{\leftarrow}^{\eta,\theta}(x, y)$ by the conjunction of clauses $\bigwedge_{s \in \Sigma}(x = y \wedge Q_s(x) \to R_{\leftarrow}^{\eta,\theta}(x, y))$.

Transforming the computation clauses: Let us consider successively the clauses of forms (a), (b) (with or without negation) and (c).

`Transforming clauses` $x < y \wedge \neg S(x + 1, y) \to R(x, y)$`:`
The transformation rests on the three following obvious facts:

- $\neg S(x+1,y)$ is equivalent to $\bigvee_{\eta\in H,\theta\in\Theta}(\eta(x+1)\wedge\theta(y)\wedge\neg S(x+1,y))$;
- $\eta(x+1)\wedge\theta(y)\wedge\neg S(x+1,y)$ is equivalent to $\neg(\eta(x+1)\wedge\theta(y)\rightarrow S(x+1,y))$, that is $\neg S^{\eta,\theta}_{\leftarrow}(x+1,y)$;
- $R(x,y)$ is equivalent to $\bigwedge_{\eta\in H,\theta\in\Theta}(\eta(x)\wedge\theta(y)\rightarrow R(x,y))$, that is $\bigwedge_{\eta\in H,\theta\in\Theta} R^{\eta,\theta}_{\leftarrow}(x,y)$.

These facts justify the replacement of the clause $x<y\wedge\neg S(x+1,y)\rightarrow R(x,y)$ by the "equivalent" conjunction of clauses $x<y\wedge\neg S^{\eta,\theta}_{\leftarrow}(x+1,y)\rightarrow R^{\eta',\theta'}_{\leftarrow}(x,y)$, for $\eta,\eta'\in H$ and $\theta,\theta'\in\Theta$.

`Transforming clauses` $x<y\wedge S(x+1,y)\rightarrow R(x,y)$`:`
Let $\eta^{x=a,x-1,\ldots,x-(a-1)}_{s_1,\ldots,s_{a-1}}(x)$ (resp. $\eta^{x>A,x-1,\ldots,x-A}_{s_1,\ldots,s_A}(x)$) denote the *x-description* $x=a\wedge\bigwedge_{i=1}^{a-1}Q_{s_i}(x-i)$, for $a\in[1,A]$ and $(s_1,\ldots,s_{a-1})\in\Sigma^a$ (resp. $x>A\wedge\bigwedge_{i=1}^{A}Q_{s_i}(x-i)$, for $(s_1,\ldots,s_A)\in\Sigma^A$).
The following equivalences which are easy to check[11] will be useful to justify the transformation of clauses (a):

$$\eta^{x=a,x-1,\ldots,x-(a-1)}_{s_1,\ldots,s_{a-1}}(x+1)\iff\Big(Q_{s_1}(x)\wedge\eta^{x=a-1,x-1,\ldots,x-(a-2)}_{s_2,\ldots,s_{a-1}}(x)\Big)\tag{5}$$

$$\eta^{x>A,x-1,\ldots,x-A}_{s_1,\ldots,s_A}(x+1)\iff\bigvee_{s\in\Sigma}\big(Q_{s_1}(x)\wedge\eta^{x>A,x-1,\ldots,x-A}_{s_2,\ldots,s_A,s}(x)\big)\tag{6}$$

Notation. *For any* x-equal-description $\eta:=\eta^{x=a,x-1,\ldots,x-(a-1)}_{s_1,\ldots,s_{a-1}}$*, let* $\eta^{\triangleright}$ *denote the* x-equal-description $\eta^{x=a-1,x-1,\ldots,x-(a-2)}_{s_2,\ldots,s_{a-1}}$ *involved in equivalence 5. For any* x*-sup-description* $\eta:=\eta^{x>A,x-1,\ldots,x-A}_{s_1,\ldots,s_A}$ *and any* $s\in\Sigma$*, let* η^{x-A}_{s} *denote the* x*-sup-description* $\eta^{x>A,x-1,\ldots,x-A}_{s_2,\ldots,s_A,s}$ *involved in equivalence 6.*

Replace any clause (a) $x<y\wedge S(x+1,y)\rightarrow R(x,y)$ by the conjunction of clauses

$$x<y\wedge S^{\eta,\theta}(x+1,y)\wedge W^x_{s_1}(x,y)\rightarrow R^{\eta^{\triangleright},\theta}_{\leftarrow}(x,y)$$

for all $s_1\in\Sigma$, every *x-equal-description* η that contains $Q_{s_1}(x-1)$, and each $\theta\in\Theta$;

$$x<y\wedge S^{\eta,\theta}(x+1,y)\wedge W^x_{s_1}(x,y)\rightarrow R^{\eta^{x-A}_{s},\theta}_{\leftarrow}(x,y)$$

for all $s_1,s\in\Sigma$, every x-sup-description η that contains $Q_{s_1}(x-1)$ and each $\theta\in\Theta$. This replacement is justified by the above-mentioned properties of x-descriptions and y-descriptions, equivalences 5 and 6 and the meaning of $W^x_{s_1}$, i.e., $W^x_{s_1}(x,y)\iff Q_{s_1}(x)$.

`Transforming clauses` $x<y\wedge\neg S(x,y-1)\rightarrow R(x,y)$ `and` $x<y\wedge S(x,y-1)\rightarrow R(x,y)$`:`
These clauses are transformed by the same method as for clauses (a). So, we let the reader complete the details.

`Transforming clauses` $x\preceq y\wedge S(x,y)\wedge T(x,y)\rightarrow R(x,y)$`, where` $\preceq\in\{<,=\}$`:`
Replace each clause (c) $x\preceq y\wedge S(x,y)\wedge T(x,y)\rightarrow R(x,y)$, where $\preceq\in\{<,=\}$, by the set of clauses $x\preceq y\wedge S^{\eta,\theta}_{\leftarrow}(x,y)\wedge T^{\eta,\theta}_{\leftarrow}(x,y)\rightarrow R^{\eta,\theta}_{\leftarrow}(x,y)$, for $(\eta,\theta)\in H\times\Theta$.

Processing the contradiction clause: The initial contradiction clause is equivalent to $x=1\wedge y=n\wedge(x=1\wedge y=n\rightarrow R_{\perp}(x,y))\rightarrow\perp$. So, here again, it should be replaced by the clause $x=1\wedge y=n\wedge(R_{\perp})^{x=1,y=n}_{\leftarrow}(x,y)\rightarrow\perp$, which is the contradiction clause required if the predicate $(R_{\perp})^{x=1,y=n}_{\leftarrow}$ is renamed $R_{\perp}$.

Recapitulation. After Step 6, all the initialization clauses are of the form $x=y\wedge Q_s(x)\rightarrow R(x,y)$, i.e. are the required *input clauses.*

Step 7: Elimination of atoms $R(x,y)$ *as hypotheses.* This step is not modified.

This completes the proof of the equality `incl-ESO-IND` = `normal-incl-ESO-IND`, i.e., Lemma 5. □

[11] We only shift the data from $x+1$ to x using the equivalences: $x+1=a\iff x=a-1$; $Q_s((x+1)-i)\iff Q_s(x-(i-1))$; $x+1>A\iff x>A-1$.

6. Equivalence between our logics and real-time cellular automata

In this section, we examine how to convert the formulas of our logics to real-time CA, and vice versa. Basically, the transition function (resp. the input mode, the output mode) of the automata will be the counterpart of the computation clauses (resp. the input clauses, the contradiction clauses) of the formulas. Further, the correspondence between the set of computation predicates of the formulas and the set of states of the automata will be based on the following encoding. Starting from a formula whose set of computation predicates is $\mathbf{R}$, the states set $\mathtt{S}$ of the corresponding automaton will be the power set of $\mathbf{R}$; the idea behind is that the state of the cell c at time t records the subset $\{R_i \in \mathbf{R} \mid R_i(c,t) \text{ is true}\}$. By the way, any subset not containing the predicate $R_\perp$ will be an accepting state. Conversely, starting from any automaton, its set of states $\mathtt{S}$ will be expressed by the set of computation predicates $\mathbf{R} = \{R_q \mid q \in \mathtt{S}\}$, with the intuitive meaning that $R_q(c,t)$ is true *iff* the cell c is in state q at time t.

6.1. Logical characterization of $\mathtt{RealTime_{CA}}$

We are now ready to prove Theorem 1 that states the languages accepted in real-time by two-way CA's with input fed in a parallel way and output read on the first cell are exactly the languages defined by the predecessor logic. We recall it here for convenience:

Theorem 1. $\mathtt{RealTime_{CA}} = \mathtt{pred\text{-}ESO\text{-}HORN} = \mathtt{pred\text{-}ESO\text{-}IND}$.

The proof will use the following inclusion scheme:

$$\mathtt{pred\text{-}ESO\text{-}IND} = \mathtt{normal\text{-}pred\text{-}ESO\text{-}IND} \subseteq \mathtt{RealTime_{OIA}} = \mathtt{RealTime_{CA}} \subseteq \mathtt{pred\text{-}ESO\text{-}HORN}.$$

The equality $\mathtt{pred\text{-}ESO\text{-}IND} = \mathtt{normal\text{-}pred\text{-}ESO\text{-}IND}$ has been proved in Section 5 and the other equality $\mathtt{RealTime_{OIA}} = \mathtt{RealTime_{CA}}$ is well-known in automata theory (see [4, 20] and the survey [39], pp. 136-137). The two remaining inclusions are proved in the next two lemmas.

Lemma 6. $\mathtt{RealTime_{CA}} \subseteq \mathtt{pred\text{-}ESO\text{-}HORN}$.

Proof. We will show that for any automaton $\mathcal{A} = (\mathtt{S}, \mathtt{S}_{accept}, \{-1,0,1\}, \mathtt{f}) \in \mathtt{RealTime_{CA}}$ we can build a formula $\Phi \in \mathtt{pred\text{-}ESO\text{-}HORN}$ such that: $w \in \mathcal{L}(\mathcal{A}) \iff \langle w \rangle$ satisfies the formula Φ. We turn $\mathcal{A}$ into an equivalent OCA $\mathcal{A}' = (\mathtt{S}, \mathtt{S}_{accept}, \{-2,-1,0\}, \mathtt{f})$ running within the same computation time n where n is the length of the input word (see Figure 7). This transformation can be seen on the space-time diagrams as the variable change: $(c,t) \mapsto (c+t-1,t)$, where c is the index of the cell and t the time step of the computation.

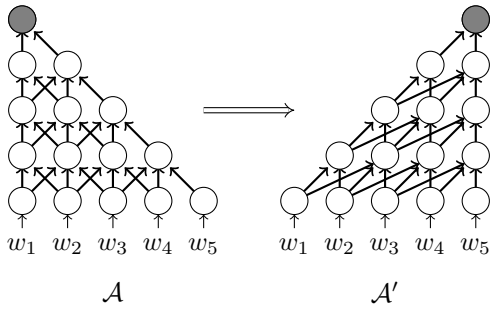


Figure 7: Neighborhood's change: from $\{-1,0,1\}$ to $\{-2,-1,0\}$

Input: With $w = w_1 \dots w_n$ as the input word, the parallel input mode of the automata is expressed by the conjunction

$$\psi_{input} = \bigwedge_{s \in \Sigma} (\mathtt{min}(t) \wedge Q_s(c) \rightarrow R_s(c,t)) \text{ for } \Sigma \text{ the input alphabet}$$

This conjunction tells that the cell c is in state $w_c \in \Sigma$ at the start of the computation.

Computation: Every transition rule $\mathtt{f}(q_2, q_1, q_0) = q$ with $q_2 \neq \sharp$ is expressed by the computation clause:

$$c > t \wedge \neg\mathtt{min}(t) \wedge R_{q_2}(c-2, t-1) \wedge R_{q_1}(c-1, t-1) \wedge R_{q_0}(c, t-1) \rightarrow R_q(c, t).$$

The specific case where q_2 is the permanent state $\sharp$ ($\mathtt{f}(\sharp, q_1, q_0) = q$) is expressed by the clause:

$$c = t \wedge \neg\mathtt{min}(t) \wedge R_{q_1}(c-1, t-1) \wedge R_{q_0}(c, t-1) \rightarrow R_q(c, t).$$

Let $\psi_{compute}$ be the conjunction of the above two sets of clauses.

With $\psi' \coloneqq \psi_{input} \wedge \psi_{compute}$, we have the following equivalence: The computation atom $R_q(c,t)$ is true in the minimal model $(\langle w\rangle, \mathbf{R})$ of $\forall c \forall t \psi'(c,t)$ *iff* the cell c is in state q at time t.

Output: The output get on the last cell is expressed by the contradiction clauses:

$$\psi_{output} \coloneqq \bigwedge_{q \in \mathtt{S} \setminus \mathtt{S}_{accept}} (\mathtt{max}(c) \wedge \mathtt{max}(t) \wedge R_q(c,t) \rightarrow \bot).$$

The formula ψ expressing the computation of $\mathcal{A}'$ is the conjunction $\psi \coloneqq \psi' \wedge \psi_{output}$. For each word $w = w_1 \dots w_n \in \Sigma^+$ we have the following equivalences:

With w as input, $\mathcal{A}$ enters an accepting state on cell 1 at time n
$\Longleftrightarrow$
With w as input, $\mathcal{A}'$ enters an accepting state on cell n at time n
$\Longleftrightarrow$
The conjunction $\bigwedge_{q \in \mathtt{S} \setminus \mathtt{S}_{accept}} \neg R_q(n,n)$ is true in the minimal model $(\langle w\rangle, \mathbf{R})$ of $\forall c \forall t \psi'(c,t)$
$\Longleftrightarrow$
$\langle w\rangle$ satisfies the formula $\exists \mathbf{R} \forall c \forall t \psi(c,t)$.

This proves $\mathcal{L}(\mathcal{A}) \in$ `pred-ESO-HORN`. □

Lemma 7. `normal-pred-ESO-IND` $\subseteq \mathtt{RealTime}_{\mathtt{OIA}}$

Proof. We convert a formula $\Phi \in$ `normal-pred-ESO-IND` into a one-way iterative array $\mathcal{A} = (\mathtt{S}, \mathtt{S}_{accept}, \{-1, 0\}, \mathtt{f}, \mathtt{f}_{\mathtt{input}}) \in \mathtt{RealTime}_{\mathtt{OIA}}$.

Let $\Phi \in$ `normal-pred-ESO-IND` be a formula of the form $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x,y)$ where $\mathbf{R} = \{R_1, \dots, R_m\}$, and ψ is a conjunction of Horn clauses of the following three forms:

1. *input clause* of either form:
   $\mathtt{min}(x) \wedge \mathtt{min}(y) \wedge Q_s(y) \rightarrow R(x,y)$, or $\mathtt{min}(x) \wedge \neg\mathtt{min}(y) \wedge Q_s(y) \rightarrow R(x,y)$,
2. the *contradiction clause*
   $\mathtt{max}(x) \wedge \mathtt{max}(y) \wedge R_\bot(x,y) \rightarrow \bot$,
3. *computation clause* of one of the following forms for some nonempty sets $H, H' \subseteq [1,m]$ and $i \in [1,m]$:
   - $\neg\mathtt{min}(x) \wedge \bigwedge_{h \in H} \epsilon_h R_h(x-1, y) \wedge \neg\mathtt{min}(y) \wedge \bigwedge_{h \in H'} \epsilon_h R_h(x, y-1) \rightarrow R_i(x,y)$,
   - $\neg\mathtt{min}(x) \wedge \bigwedge_{h \in H} \epsilon_h R_h(x-1, y) \rightarrow R_i(x,y)$,
   - $\neg\mathtt{min}(y) \wedge \bigwedge_{h \in H} \epsilon_h R_h(x, y-1) \rightarrow R_i(x,y)$,

   where each ϵ_h either is $\neg$ or is nothing, written $\epsilon_h \in \{-, +\}$

   We denote ψ' the formula ψ without the contradiction clause.

The set of states $\mathtt{S}$ used by $\mathcal{A}$ is the power set of $\mathbf{R} = \{R_1, \dots, R_m\}$ augmented by the quiescent state λ and the permanent state $\sharp$: $\mathtt{S} = \mathcal{P}(\mathbf{R}) \cup \{\lambda, \sharp\}$.
The *input transition function* $\mathtt{f}_{\mathtt{input}}$ only applies to the first cell and the *transition function* $\mathtt{f}$ applies to all the other cells. Below are the definitions of those transition functions where $i \in [1,m]$, $s \in \Sigma$, and $q, l, r \in \mathtt{S} \setminus \{\sharp, \lambda\}$:

The input transition function $\mathtt{f_{input}} : \Sigma \times \mathtt{S} \to \mathtt{S}$: $R_i \in \mathtt{f}_{input}(s, \lambda)$ *iff* there is in ψ an input clause $\mathtt{min}(x) \wedge \mathtt{min}(y) \wedge Q_s(y) \to R_i(x, y)$; moreover, $R_i \in \mathtt{f}_{input}(s, q)$ *iff* there is in ψ an input clause $\mathtt{min}(x) \wedge \neg\mathtt{min}(y) \wedge Q_s(y) \to R_i(x, y)$ or a computation clause $\neg\mathtt{min}(y) \wedge \bigwedge_{h\in H} \epsilon_h R_h(x, y-1) \to R_i(x, y)$ such that, for all $h \in H$, $R_h \notin q$ if $\epsilon_h = -$ and $R_h \in q$ if $\epsilon_h = +$;

The transition function $\mathtt{f} : \mathtt{S} \times \mathtt{S} \to \mathtt{S}$ applies to cells $c \in [2, n]$: $R_i \in \mathtt{f}(l, r)$ *iff* there is in ψ a computation clause $\bigwedge_{h\in H} \epsilon_h R_h(x-1, y) \wedge \neg\mathtt{min}(x) \wedge \bigwedge_{h\in H'} \epsilon_h R_h(x, y-1) \wedge \neg\mathtt{min}(y) \to R_i(x, y)$ such that, for all $h \in H$, $R_h \notin l$ if $\epsilon_h = -$ and $R_h \in l$ if $\epsilon_h = +$, and, for all $h \in H'$, $R_h \notin r$ if $\epsilon_h = -$ and $R_h \in r$ if $\epsilon_h = +$; moreover, $R_i \in \mathtt{f}(l, \lambda)$ *iff* there is in ψ a computation clause $\neg\mathtt{min}(x) \wedge \bigwedge_{h\in H} \epsilon_h R_h(x-1, y) \to R_i(x, y)$ such that, for all $h \in H$, $R_h \notin l$ if $\epsilon_h = -$ and $R_h \in l$ if $\epsilon_h = +$.

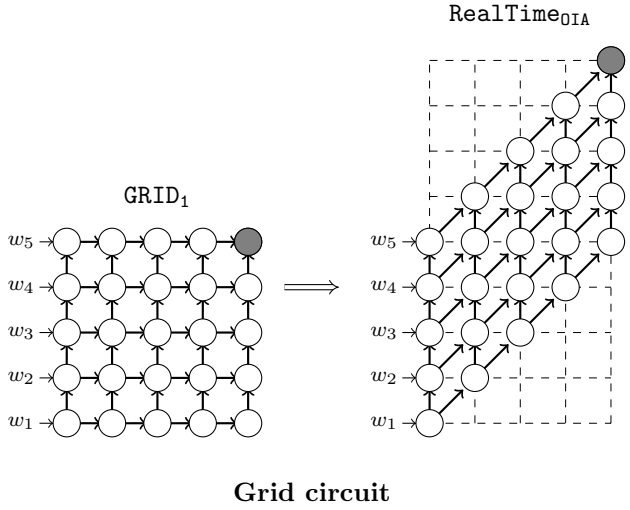


Figure 8: Geometrical interpretation of the transition from $\mathtt{GRID_1}$ to $\mathtt{RealTime_{OIA}}$

By construction of $\mathcal{A}$, we have the following equivalences, for all $w \in \Sigma^n$:

- $\forall (a, b) \in [1, n]^2, \forall i \in [1, m]$, the atom $R_i(a, b)$ is true in the minimal model $(\langle w \rangle, \mathbf{R})$ of $\forall x \forall y\ \psi'(x, y) \iff$ the cell $c = a$ is in a state q with $R_i \in q$ at time $t = a + b - 1$;
- $\langle w \rangle$ satisfies $\Phi \iff \mathcal{A}$ accepts w in real-time.

□

6.2. Logical characterization of $\mathtt{RealTime_{IA}}$

In this section, we will show Theorem 2 that states the languages accepted in real-time by IA are exactly the languages defined by the predecessor logic with diagonal input-output. For convenience, we restate it here:

Theorem 2. $\mathtt{RealTime_{IA}} = \mathtt{pred\text{-}dio\text{-}ESO\text{-}HORN} = \mathtt{pred\text{-}dio\text{-}ESO\text{-}IND}$.

The proof is close to the one of Theorem 1. It is obtained by the following inclusion scheme:

$$\mathtt{pred\text{-}dio\text{-}ESO\text{-}IND} = \mathtt{normal\text{-}pred\text{-}dio\text{-}ESO\text{-}IND} \subseteq \mathtt{RealTime_{IA}} \subseteq \mathtt{pred\text{-}dio\text{-}ESO\text{-}HORN}.$$

The equality $\mathtt{pred\text{-}dio\text{-}ESO\text{-}IND} = \mathtt{normal\text{-}pred\text{-}dio\text{-}ESO\text{-}IND}$ has been proved in Section 5. The two remaining inclusions are proved in the next two lemmas.

Lemma 8. $\mathtt{RealTime_{IA}} \subseteq \mathtt{pred\text{-}dio\text{-}ESO\text{-}HORN}$.

Proof. Let $\mathcal{A} = (\mathtt{S}, \mathtt{S}_{accept}, \{-1, 0, 1\}, \mathtt{f_{input}}, \mathtt{f})$ be an automaton in $\mathtt{RealTime_{IA}}$. We turn $\mathcal{A}$ into a new automaton $\mathcal{A}' = (\mathtt{S}, \mathtt{S}_{accept}, \{-2, -1, 0\}, \mathtt{f_{input}}, \mathtt{f})$. This transformation can be seen on the space-time diagrams as the variable change: $(c, t) \mapsto (c + t - 1, t)$. The input is still fed sequentially but in the following way: the i^{th} bit of the input is given to the cell i at time i as shown in Figure 9. Since the input presentation is the only change between $\mathtt{RealTime_{IA}}$ and $\mathtt{RealTime_{CA}}$, we obtain in

the same way the conjunction of computation clauses $\psi_{compute}$ and the conjunction of contradiction clauses ψ_{output}.

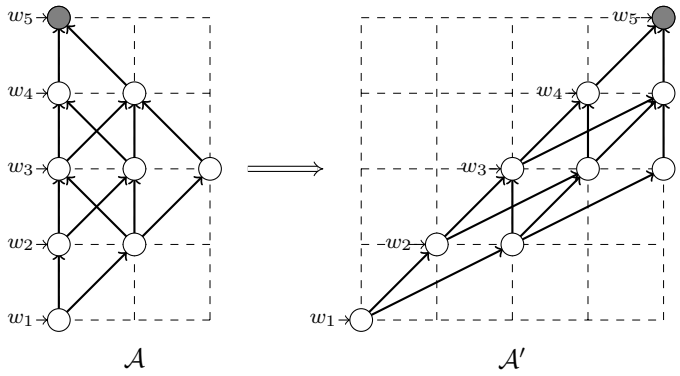


Figure 9: Neighborhood's change: from $\{-1, 0, 1\}$ to $\{-2, -1, 0\}$

Input: With $w = w_1 \dots w_n$ as the input word, the diagonal input mode of $\mathcal{A}'$ is expressed by the conjunction:

$$\psi_{input} \coloneqq \bigwedge_{s \in \Sigma} (t = c \wedge Q_s(c) \rightarrow R_s(c, t)) \text{ for } \Sigma \text{ the input alphabet}$$

This conjunction tells that the cell c is in the state $w_c \in \Sigma$ at time c.

Let ψ' be the conjunction $\psi_{input} \wedge \psi_{compute}$. The formula ψ of `pred-dio-ESO-HORN` expressing the computation of $\mathcal{A}'$ is $\psi \coloneqq \psi' \wedge \psi_{output}$. For each word $w = w_1 \dots w_n \in \Sigma^+$, we have the following equivalences:

With w as input, $\mathcal{A}$ enters an accepting state on cell 1 at time n
$\iff$
With w as input, $\mathcal{A}'$ enters an accepting state on cell n at time n
$\iff$
The conjunction $\bigwedge_{q \in \mathtt{S} \setminus \mathtt{S}_{accept}} \neg R_q(n, n)$ is true in the minimal model $(\langle w \rangle, \mathbf{R})$ of $\forall c \forall t \psi'(c, t)$
$\iff$
$\langle w \rangle$ satisfies the formula $\exists \mathbf{R} \forall c \forall t \psi(c, t)$.

This proves $\mathcal{L}(\mathcal{A}) \in$ `pred-dio-ESO-HORN`.

□

Lemma 9. `normal-pred-dio-ESO-IND` $\subseteq$ $\mathtt{RealTime}_{\mathtt{IA}}$.

Proof. Since the proof is similar to that of Lemma 7, we will here just give an hint on how to associate to each site $(x, y) \in [1, n]^2$ of $\mathtt{GRID}_2$ with $x \leq y$, a site (c, t) of the space-time diagram of an iterative array $\mathcal{A}$ running in real-time (see Figure 10). The sites $(x, y) \in [1, n]^2$ with $x \geq y$ are similarly handled.
First, we apply to the set of sites (x, y) of $\mathtt{GRID}_2$ the variable change $c' = y - x + 1$; $t' = x + y - 1$. This variable change turns respectively the dependencies $(x - 1, y) \rightarrow (x, y)$ and $(x, y - 1) \rightarrow (x, y)$ into $(c' + 1, t' - 1) \rightarrow (c', t')$ and $(c' - 1, t' - 1) \rightarrow (c', t')$ expressing the two-way communication of an iterative array $\mathcal{A}'$. The sites (c', t') of the space-time diagram of $\mathcal{A}'$ takes their values in $[1, n] \times [1, 2n - 1]$ and the i^{th} bit of the input is fed on the site $(1, 2i - 1)$ (see Figure 10).
Second, in order to obtain the space-time diagram of an IA $\mathcal{A}$ running in real-time, each site $(c, t) = (\lceil c'/2 \rceil, \lceil t'/2 \rceil)$ of this diagram will record the set of sites $\{(c' - 1, t' - 1), (c', t'), (c' + 1, t' - 1)\}$ of the space-time diagram of $\mathcal{A}'$, with c' and t' odd and greater than 1 (see Figure 10). □

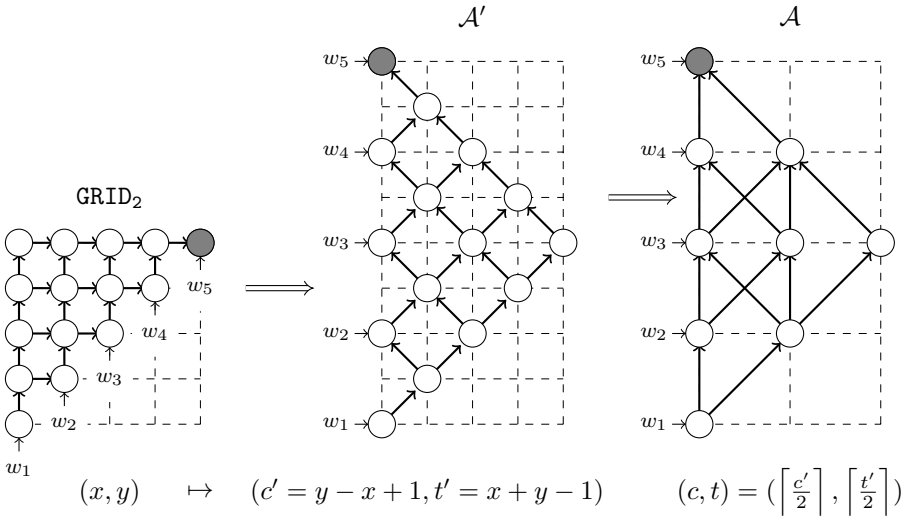


Figure 10: Variable change and grouping

6.3. Logical characterization of `Trellis` *and linear conjunctive grammars*

Introduced by Okhotin [28], conjunctive grammars are an extension of context-free grammars. This class of formal grammars is obtained by allowing the use of a conjunction operator (denoted &) in the right side of the context-free rules, meaning an intersection between derived languages. It has been shown in [28] that the class of languages obtained by the linear restriction of conjunctive grammars (denoted `LinConj`) is equal to `Trellis`. Each rule of a linear conjunctive grammar $\mathcal{G} = (\Sigma, N, P, S)$ in normal form is a rule of the form

$$A \to sB_1 \& \ldots \& sB_\ell \& C_1 t \& \ldots \& C_p t, \text{ or } A \to s,$$

with $\ell + p \geq 1$, $A, B_i, C_j \in N$, and $s, t \in \Sigma$.

As a context-free language, a conjunctive language has an algebraic representation as the least solution of a system of language equations with concatenation, union and moreover *intersection*. For example, the rule

$$A \to sB_1 \,\&\, \ldots \,\&\, sB_\ell \,\&\, C_1 t \,\&\, \ldots \,\&\, C_p t$$

gives the language equation:

$$\mathcal{L}(A) = s\mathcal{L}(B_1) \cap \cdots \cap s\mathcal{L}(B_\ell) \cap \mathcal{L}(C_1)t \cap \cdots \cap \mathcal{L}(C_p)t$$

where $\mathcal{L}(A)$ is the set of words having the property A.

In this subsection, we prove (or reprove) in a simple way the equality

$$\mathtt{LinConj} = \mathtt{incl\text{-}ESO\text{-}HORN} = \mathtt{Trellis}$$

First, notice that our inclusion logic naturally characterizes the class of complements of linear conjunctive languages[12]:

Theorem 4. *For any $L \in \Sigma^+$, we have: $L \in$* `incl-ESO-HORN` $\iff (\Sigma^+ \setminus L) \in$ `LinConj`.

Proof. The key point is that normal forms of linear conjunctive grammars are *essentially the same* as those of `incl-ESO-HORN`. To each rule of a linear conjunctive grammar $\mathcal{G} = (\Sigma, N, P, S)$ in normal form $A \to s$ (resp. $A \to sB_1 \,\&\, \ldots \,\&\, sB_\ell \,\&\, C_1 t \,\&\, \ldots \,\&\, C_p t$) associate the clause $x = y \wedge Q_s(x) \to A(x, y)$ (resp. $x < y \wedge Q_s(x) \wedge \bigwedge_{i=1}^{\ell} B_i(x + 1, y) \wedge \bigwedge_{i=1}^{p} C_i(x, y - 1) \wedge Q_t(y) \to A(x, y)$), where the computation predicates A, B_i, C_i are associated to the nonterminal symbols with the same names. Let $\psi_\mathcal{G}$ be the conjunction of the clauses so obtained with the contradiction clause $\mathtt{min}(x) \wedge \mathtt{max}(y) \wedge S(x, y) \to \bot$. Define $\Phi_\mathcal{G} := \exists N \forall x \forall y \; \psi_\mathcal{G}$,

[12] Then, it will be sufficient to prove the equality `incl-ESO-HORN` = `Trellis` (Theorem 3) since the class of languages `Trellis` is obviously closed under complement.

where N now denotes the set of computation predicates associated to the nonterminal symbols of N. Clearly, $\Phi_{\mathcal{G}}$ belongs to `incl-ESO-HORN` and the following equivalence holds for each $w \in \Sigma^+$: $w \notin \mathcal{L}(\mathcal{G}) \iff \langle w \rangle \models \Phi_{\mathcal{G}}$.

Conversely, to each formula $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x,y)$ in `normal-incl-ESO-HORN`, associate the linear conjunctive grammar $\mathcal{G}_\Phi = (\Sigma, N, P, S)$ defined as follows: $N := \mathbf{R}$ and $S := R_\perp$; the set of rules P is the following: to each input clause $x = y \wedge Q_s(x) \to R(x,y)$ of Φ associate the rule $R \to s$, and to each computation clause $x < y \wedge \bigwedge_{i=1}^{\ell} S_i(x+1,y) \wedge \bigwedge_{i=1}^{p} T_i(x,y-1) \to R(x,y)$ of Φ and each $s,t \in \Sigma$, associate the rule $R \to sS_1 \,\&\, \ldots \,\&\, sS_\ell \,\&\, T_1 t \,\&\, \ldots \,\&\, T_p t$. Clearly, the following equivalence holds for each $w \in \Sigma^+$: $\langle w \rangle \models \Phi \iff w \notin \mathcal{L}(\mathcal{G}_\Phi)$ □

Second, notice that the normalized inclusion logic naturally expresses the computation of trellis automata.

Theorem 3. `Trellis` = `incl-ESO-HORN` = `incl-ESO-IND`.

Theorem 3 is proved using the following inclusion scheme:

$$\texttt{incl-ESO-IND} = \texttt{normal-incl-ESO-IND} \subseteq \texttt{Trellis} \subseteq \texttt{incl-ESO-HORN}.$$

The equality `incl-ESO-IND` = `normal-incl-ESO-IND` has been proved in Section 5. The two remaining inclusions are proved in the next two lemmas.

Lemma 10. `Trellis` ⊆ `incl-ESO-HORN`.

Proof. Let $\mathcal{A} = (\mathtt{S}, \mathtt{S}_{accept}, \{-1,0\}, \mathtt{f})$ be a trellis automaton that accepts some language $L \subseteq \Sigma^+$ in real-time and let $w = w_1 \ldots w_n$ be a word on Σ. The final state q of $\mathcal{A}$ acting on the word $w_x \ldots w_y$ is completely determined by the final state q_l of $\mathcal{A}$ acting on the word $w_x \ldots w_{y-1}$ and the final state q_r of $\mathcal{A}$ acting on the word $w_{x+1} \ldots w_y$. Therefore, we introduce the set of binary predicates $\mathbf{R} = \{R_q \mid q \in \mathtt{S}\}$ with intuitive meaning: $R_q(x,y)$ is true $\iff$ the final state of $\mathcal{A}$ acting on the subword $w_x \ldots w_y$ is q. The functional dependence $((x,y-1),(x+1,y)) \mapsto (x,y)$ will then be exactly expressed by our computation clauses.

The following clauses describe the computation of $\mathcal{A}$.

Input clauses:

$$\psi_{input} \coloneqq \bigwedge_{s \in \Sigma} (x = y \wedge Q_s(x) \to R_s(x,y)).$$

Computation clauses:

$$\psi_{compute} \coloneqq \bigwedge_{(q_l,q_r) \in \mathtt{S}^2} (x < y \wedge R_{q_l}(x,y-1) \wedge R_{q_r}(x+1,y) \to R_q(x,y)) \text{ where } q = \mathtt{f}(q_l,q_r).$$

Contradiction clauses:

$$\psi_{output} \coloneqq \bigwedge_{q \in \mathtt{S} \setminus \mathtt{S}_{accept}} (\mathtt{min}(x) \wedge \mathtt{max}(y) \wedge R_q(x,y) \to \perp).$$

Let $\psi' \coloneqq \psi_{input} \wedge \psi_{compute}$ and $\psi \coloneqq \psi' \wedge \psi_{output}$. For each word $w = w_1 \ldots w_n \in \Sigma^+$ we have the following equivalences:

$\mathcal{A}$ accepts w in time n
$\iff$
The conjunction $\bigwedge_{q \in \mathtt{S} \setminus \mathtt{S}_{accept}} \neg R_q(1,n)$ is true in the minimal model $(\langle w \rangle, \mathbf{R})$ of $\forall x \forall y \psi'(x,y)$
$\iff$
$\langle w \rangle$ satisfies the formula $\exists \mathbf{R} \forall x \forall y \psi(x,y)$.

This proves $\mathcal{L}(\mathcal{A}) \in$ `incl-ESO-HORN`. □

Lemma 11. `normal-incl-ESO-IND` ⊆ `Trellis`.

Proof. Let Φ be a formula of `normal-incl-ESO-IND`. We establish a natural bijection between the sites (x, y) of the domain of the formula and the sites (c, t) of the space-time diagram of a OCA running in real-time: see Figure 11. The transition function of this automaton is then deduced from the computation clauses of Φ as in the proofs of Lemma 7 in Section 6.1 and of Lemma 9 in Section 6.2. □

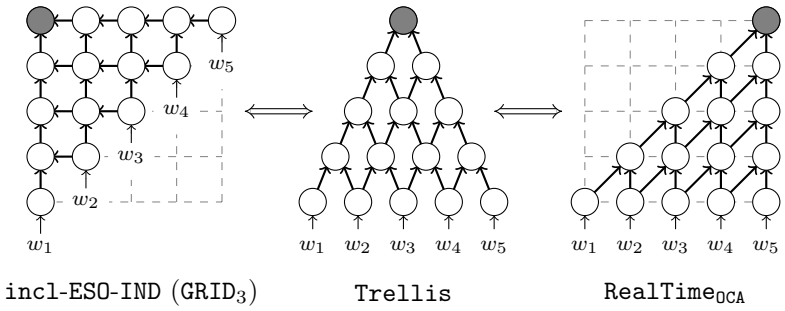


Figure 11: Bijection between `incl-ESO-IND`, `Trellis` and `RealTime`$_{\text{OCA}}$

7. Programming some reference problems in our logics

In this section, we will exhibit natural inductive definitions within our logics for some famous problems of the literature. By the results of the previous sections, such a logical definition can be automatically translated into a program of a real-time cellular automaton of the appropriate type: CA, IA or trellis automaton.

7.1. Defining in predecessor Horn logic the product of integers

We now show that school multiplication is naturally described in logic `pred-ESO-HORN`. Since our logics only define decision problems, the problem should be formalized as $\texttt{Product} := \{(a, b, p) \in (\mathbb{N}_{>0})^3 \mid a \times b = p\}$, where each integer is represented in binary notation. More precisely, let $p = a \times b$ be a product of positive integers and let $p = \overline{p_n \dots p_1} = 2^{n-1}p_n + \dots + 2p_2 + p_1$, with $p_n = 1$, $\texttt{length}(p) = n$, be the binary writing of the product; similarly, represent the operands of the product on n bits as $a = \overline{a_n \dots a_1}$ and $b = \overline{b_n \dots b_1}$, adding leading zeros if necessary. Let us represent an input $(a, b, p) \in (\mathbb{N}_{>0})^3$, $\texttt{length}(p) = n$, $\texttt{length}(a) \leq n$, and $\texttt{length}(b) \leq n$, by the word $t = t_1 \dots t_n$ on the alphabet $\{0, 1\}^3$, where $t_x = (a_x, b_x, p_x)$, for $x \in [1, n]$. This word is naturally translated into the *input structure* of domain $[1, n]$

$$\langle a, b, p\rangle := ([1, n]; A_0, A_1, B_0, B_1, P_0, P_1, \texttt{min}, \texttt{max}, \texttt{pred})$$

where the input predicates A_i, B_i, P_i, for $i \in \{0, 1\}$, are defined as $A_i(x) \iff a_x = i$, $B_i(x) \iff b_x = i$, and $P_i(x) \iff p_x = i$, for $x \in [1, n]$.

The school multiplication. Let us define the successive partial sums, for $1 \leq x \leq n$:

$$c(x) := \Sigma_{z=1}^{x} a \times b_z 2^{z-1}$$

so that $p = c(n)$. We have the induction $c(1) = a \times b_1$ and $c(x) = c(x-1) + a \times b_x \times 2^{x-1}$, for $1 < x \leq n$.

If, for $x \geq 1$, we define $e(x) := a \times 2^{x-1}$ and $d(x) := b_x \times e(x)$, then one can write $c(x) = c(x-1) + d(x)$.

Computation predicates. We use the six binary predicates C_i, D_i, E_i, for $i \in \{0, 1\}$, defined on $[1, n]$ by the equivalence: $C_i(x, y)$ (resp. $D_i(x, y)$, $E_i(x, y)$) $\iff$ the bit y of $c(x)$ (resp. $d(x)$, $e(x)$) is i.

Clauses defining predicates E_i by expressing $e(x) = a \times 2^{x-1}$.

- clauses expressing $e(1) = a$: $x = 1 \wedge A_i(y) \to E_i(x, y)$, for $i \in \{0, 1\}$;
- clauses expressing $e(x) = 2 \times e(x-1)$, for $x > 1$:
  - $x > 1 \wedge y = 1 \to E_0(x, y)$;
  - $x > 1 \wedge y > 1 \wedge E_i(x-1, y-1) \to E_i(x, y)$, for $i \in \{0, 1\}$.

Clauses defining predicates D_i by expressing $d(x) = b_x \times e(x)$.

- $B_0(x) \to D_0(x,y)$; $B_1(x) \wedge E_i(x,y) \to D_i(x,y)$, for $i \in \{0,1\}$

It remains to implement the induction: $c(1) = d(1)$ and $c(x) = c(x-1) + d(x)$, for $x > 1$. We use for that the classical addition algorithm.

Addition algorithm `Add`. Let us define the sum $w = \overline{w_n \dots w_1} = u + v$ of two integers $u = \overline{u_n \dots u_1}$ and $v = \overline{v_n \dots v_1}$, written with n bits, for $\texttt{length}(u+v) \leq n$. Let $r_1, \dots, r_n$ denote the sequence of carries of this addition, with $r_i \in \{0,1\}$. We have the equalities

`Add`: $u_1 + v_1 = \overline{r_1 w_1}$, and $u_y + v_y + r_{y-1} = \overline{r_y w_y}$, for $y > 1$, with $r_n = 0$ (since $w < 2^n$).

Clauses defining predicates C_i and the carry predicates R_i.

- the clauses $x = 1 \wedge D_i(x,y) \to C_i(x,y)$, for $i \in \{0,1\}$, express $c(1) = d(1)$;
- the following clauses express $c(x) = c(x-1) + d(x)$, for $1 < x \leq n$, by implementing the addition algorithm `Add` and using the new computation predicates R_0, R_1 for carries, with $h, i, j \in \{0,1\}$:
  1. $x > 1 \wedge y = 1 \wedge C_h(x-1,y) \wedge D_i(x,y) \to (R_k(x,y) \wedge C_\ell(x,y))$, where $h + i = \overline{k\ell}$;
  2. $x > 1 \wedge y > 1 \wedge C_h(x-1,y) \wedge D_i(x,y) \wedge R_j(x,y-1) \to (R_k(x,y) \wedge C_\ell(x,y))$, where $h + i + j = \overline{k\ell}$;
  3. $y = n \wedge R_1(x,y) \to \bot$ (meaning: $c(x) < 2^n$).

  Remark: For a better reading, we have written the conclusion of clauses (1) and (2) as a conjunction of two atoms. Such a clause can be trivially rewritten as the conjunction of two Horn clauses.

Clauses that check $c(n) = p$.

- $x = n \wedge C_i(x,y) \wedge \neg P_i(y) \to \bot$, for $i \in \{0,1\}$ (meaning: $C_i(n,y) \to P_i(y)$).

Let $\Phi_{\texttt{Product}}$ denote the formula $\exists C_0, C_1, D_0, D_1, E_0, E_1, R_0, R_1 \ \forall x, y \ \psi$, where ψ denotes the conjunction of the previous clauses. By construction, we have $\Phi_{\texttt{Product}} \in$ `pred-ESO-HORN`. So, the following proposition is proved:

Proposition 1. *For all $(a,b,p) \in (\mathbb{N}_{>0})^3$, we have the equivalence:* $(a,b,p) \in$ `Product` iff $\langle a,b,p \rangle \models \Phi_{\texttt{Product}}$. *This implies* `Product` $\in$ `pred-ESO-HORN` *and therefore* `Product` $\in \texttt{RealTime}_{\texttt{CA}}$ *[2, 27].*

7.2. Expressing in logic the set of primes by Fischer's algorithm

Fischer's algorithm [12] generates in real-time on a cellular automaton the set of prime numbers denoted `Prime` by the sieve of Eratosthenes: that means there is some state $q_{\texttt{Prime}}$ of the automaton such that at each instant t, the state of the first cell of the automaton is $q_{\texttt{Prime}}$ *iff* $t \in$ `Prime`. We now want to reformulate Fischer's algorithm in logic. Let us identify each positive integer n and its unary notation 1^n; so, `Prime` is the language of words 1^p of length $p \in$ `Prime`. We will define `Prime` in `pred-ESO-IND`, or, here in an equivalent way, in `pred-dio-ESO-IND`.

Let us classify the clauses defining `Prime` and the predicates they use in two parts:

- the clauses defining `Prime` by the sieve of Eratosthenes and by using two pre-definite predicates `Square` and `Rail`: those clauses construct two new predicates called suggestively `Multiple` and `Rebound`;
- the more technical clauses defining the predicates `Square` and `Rail` and three auxiliary predicates denoted `HalfSlope`, `Vertical` and `Horizontal`.

Defining `Prime` *by using the pre-definite predicates* `Square` *and* `Rail`. Let `Square` and `Rail` denote the binary predicates on domain $[1, n]$ defined as follows (see Figure 12):

- $\texttt{Square}(x, y) \iff \exists k \geq 2,\ x = k^2 \wedge y = \Sigma_{i=1}^{k-1} i$;
- $\texttt{Rail}(x, y) \iff \exists k \geq 1,\ x \geq k^2 \wedge x = y + \Sigma_{i=1}^{k} i$.

Idea: The predicate `Rail` is composed of parallel "rails"; the k-th rail, denoted $rail_k$, has equation $x = y + \Sigma_{i=1}^{k} i$, for $x \geq k^2$. The first point of $rail_k$ is the site (x, y) such that $\texttt{Square}(x, y)$ with $x = k^2$. The "corridor" between two consecutive rails, $rail_{k-1}$ and $rail_k$, for $k \geq 2$, has horizontal (and vertical) width k. This allows to define the set of multiples $\geq k^2$ of the integer k by zig-zag (vertical-horizontal) between $rail_{k-1}$ and $rail_k$.

The predicates `Multiple` (the vertical "zig") and `Rebound` (the horizontal "zag") are defined by the following clauses:

- $\texttt{Square}(x, y) \rightarrow \texttt{Multiple}(x, y)$ (initialization of vertical zig with $x = k^2$);
- $\texttt{Multiple}(x, y-1) \wedge \neg\texttt{Rail}(x-1, y-1) \rightarrow \texttt{Multiple}(x, y)$ (vertical zig);
- $\texttt{Multiple}(x, y-1) \wedge \texttt{Rail}(x-1, y-1) \rightarrow \texttt{Rebound}(x, y)$ (zig meets a rail);
- $\texttt{Rebound}(x-1, y) \wedge \neg\texttt{Rail}(x-1, y-1) \rightarrow \texttt{Rebound}(x, y)$ (horizontal zag);
- $\texttt{Rebound}(x-1, y) \wedge \texttt{Rail}(x-1, y-1) \rightarrow \texttt{Multiple}(x, y)$ (zag meets a rail).

The following equivalences can be established by induction:

- $\texttt{Multiple}(x, y) \iff \exists k \geq 2, \exists k' \geq 0,\ x = k(k+k') \wedge kk' + \Sigma_{i=1}^{k-1} i \leq y < kk' + \Sigma_{i=1}^{k} i$;
- $\texttt{Rebound}(x, y) \iff \exists k \geq 2, \exists k' \geq 0,\ y = kk' + \Sigma_{i=1}^{k} i \wedge k(k+k') \leq x < k(k+k'+1)$.

An integer $x \geq 2$ is prime *iff* it is not of the form $x = kk'$ for any $k' \geq k \geq 2$, i.e., *iff* x is not a multiple $\geq k^2$ of k for any $k \geq 2$. So, the following clause, which is equivalent to $\forall y\ \neg\texttt{Multiple}(n, y)$, expresses that n is a prime integer:

- $x = n \wedge \texttt{Multiple}(x, y) \rightarrow \bot$.

It remains to define the predicates `Square`, `Rail`.

Defining the predicates `Square`*,* `Rail`*,* `HalfSlope`*,* `Vertical` *and* `Horizontal`. Let `HalfSlope`, `Vertical` and `Horizontal` denote the following predicates (see Figure 12):

- $\texttt{HalfSlope}(x, y) \iff \exists k, k',\ 0 \leq k' < k \wedge x = k^2 + 1 + 2k' \wedge y = k' + \Sigma_{i=1}^{k-1} i$;
- $\texttt{Vertical}(x, y) \iff \exists k, k',\ 0 < k' < k \wedge x = k^2 \wedge y = k' + \Sigma_{i=1}^{k-1} i$;
- $\texttt{Horizontal}(x, y) \iff \exists k \geq 2,\ k^2 \leq x \leq k^2 + 2k \wedge y = \Sigma_{i=1}^{k} i$.

The predicates `Square` and `Rail` are defined inductively and simultaneously with the auxiliary predicates `HalfSlope`, `Vertical` and `Horizontal` by the following clauses:

- *Initialization clauses:* $x = 2 \wedge y = 1 \rightarrow \texttt{Rail}(x, y)$; $x = 4 \wedge y = 1 \rightarrow \texttt{Square}(x, y)$;
- *Computing the rails:* $\texttt{Square}(x, y) \rightarrow \texttt{Rail}(x, y)$; $\texttt{Rail}(x-1, y-1) \rightarrow \texttt{Rail}(x, y)$;
- *Defining the successive* `Square` *points by defining and using the predicates* `HalfSlope`*,* `Vertical` *and* `Horizontal`*:*
  - *Defining* `HalfSlope`*:* $\texttt{Square}(x-1, y) \rightarrow \texttt{HalfSlope}(x, y)$;
    $\texttt{HalfSlope}(x-2, y-1) \wedge \neg\texttt{Horizontal}(x-1, y) \rightarrow \texttt{HalfSlope}(x, y)$;
  - *Defining* `Vertical`*:* $\texttt{Square}(x, y-1) \rightarrow \texttt{Vertical}(x, y)$;
    $\texttt{Vertical}(x, y-1) \wedge \neg\texttt{Rail}(x-1, y-1) \rightarrow \texttt{Vertical}(x, y)$;
  - *Defining* `Horizontal`*:* $\texttt{Vertical}(x, y-1) \wedge \texttt{Rail}(x-1, y-1) \rightarrow \texttt{Horizontal}(x, y)$;
    $\texttt{Horizontal}(x-1, y) \wedge \neg\texttt{HalfSlope}(x-2, y-1) \rightarrow \texttt{Horizontal}(x, y)$;

– *Defining the next* `Square` *point:*
$\texttt{Horizontal}(x-1,y) \wedge \texttt{HalfSlope}(x-2,y-1) \rightarrow \texttt{Square}(x,y)$.

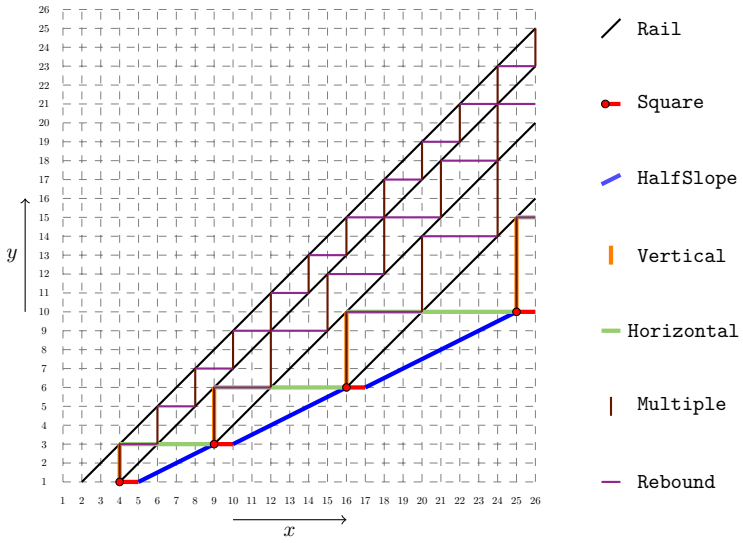


Figure 12: A computation example of $\Phi_{\texttt{Prime}}$ on $\texttt{GRID}_2$

Let ψ denote the conjunction of the previous clauses and let $\sigma_{\texttt{Prime}}$ denote its set of computation predicates : $\sigma_{\texttt{Prime}} := \{\texttt{Multiple}, \texttt{Rebound}, \texttt{Square}, \texttt{Rail}, \texttt{HalfSlope}, \texttt{Vertical}, \texttt{Horizontal}\}$. Let $\Phi_{\texttt{Prime}}$ denote the formula $\exists\sigma_{\texttt{Prime}}\ \forall x, y\ \psi$. By construction, we have $\Phi_{\texttt{Prime}} \in \texttt{pred-dio-ESO-IND}$. So, the following proposition is proved:

Proposition 2. *For any integer $p \geq 2$, we have the equivalence: $p \in$* `Prime` iff $\langle p \rangle \models \Phi_{\texttt{Prime}}$. *This implies* $\texttt{Prime} \in \texttt{pred-dio-ESO-IND}$.

7.3. Dyck languages, inclusion inductive logic and linear conjunctive grammars

For an integer $k \geq 1$, let $\texttt{Dyck}_k$ denote the set of well-parenthesed non-empty expressions (Dyck words) on the alphabet with k types of parentheses $\Sigma_k = \{\ (_i \mid 1 \leq i \leq k\} \cup \{\ )_i \mid 1 \leq i \leq k\}$.

Definition 9. *A factor (resp. prefix, suffix) of a Dyck word is called a* Dyck factor *(resp.* Dyck prefix*,* Dyck suffix*).*

Principle of the decision algorithm on $\texttt{GRID}_3$*:.* Recall that the input word w is placed on the diagonal $x = y$ of the grid $n \times n$, where n is the length of $w = w_1 \dots w_n$. Each opening parenthesis $(_i$ ascends vertically, i.e., for y increasing, if it does not encounter a closing one; a closing parenthesis $)_i$ goes horizontally to the left, i.e., for x decreasing, if it does not encounter an opening one. The input word is accepted if none of the following events occurs:

1. an opening parenthesis $(_i$ meets a closing parenthesis $)_j$ of a different type $j \neq i$;
2. an opening parenthesis $(_i$ reaches the upper side $y = n$ without meeting a closing parenthesis;
3. a closing parenthesis $)_i$ reaches the left side $x = 1$ without meeting an opening parenthesis.

A formula of `incl-ESO-IND` *that defines* $\texttt{Dyck}_k$

It is the formula $\Phi_{\texttt{Dyck}_k} := (\exists R_{(_i}, R_{)_i})_{1 \leq i \leq k} \exists R_\perp^{\texttt{min}} \exists R_\perp^{\texttt{max}} \exists R_\perp\ \forall x \forall y\ \psi$. Here, ψ is the conjunction of the clauses given below that involve the computation predicates $R_{(_i}, R_{)_i}$, for $1 \leq i \leq k$, $R_\perp^{\texttt{min}}$, $R_\perp^{\texttt{max}}$ and $R_\perp$, with the following intended meaning:

- $R_{(_i}(x,y) \iff w_x \dots w_y$ is a Dyck factor starting with an opening parenthesis $w_x = (_i$ which is not matched in the word $w_x \dots w_y$;
- $R_{)_i}(x,y) \iff w_x \dots w_y$ is a Dyck factor ending with a closing parenthesis $w_y =)_i$ which is not matched in the word $w_x \dots w_y$;

- $R_{\perp}^{\mathtt{min}}(x,y) \iff w_x \dots w_y$ is a Dyck factor but not a Dyck prefix, therefore if $x = 1$ then w is not a Dyck word;
- $R_{\perp}^{\mathtt{max}}(x,y) \iff w_x \dots w_y$ is a Dyck factor but not a Dyck suffix, therefore if $y = n$ then w is not a Dyck word;
- $R_{\perp}(x,y) \iff w_x \dots w_y$ is not a Dyck factor.

Inductive definitions of predicates

Defining predicates $R_{(_i}$ and $R_{)_i}$. We have $R_{(_i}(x,y)$, i.e., $w_x \dots w_y$ is a Dyck factor starting with an opening parenthesis $w_x = (_i$ which is not matched in the factor $w_x \dots w_y$ *iff* condition (1) or (2) occurs:

(1) $x = y$ and $w_x = (_i$;

(2) $x < y$ and we have $R_{(_i}(x, y-1)$ and $w_{x+1} \dots w_y$ is a Dyck factor where w_y is either a closing parenthesis which is matched in the factor $w_{x+1} \dots w_y$ or an opening parenthesis.

By this equivalence, $R_{(_i}$ is defined by the conjunction of the two following clauses:

- $x = y \wedge Q_{(_i}(x) \to R_{(_i}(x,y)$;
- $x < y \wedge R_{(_i}(x, y-1) \wedge \neg R_{\perp}(x+1, y) \wedge \bigwedge_{j=1}^{k} \neg R_{)_j}(x+1, y) \to R_{(_i}(x,y)$.

For similar reasons, the conjunction of the two following clauses defines $R_{)_i}$:

- $x = y \wedge Q_{)_i}(x) \to R_{)_i}(x,y)$;
- $x < y \wedge \neg R_{\perp}(x, y-1) \wedge \bigwedge_{j=1}^{k} \neg R_{(_j}(x, y-1) \wedge R_{)_i}(x+1, y) \to R_{)_i}(x,y)$.

Defining predicate $R_{\perp}$. We have $R_{\perp}(x,y)$, i.e., $w_x \dots w_y$ is not a Dyck factor *iff* condition (1), (2) or (3) occurs:

(1) $x < y$ and there are i, j, $i \neq j$, such that:
   - we have $R_{(_i}(x, y-1)$, i.e., $w_x \dots w_{y-1}$ is a Dyck factor starting with the parenthesis $w_x = (_i$ which is not matched in $w_x \dots w_{y-1}$;
   - we have $R_{)_j}(x+1, y)$, i.e., $w_{x+1} \dots w_y$ is a Dyck factor ending with the parenthesis $w_y =)_j$ which is not matched in $w_{x+1} \dots w_y$;

(2) $x < y$ and $w_x \dots w_{y-1}$ is not a Dyck factor;

(3) $x < y$ and $w_{x+1} \dots w_y$ is not a Dyck factor.

By this equivalence, $R_{\perp}$ is defined by the conjunction of the following clauses:

- $x < y \wedge R_{(_i}(x, y-1) \wedge R_{)_j}(x+1, y) \to R_{\perp}(x,y)$, for $1 \leq i, j \leq k$ and $i \neq j$;
- $x < y \wedge R_{\perp}(x, y-1) \to R_{\perp}(x,y)$;
- $x < y \wedge R_{\perp}(x+1, y) \to R_{\perp}(x,y)$.

Defining predicates $R_{\perp}^{\mathtt{max}}$ and $R_{\perp}^{\mathtt{min}}$. We have $R_{\perp}^{\mathtt{max}}(x,y)$, i.e., $w_x \dots w_y$ is a Dyck factor but not a Dyck suffix *iff* condition (1) or (2) occurs:

(1) we have $R_{(_i}(x,y)$, for some i, i.e., $w_x \dots w_y$ is a Dyck factor starting with an opening parenthesis $w_x = (_i$ which is not matched in the factor $w_x \dots w_y$;

(2) $x < y$ and $w_x \dots w_{y-1}$ is a Dyck factor and $w_{x+1} \dots w_y$ is a Dyck factor but not a Dyck suffix.

By this equivalence, $R_{\perp}^{\mathtt{max}}$ is defined by the conjunction of the following clauses:

- $R_{(_i}(x,y) \to R_{\perp}^{\mathtt{max}}(x,y)$, for $1 \leq i \leq k$;

- $x < y \wedge \neg R_\perp(x, y-1) \wedge R_\perp^{\texttt{max}}(x+1, y) \rightarrow R_\perp^{\texttt{max}}(x, y)$.

For similar reasons, the conjunction of the two following clauses defines $R_\perp^{\texttt{min}}$:

- $R_{)_i}(x, y) \rightarrow R_\perp^{\texttt{min}}(x, y)$, for $1 \leq i \leq k$;
- $x < y \wedge R_\perp^{\texttt{min}}(x, y-1) \wedge \neg R_\perp(x+1, y) \rightarrow R_\perp^{\texttt{min}}(x, y)$.

Contradiction clauses. A word w is a Dyck word if the three following conditions hold: all closing parenthesis of w have a match, all opening parenthesis of w have a match and w only contains Dyck factors. It is expressed by the following clauses:

1') $x = 1 \wedge y = n \wedge R_\perp^{\texttt{min}}(x, y) \rightarrow \perp$;

2') $x = 1 \wedge y = n \wedge R_\perp^{\texttt{max}}(x, y) \rightarrow \perp$;

3') $x = 1 \wedge y = n \wedge R_\perp(x, y) \rightarrow \perp$.

By construction, the formula $\Phi_{\texttt{Dyck}_k} \in \texttt{incl-ESO-IND}$ defines the language $\texttt{Dyck}_k$. Therefore, we have the following proposition:

Proposition 3. *We have the equivalence* $w \in \texttt{Dyck}_k \iff \langle w \rangle \models \Phi_{\texttt{Dyck}_k}$*, for any word* $w \in (\Sigma_k)^+$ *of length* $n \geq 2$*. This implies* $\texttt{Dyck}_k \in \texttt{incl-ESO-IND}$.

The formula $\Phi_{\texttt{Dyck}_k}$ defining the language $\texttt{Dyck}_k$ can be used to partition the set of words on the alphabet $\Sigma_k = \{ (_i \mid 1 \leq i \leq k\} \cup \{)_i \mid 1 \leq i \leq k\}$. This partition is made according to the set of predicates that are the only ones verified by a word. Figure 13 depicts the successive steps resulting in this partition.

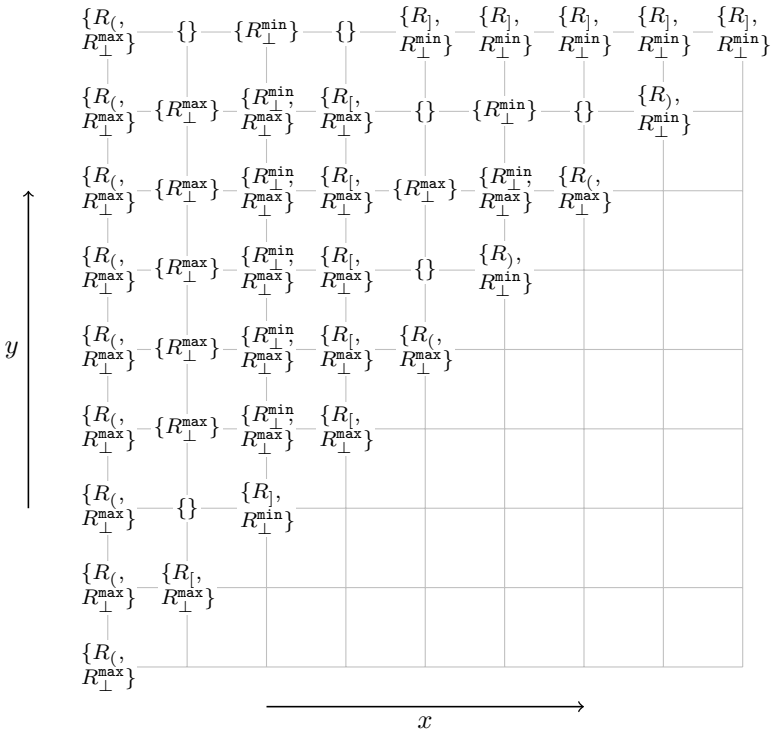
<table>
<tr><td colspan="3">not a Dyck factor</td><td>$\{R_\perp\}$</td></tr>
<tr><td rowspan="5">Dyck factor</td><td colspan="2">not prefix and not suffix</td><td>$\{R_\perp^{\texttt{min}}, R_\perp^{\texttt{max}}\}$</td></tr>
<tr><td rowspan="2">prefix and not suffix</td><td>starts with a closing parenthesis or a matched opening parenthesis</td><td>$\{R_\perp^{\texttt{max}}\}$</td></tr>
<tr><td>starts with an unmatched opening parenthesis $(_i$</td><td>$\{R_\perp^{\texttt{max}}, R_{(_i}\}$</td></tr>
<tr><td rowspan="2">suffix and not prefix</td><td>ends with an opening parenthesis or a matched closing parenthesis</td><td>$\{R_\perp^{\texttt{min}}\}$</td></tr>
<tr><td>ends with an unmatched closing parenthesis $)_i$</td><td>$\{R_\perp^{\texttt{min}}, R_{)_i}\}$</td></tr>
<tr><td colspan="2">suffix and prefix</td><td>{}</td></tr>
</table>

Figure 13: Partition obtained from the formula $\Phi_{\texttt{Dyck}_k}$

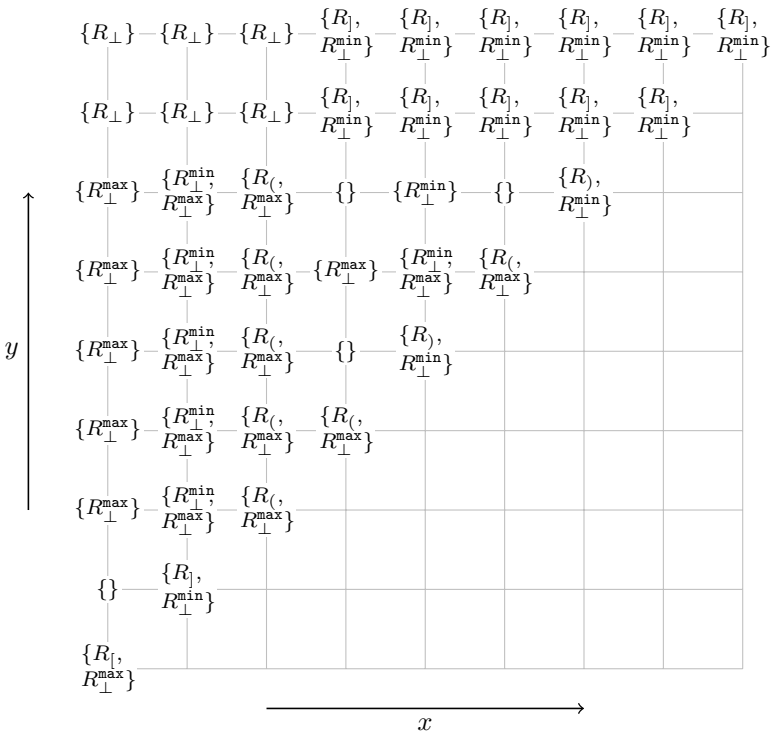
The $2k+5$ sets of predicates partitioning the set of words on the alphabet $\Sigma_k = \{ (_i \mid 1 \leq i \leq k\} \cup \{)_i \mid 1 \leq i \leq k\}$ also correspond to the $2k+5$ states of the grid circuit mimicked by $\Phi_{\texttt{Dyck}_k}$. Figure 14 shows two computation examples on the grid circuit obtained from $\Phi_{\texttt{Dyck}_2}$.

Remark: For those who are familiar with Boolean grammars, we can directly deduce from the almost normalized formula $\Phi_{\texttt{Dyck}_k}$ of $\texttt{incl-ESO-IND}$ a linear Boolean grammar describing the language $\texttt{Dyck}_k$. This grammar uses $2k+4$ nonterminal symbols corresponding to the $2k+3$ binary predicates used in ψ and the axiom D. The correspondence between binary predicates and nonterminal symbols is the following:

- $R_{(_i} \iff O_i$ for $1 \leq i \leq k$;
- $R_{)_i} \iff C_i$ for $1 \leq i \leq k$;
- $R_\perp^{\texttt{max}} \iff F_{max}$;



(a) On the input word: ([][()()]



(b) On the input word :[](()()]]

Figure 14: Computation examples of $\Phi_{\texttt{Dyck}_2}$ on $\texttt{GRID}_3$

- $R_\perp^{\texttt{min}} \iff F_{min}$;
- $R_\perp \iff F$.

Here below are the rules of the linear Boolean grammar where x and y stand for any character of Σ_k:

$$\left.\begin{array}{l} O_i \to O_i y \,\&\, \neg xC_1 \,\&\, \neg xC_2 \,\&\, \ldots \,\&\, \neg xC_k \,\&\, \neg xF \mid (_i \\ C_i \to xC_i \,\&\, \neg O_1 y \,\&\, \neg O_2 y \,\&\, \ldots \,\&\, \neg O_k y \,\&\, \neg Fy \mid)_i \end{array}\right\} \text{for } 1 \le i \le k;$$
$$D \to \neg F_{max} \,\&\, \neg F_{min} \,\&\, \neg F;$$
$$F_{max} \to O_1 \mid O_2 \mid \cdots \mid O_k \mid xF_{max} \,\&\, \neg Fy;$$
$$F_{min} \to C_1 \mid C_2 \mid \cdots \mid C_k \mid F_{min} y \,\&\, \neg xF;$$
$$F \to xF \mid Fy \mid O_i y \,\&\, xC_j \text{ for } 1 \le i, j \le k \text{ and } i \ne j.$$

A linear conjunctive grammar for $\texttt{Dyck}_k$. To each language of the partition described in Figure 13, we associate a nonterminal symbol generating this language.

- D generates the set of Dyck words: $\{\}$;
- R generates the set of Dyck factors which are neither prefix nor suffix: $\{R_\perp^{\texttt{min}}, R_\perp^{\texttt{max}}\}$;
- O_i generates the set of Dyck factors starting with an unmatched opening parenthesis $(_i$: $\{R_{(_i}\}$;
- C_i generates the set of Dyck factors ending with an unmatched closing parenthesis $)_i$: $\{R_{)_i}\}$;
- P generates the set of not suffix Dyck factors starting with a closing parenthesis or a matched opening parenthesis: $\{R_\perp^{\texttt{max}}\}$;
- S generates the set of not prefix Dyck factors ending with an opening parenthesis or a matched closing parenthesis: $\{R_\perp^{\texttt{min}}\}$;
- F generates the set of not Dyck factors: $\{R_\perp\}$.

Here below are the rules of the linear conjunctive grammar (nonterminals O and C are added for sake of clarity, and x and y stand for any character of Σ_k):

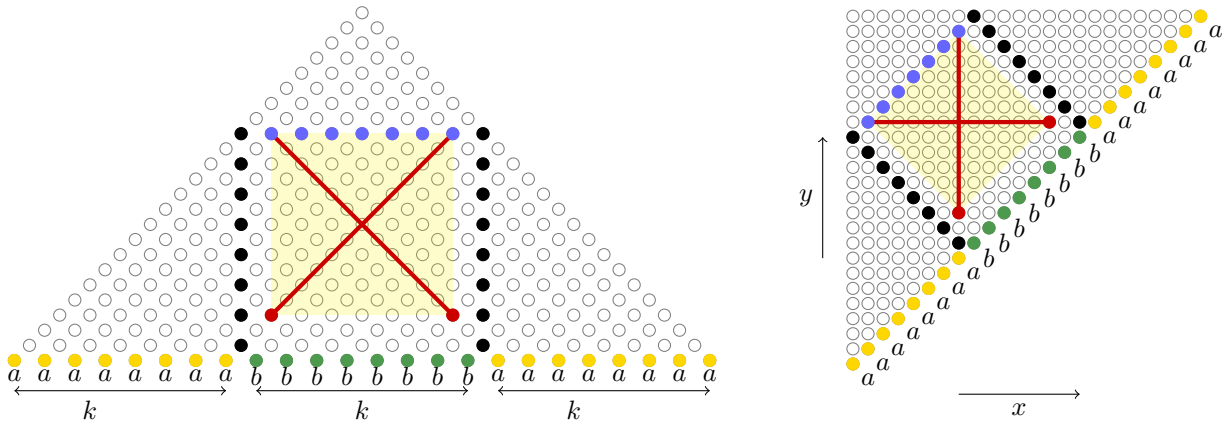
$$\left.\begin{array}{l} O_i \to O_i y \,\&\, xD \mid O_i y \,\&\, xR \mid O_i y \,\&\, xO \mid O_i y \,\&\, xP \mid O_i y \,\&\, xS \mid (_i \\ C_i \to Dy \,\&\, xC_i \mid Ry \,\&\, xC_i \mid Cy \,\&\, xC_i \mid Py \,\&\, xC_i \mid Sy \,\&\, xC_i \mid)_i \end{array}\right\} \text{for } 1 \le i \le k;$$
$$D \to O_1 y \,\&\, xC_1 \mid \cdots \mid O_k y \,\&\, xC_k \mid Py \,\&\, xS;$$
$$O \to O_1 \mid \cdots \mid O_k;$$
$$C \to C_1 \mid \cdots \mid C_k;$$
$$P \to Py \,\&\, xR \mid Py \,\&\, xO \mid Py \,\&\, xP \mid Dy \,\&\, xR \mid Dy \,\&\, xO \mid Dy \,\&\, xP;$$
$$S \to Ry \,\&\, xS \mid Cy \,\&\, xS \mid Sy \,\&\, xS \mid Ry \,\&\, xD \mid Cy \,\&\, xD \mid Sy \,\&\, xD;$$
$$R \to Ry \,\&\, xR \mid Ry \,\&\, xO \mid Ry \,\&\, xP \mid Cy \,\&\, xR \mid Sy \,\&\, xR \mid Cy \,\&\, xO \mid Sy \,\&\, xC \mid Cy \,\&\, xP \mid Sy \,\&\, xP;$$
$$F \to xF \mid Fy \mid O_i y \,\&\, xC_j \text{ for } 1 \le i, j \le k \text{ and } i \ne j.$$

8. The language of Čulik and the Firing Squad Synchronization Problem

This long section is dedicated to our most elaborate expression (programming) of a specific problem in one of our logics, here `incl-ESO-IND`. In [6], Čulik exhibited a trellis automaton that recognizes the language

$$\texttt{Culik} = \{a^i b^k a^{k-i} \mid 0 < i < k\}$$

He noticed that on a word $a^k b^k a^k$, the sites corresponding to the words $a^i b^k a^{k-i}$ with $0 < i < k$ are the sites resulting of a *synchronization* (depicted by ● in Figure 15a). The *synchronization* is a problem specific to parallel devices. On CA, it consists of all cells reaching at the same time a new state never reached. Figure 15a shows the Čulik's trellis automaton. The idea is to use a firing squad with two generals set on the sites corresponding to the factors ab^3 and b^3a. So, unless explicitly stated otherwise, k will be greater than or equal to 3 (the case $k \le 2$ will be the subject of special treatment). Figure 15b shows the $\texttt{GRID}_3$ version $3k \times 3k$ of the automaton.



(a) The trellis version

(b) The $\texttt{GRID}_3$ version

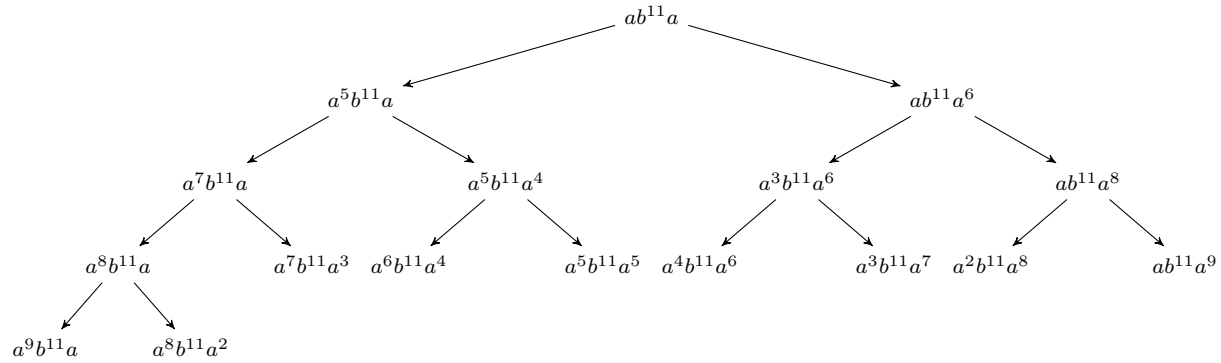
Figure 15: Both trellis and $\texttt{GRID}_3$ versions of Čulik's automaton recognizing the language $\{a^i b^k a^{k-i} \mid 0 < i < k \text{ and } k \geq 3\}$

The main idea of the recognition is a recursive process starting from the factor $ab^k a$. We call the *potential* of a factor $a^i b^k a^j$ the difference $k-i-j$ between its number of b and its number of a. In this way, the synchronized factors are the factors of potential 0. The goal is to characterize the factors with potential 1: the ones adjacent to the factors of potential 0. This is done by recursively adding to the left or to the right of a factor, a block of a's of size *half* its potential until it reaches 1. Any factor corresponding to a step of this recursive process is called a *marked factor*.

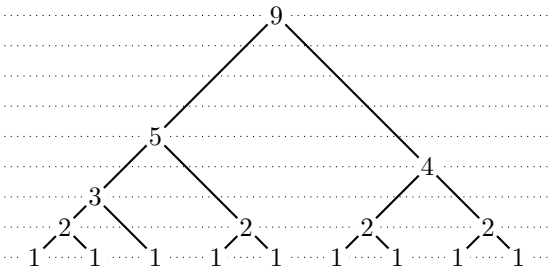
Of course, the recurrence uses a discrete division method. The *floor* operation is used when adding a block of a's to the left and the *ceiling* operation is used when adding a block of a's to the right. Thus we obtain the following recurrence scheme:

- The factor $ab^k a$ is a *marked factor*;
- if $a^i b^k a^j$ is a *marked factor* with $i, j \geq 0$ and $k-(i+j) \geq 2$ then both $a^l a^i b^n a^j$ and $a^i b^n a^j a^r$ with $l = \left\lfloor \frac{k-(i+j)}{2} \right\rfloor$ and $r = \left\lceil \frac{k-(i+j)}{2} \right\rceil$ are *marked factors*.

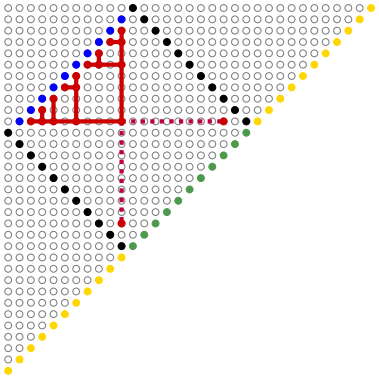
An example of the tree corresponding to this recurrence scheme is depicted in Figure 16a: the left (resp. right) son of a marked factor is the one obtained by the floor (resp. ceiling) operation. The *potential tree* is a synthetic view of the marked factors tree which only takes into account the potential of the factors. On this tree the height of a node is given by its potential. The distance between a node of potential p and its left (resp. right) son is $\left\lfloor \frac{p}{2} \right\rfloor$ (resp. $\left\lceil \frac{p}{2} \right\rceil$) since the left (resp. right) son of a node with potential p has a potential $\left\lceil \frac{p}{2} \right\rceil$ (resp. $\left\lfloor \frac{p}{2} \right\rfloor$) as depicted on Figure 16b. Figure 16c shows the embedding of the potential tree on $\texttt{GRID}_3$.



(a) The marked factors tree



(b) The potential tree



(c) Embedding of the tree on $\texttt{GRID}_3$

Figure 16: Example of the recurrence with $k = 11$

For convenience, we will call by the same name, the *potential tree*, denoted T, the "marked factors" tree, the corresponding potential tree, and its embedding in the grid.

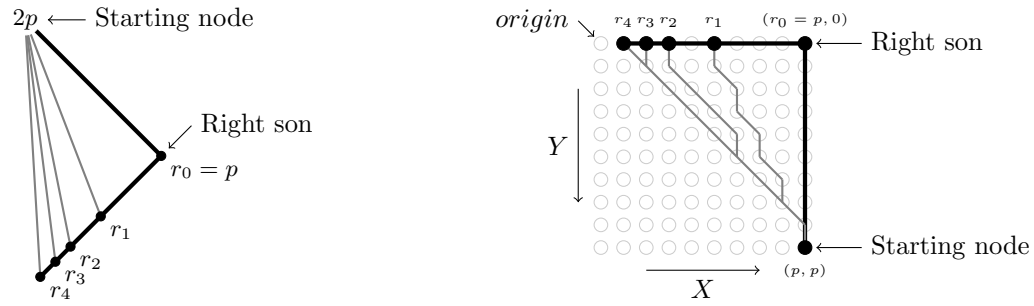
Fact. *The marked factors of the form $a^ib^ka^{k-i-1}$, for $1 \leq i \leq k-2$, correspond to the $k-2$ leaves of the potential tree: the nodes of T with potential* 1.

8.1. Construction of the potential tree

In this section we will detail the process leading to the construction of the potential tree T in $\texttt{GRID}_3$. The tree is obtained by a recursive construction of its branches. The leftmost and rightmost branches of the tree are built first. Then, from a node (starting node) we build the leftmost branch starting from its right son and the rightmost branch starting from its left son. We will denote r_0, r_1, $r_2\ldots$ the potentials of the successive nodes of a leftmost branch and l_0, l_1, $l_2\ldots$ the potentials of the successive nodes of a rightmost branch. Recall that by definition of the potential tree, we have: $r_{i+1} = \lceil \frac{r_i}{2} \rceil$ and $l_{i+1} = \lfloor \frac{l_i}{2} \rfloor$.
We present the construction of the two branches first in the case of even starting node (i.e., of even potential), secondly in the case of odd starting node that requires some adjustments to handle the asymmetry between the two branches. Then, we carry out the construction of the leftmost and rightmost branches of the potential tree. Finally, we show how to characterize the leaves of the potential tree, i.e., the marked nodes of potential 1.

A. Construction of the leftmost and rightmost branches for an even node

Construction of the leftmost branch of the right son. We now describe the construction of the leftmost branch originating from the right son of a node of even potential $2p$. It involves the starting node (the node with potential $2p$) and its right son (see Figure 17a). It makes use of a family of lines initialized from the starting node that cross the leftmost branch originated from the right son at the desired spots. The construction on the grid is shown Figure 17b. By convenience we use the following coordinate system (X, Y): the starting node is on the site (p, p) and the right son is on the site $(p, 0)$. In this way, the sites we want to characterize are of the form $(r_i, 0)$.



(a) On the tree (b) On the grid

Figure 17: Construction of the leftmost branch starting from the right son of an even node

In order to characterize the site $(r_i, 0)$, the i-th line starts from (p, p) and moves in this way: one vertical move $(0, -1)$ followed by alternations between $2^i - 1$ diagonal moves $(-1, -1)$ and a vertical move $(0, -1)$.
Let $(X_i, 0)$ be the intersection between this i-th line and the leftmost branch. Let us verify that $X_i = r_i$. Denoting α the number of times the phase composed by $2^i - 1$ diagonal moves and a vertical move is done, and $\beta < 2^i$ the number of diagonal moves left. We have the following equality:
$(X_i, 0) = (p, p) + (0, -1) + \alpha(-2^i + 1, -2^i + 1) + \alpha(0, -1) + \beta(-1, -1)$
From this equality we deduce that α is the quotient of the euclidean division of $p - 1$ by 2^i and β the remainder: $\alpha = (p-1)//2^i$ and $\beta = (p-1)\%2^i$. This gives us the following expression for X_i:

$$\begin{aligned} X_i &= p - 2^i\alpha + \alpha - \beta = 1 + 2^i\alpha + \beta - 2^i\alpha + \alpha - \beta = 1 + \alpha \\ &= 1 + (p-1)//2^i \end{aligned}$$

First, using the identity $\lceil \frac{a}{b} \rceil = 1 + \lfloor \frac{(a-1)}{b} \rfloor$ we have $X_i = \lceil \frac{p}{2^i} \rceil$. Second, recall that the potentials r_0, r_1,... of the nodes of the leftmost branch follow the recurrence: $r_0 = p$ and $r_{i+1} = \lceil \frac{r_i}{2} \rceil$. Since $\lceil \lceil \frac{a}{b} \rceil / c \rceil = \lceil \frac{a}{bc} \rceil$, we get $r_i = \lceil \frac{p}{2^i} \rceil = X_i$.

Now that we have proved the correctness of the construction of the leftmost branch using a family of lines, we will detail the local process building them. The construction of a family of lines of slopes 2^i is standard [26]. It uses two types of signals: the clock signals and the filter signals. Figure 18 depicts the construction of these signals, where the family of lines are drawn by the filter signals.

The clock signals are emitted from the right side of the square on every site $(p, p - y)$ with y even and greater than 0, and runs leftward. They are symbolized by ● on Figure 18. The filter signals alternate two phases: *cross* and *delete*. They are symbolized by ● (*cross* phase) and ● (*delete* phase) on Figure 18. One diagonal signal starts on the site $(p, p - 1)$ and goes diagonally along $(-1, -1)$. Each time a clock signal intersects with this diagonal signal, a new filter signal in *cross* phase is initialized. The diagonal signal is also used to characterize the leaf of the branch.

The clock signals and the filter signals interact with each other in order to build the desired slopes for filter signals. By default a filter signal goes diagonally and deviates from its trajectory by a unitary vertical move each time it meets a clock signal. Moreover, at each intersection, the filter signal swaps from *delete* to *cross* or conversely from *cross* to *delete*, and the clock signal goes through the intersection when the filter signal is in *cross* phase or stops when the filter signal is in *delete* phase.

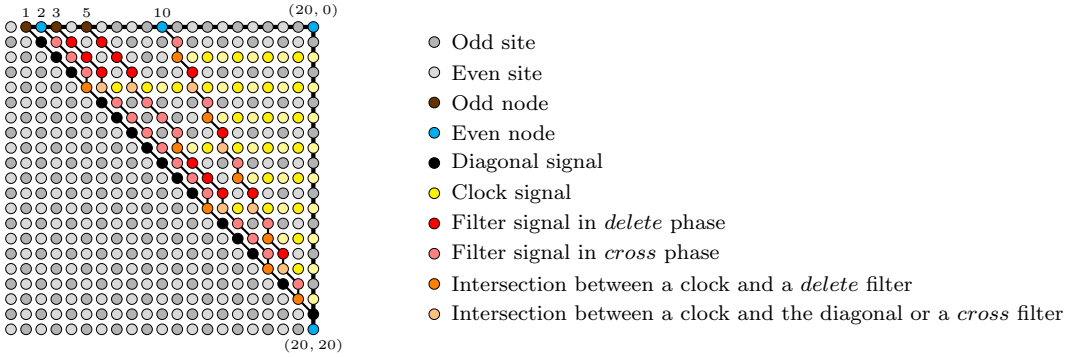
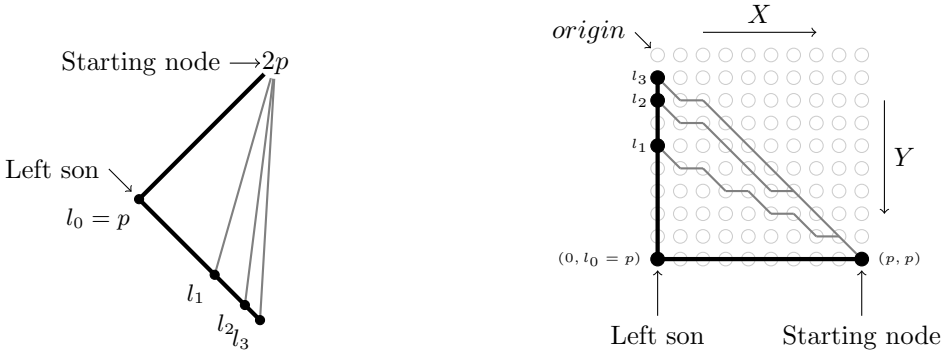


Figure 18: Construction of the leftmost branch originated from the right son of a starting node of potential $2p = 40$

Construction of the rightmost branch of the left son. As shown on Figure 19, the construction of the rightmost branch of the left son of a node of even potential $2p$ is similar to the leftmost branch case. It involves the starting node (the node with potential $2p$) and its left son. We use the same coordinate system (X, Y) as previously: the starting node is on the site (p, p) and the left son is on the site $(0, p)$. The sites we want to characterize are therefore of the form $(0, l_i)$ with $l_0 = p$ and $l_{i+1} = \left\lfloor \frac{l_i}{2} \right\rfloor$. Since $\left\lfloor \left\lfloor \frac{a}{b} \right\rfloor / c \right\rfloor = \left\lfloor \frac{a}{bc} \right\rfloor$, we get $l_i = \left\lfloor \frac{p}{2^i} \right\rfloor$ for all i.



(a) On the tree (b) On the grid

Figure 19: Construction of the rightmost branch when the starting node is even

In order to characterize the site $(0, l_i)$, the i-th line starts from (p, p) and alternates between $2^i - 1$ diagonal moves $(-1, -1)$ and one horizontal move $(-1, 0)$. Let $(0, Y_i)$ be the intersection between the i-th line and the rightmost branch. Let us verify that $Y_i = l_i$. Let α be the number of runs (composed by $2^i - 1$ diagonal moves and one horizontal move) made, and $\beta < 2^i$ be the number of diagonal moves left. We have the following equality:

$$(0, Y_i) = (p, p) + \alpha(-2^i + 1, -2^i + 1) + \alpha(0, -1) + \beta(-1, -1)$$

From this equality we deduce that α is the quotient and β the remainder of the euclidean division of p by 2^i: $\alpha = p//2^i$ and $\beta = p\%2^i$. Thus:

$$\begin{aligned} Y_i &= p - 2^i\alpha + \alpha - \beta = 2^i\alpha + \beta - 2^i\alpha + \alpha - \beta = \alpha \\ &= p//2^i = l_i \end{aligned}$$

The construction is basically the same as in the previous case: the lines are built using clock signals and filter signals. Figure 20 depicts the construction. The clock signals are emitted from the bottom side of the square on each site $(p - y, p)$ where y is odd, and runs upward. The filter signals still have two phases *cross* and *delete*. They interact with the clock signals in the same way as before but the intersection results now in an horizontal move for the filter signal. One diagonal signal emitted from the site (p, p) and moving along the vector$(-1, -1)$ is used to initialize the filter signals.

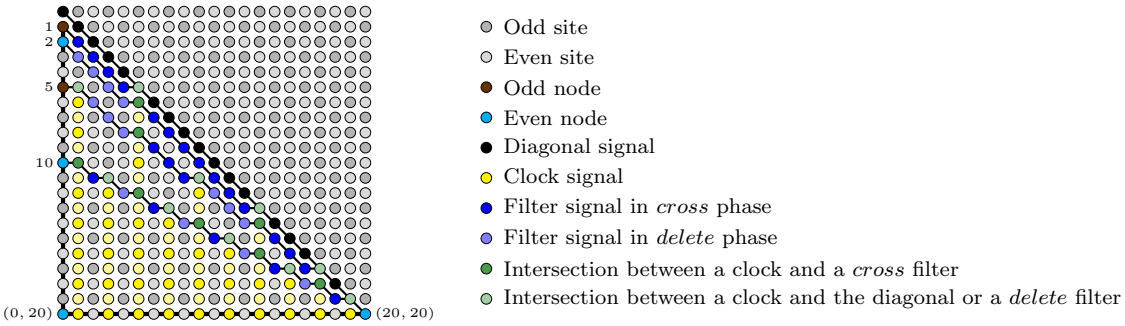
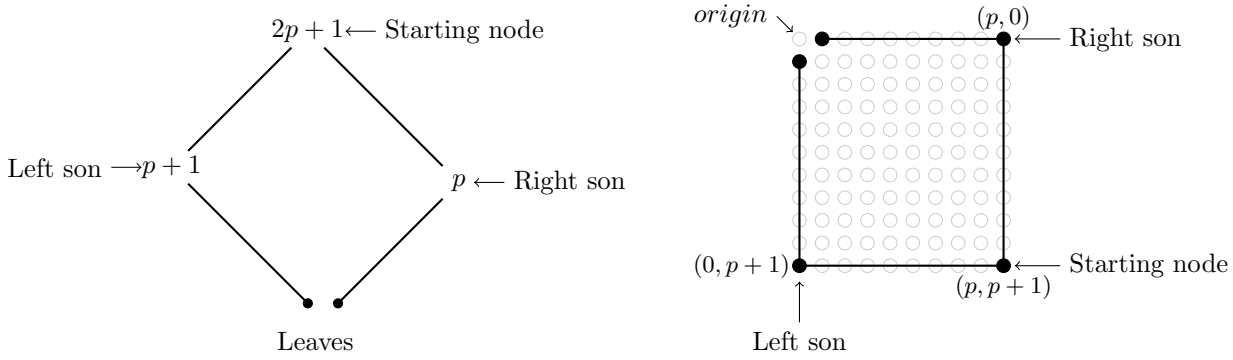


Figure 20: Construction of the rightmost branch originated from the left son of a starting node of potential $2p = 40$

B. Construction of the leftmost and rightmost branches for an odd node

When the potential of the starting node is odd (equal to $2p+1$), the space delimited by its right and left son is no more a square but a rectangle with width equal to p and height equal to $p+1$ as shown on Figure 21.



(a) On the tree (b) On the grid

Figure 21: Construction of the two branches when the starting node is odd

Construction of the leftmost branch of the right son. With r_0, r_1,... the potentials of the successive nodes of the leftmost branch starting from the right son of a node of odd potential $2p+1$: $r_0 = p$ and $r_{i+1} = \lceil \frac{r_i}{2} \rceil$. In order to characterize the sites $(r_i, 0)$ of the rectangle, we simulate the square case by doing a bigger vertical move $(0, -2)$ before starting alternations between the $2^i - 1$ diagonal moves $(-1, -1)$ and the vertical move $(0, -1)$ (see Figure 22).

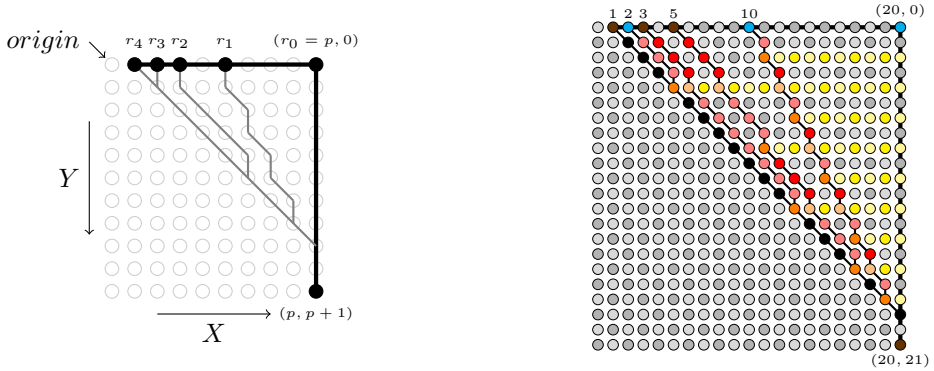


Figure 22: Construction of the leftmost branch as the top side of a rectangle

Construction of the rightmost branch of the left son. With l_0, l_1,... the potentials of the successive nodes of the rightmost branch starting from the left son of a node of odd potential $2p+1$: $l_0 = p+1$ and $l_{i+1} = \lfloor \frac{l_i}{2} \rfloor$.

In order to characterize the site $(0, l_i)$ of the rectangle, we relate to the square case by simulating a starting node on the site $(p+1, p+1)$. Starting from the site $(p, p+1)$, the line characterizing the site $(0, l_i)$ moves this way: one vertical move $(0, -1)$ followed by $2^i - 2$ diagonal moves $(-1, -1)$ and then alternations between an horizontal move $(-1, 0)$ and $2^i - 1$ diagonal moves $(-1, -1)$. The Figure 23 depicts this construction in the rectangle.

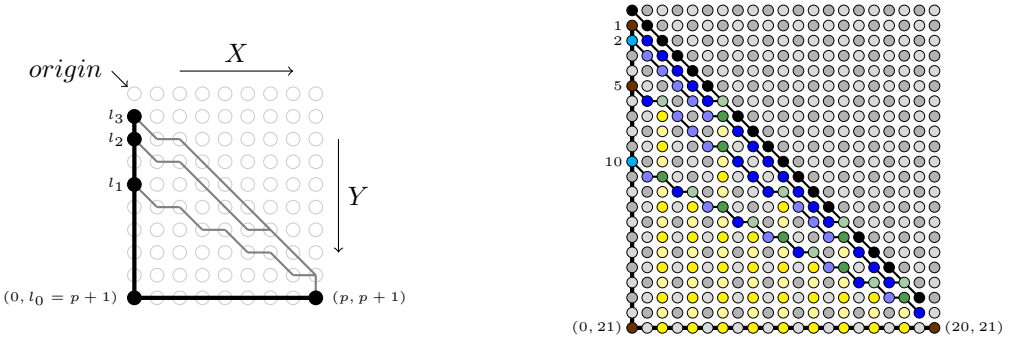


Figure 23: Construction of the rightmost branch as the left side of a rectangle

C. Initialization of the recurrence: construction of the leftmost and rightmost branches of the tree.

The construction of the leftmost (resp. the rightmost) branch of the tree starts from the site corresponding to the factor $abbb$ (resp. $bbba$). In this way, the distance between this starting site and the site corresponding to the factor $ab^k a$ (the root of the tree) is $k-2$ which is the potential of this factor (see Figure 24). Therefore we can use the same family of lines that in the even potential case with the left (resp. right) general as the starting node and the root $ab^k a$ as its right (resp. left) son.

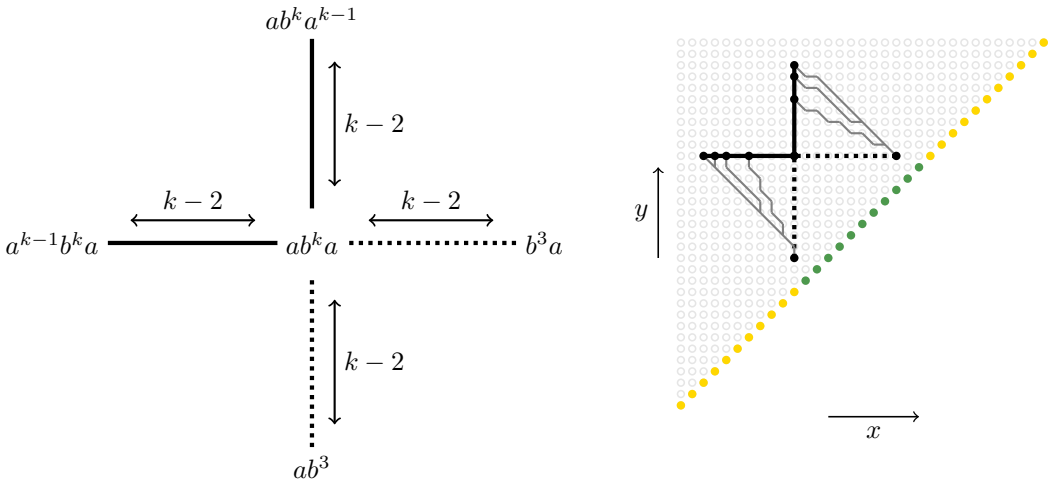


Figure 24: Construction of the rightmost and leftmost branches of the tree

D. The end of the recurrence: characterization of the leaves

In order to end the recursive building process of the potential tree, we need to locally characterize its leaves. With the usual coordinate system on the grid, this characterization is given by the following lemma.

Lemma 12 (leaf characterization)**.** *Suppose the potential tree T has at least 2 nodes. Then, a node (x, y) of T is a leaf* iff *either one of the following conditions holds:*

(1) The potential of the node (x, y) is odd and the site $(x+1, y)$ is a node of T;

(2) The site $(x, y-1)$ is a node of T;

(3) The potential of the node (x, y) is odd and the site $(x, y-2)$ is a node of T.

Proof. Let p denote the potential of the node (x, y). Then, the potentials of the sites $(x + 1, y)$, $(x, y - 1)$ and $(x, y - 2)$ are $p + 1$, $p + 1$, and $p + 2$, respectively.

If the node (x, y) is a leaf then its potential p is 1. In case of (x, y) is a left leaf, its parent node is of potential q that verifies $\lceil \frac{q}{2} \rceil = 1$. Then $q = 2$ and the parent node is $(x + 1, y)$. In case of (x, y) is a right leaf, the parent node is of potential q that verifies $\lfloor \frac{q}{2} \rfloor = 1$. Then $q \in \{2, 3\}$ and its parent node is either $(x, y - 1)$ or $(x, y - 2)$.

Conversly, if condition (1) holds, then the potential $p + 1$ of the node $(x, y - 1)$ is even. So the potential of its left son (x, y) satisfies $p = \frac{p+1}{2}$. This implies $p = 1$ and (x, y) is a leaf. If condition (2) holds, then the potential of its right descendant (x, y) satisfies $p = \lfloor \frac{p+1}{2} \rfloor$. This implies $p = 1$ and (x, y) is a leaf. If condition (3) holds, then the potential $p + 2$ of the node $(x, y - 2)$ is odd. So the potential of its right descendant (x, y) satisfies $p = \lfloor \frac{p+2}{2^i} \rfloor = \frac{p+1}{2^i}$ for some integer i. This implies $i = 1$, $p = 1$ and (x, y) is a leaf. □

8.2. An inclusion inductive formula defining the language `Culik`

The previous section describes the characterization of all factors belonging to the language $\{a^i b^k a^{k-i} \mid 0 < i < k$ `and` $k \geq 3\}$ inside the grid $3k \times 3k$ with the word $a^k b^k a^k$ as the input word. In this section we detail the clauses expressing that a word $w \in \{a, b\}^+$ belongs or not to this language. For this purpose, we consider the $\texttt{GRID}_3$ $n \times n$ with the input word $w = w_1 \dots w_n \in \{a, b\}^+$ written on the diagonal $y = x$.

For a lighter presentation of the formulas, we will simplify the presentation of inclusion clauses by adopting the following conventions: $R(x+a, y-b)$ is written in place of $x+a \leq y-b \wedge R(x+a, y-b)$. In the same spirit, we write $x = 1 \wedge y = n$ instead of $x \leq y \wedge x = 1 \wedge y = n$.

Structure of the logical definition of `Culik` *within the language* $a^+b^+ba^+$. In order to define `Culik` inside the language $a^+b^+ba^+$, we will use 20 binary predicates:

- 8 initial global predicates: parity predicates `Even` and `Odd`; border predicates (between a and bbb or between bbb and a) `LeftBorder` (resp. `RightBorder`) only true on the last (resp. the first) a of the first (resp. second) sequence of a of the input word belonging to $a^+b^+a^+$; generals `LeftGeneral` and `RightGeneral` and external branches `StartLeftBranch` and `StartRightBranch` (this predicates only have a meaning if the number of b is greater or equal to 3 i.e. for an input word belonging to $a^+b^+bba^+$);
- 4 recursive global predicates: nodes `Node`, leaves `Leaf` and branches `LeftBranch` and `RightBranch`.
- 8 local predicates labeled by either `Up` or `Low`: diagonal predicates `UpDiag` and `LowDiag`, clock predicates `UpClock` and `LowClock`, filter predicates `UpCross`, `UpDelete`, `LowCross` and `LowDelete`.

Convention: We add the two general sites ab^3 (left general) and b^3a (right general) as nodes of the tree T. The tree is no more an usual rooted tree, i.e. with only one root. We have now three initial nodes: ab^3, b^3a and ab^ka with $k \geq 3$. For convenience in the logical construction, we admit that the "tree" has two root ab^3 and b^3a and that the node ab^ka is both the right son of the node ab^3 and the left son of the node b^3a.

Clauses defining the initialization and the end of the recurrence

The parity predicates: `Even` *and* `Odd`. During the inductive construction of the potential tree, we need to know for each node the parity of its potential. The potential of a factor being the difference between its number of b and a, the parity of its potential is the same as the parity of its length. In this way, the parity of the site (x, y) is the parity of $y - x + 1$ and in particular the parity of diagonal sites is odd.

The following clauses define inductively the parity of each site (x, y) with $x \leq y$ from the diagonal:

- $x = y \rightarrow \texttt{Odd}(x, y)$; $x < y \wedge \texttt{Odd}(x, y - 1) \rightarrow \texttt{Even}(x, y)$;
  $x < y \wedge \texttt{Even}(x, y - 1) \rightarrow \texttt{Odd}(x, y)$.

The border predicates and the two generals: `LeftBorder`, `RightBorder`, `LeftGeneral` *and* `RightGeneral`. The left (resp. right) border is defined on the diagonal $x = y$ as the last (resp. first) a in the first (resp. second) sequence of a of a word belonging to $a^+b^+bba^+$. By means of the left (resp. right) borders we characterize the left (resp. right) general site corresponding to the factor ab^3 (resp. b^3a).

- $x = y \wedge Q_a(x) \wedge Q_b(x+1) \wedge Q_b(x+2) \wedge Q_b(x+3) \rightarrow \texttt{LeftBorder}(x, y)$;
- $x = y \wedge Q_b(x-3) \wedge Q_b(x-2) \wedge Q_b(x-1) \wedge Q_a(x) \rightarrow \texttt{RightBorder}(x, y)$;
- $\texttt{LeftBorder}(x, y-3) \rightarrow \texttt{LeftGeneral}(x, y)$; $\texttt{RightBorder}(x+3, y) \rightarrow \texttt{RightGeneral}(x, y)$.

The two external branches: `StartLeftBranch` *and* `StartRightBranch`. The ray defining the external right (resp. left) branch is emitted vertically (resp. horizontally) from the left (resp. right) general.

- $\texttt{LeftGeneral}(x, y) \rightarrow \texttt{StartRightBranch}(x, y)$;
  $\texttt{StartRightBranch}(x, y-1) \rightarrow \texttt{StartRightBranch}(x, y)$;
- $\texttt{RightGeneral}(x, y) \rightarrow \texttt{StartLeftBranch}(x, y)$;
  $\texttt{StartLeftBranch}(x+1, y) \rightarrow \texttt{StartLeftBranch}(x, y)$.

The first three nodes: `Node`. The left general and the right general are nodes of the tree:

- $\texttt{LeftGeneral}(x, y) \rightarrow \texttt{Node}(x, y)$; $\texttt{RightGeneral}(x, y) \rightarrow \texttt{Node}(x, y)$

The intersection between the two external branches marks also one node of the tree:

- $\texttt{StartLeftBranch}(x, y) \wedge \texttt{StartRightBranch}(x, y) \rightarrow \texttt{Node}(x, y)$

Remark: The two generals are not empty singletons sets only if the sites ab^3 and b^3a exist. Therefore, the three initial nodes are created if and only if ab^3 and b^3a are factors of the input word.

The branches predicates: `RightBranch` *and* `LeftBranch`. From each node of the tree start an horizontal leftward ray symbolizing its left branch and a vertical upward ray symbolizing its right branch.

- $\texttt{Node}(x, y) \rightarrow \texttt{LeftBranch}(x, y)$; $\texttt{LeftBranch}(x+1, y) \rightarrow \texttt{LeftBranch}(x, y)$;
- $\texttt{Node}(x, y) \rightarrow \texttt{RightBranch}(x, y)$; $\texttt{RightBranch}(x, y-1) \rightarrow \texttt{RightBranch}(x, y)$.

The leaf predicate: `Leaf`. According to Lemma 12, a node (x, y) is a leaf of T *iff* either one of the following conditions holds:
The potential of the node (x, y) is odd and the site $(x+1, y)$ is a node of T:

- $\texttt{Node}(x, y) \wedge \texttt{Odd}(x, y) \wedge \texttt{Node}(x+1, y) \rightarrow \texttt{Leaf}(x, y)$;

The site $(x, y-1)$ is a node of T:

- $\texttt{Node}(x, y) \wedge \texttt{Node}(x, y-1) \rightarrow \texttt{Leaf}(x, y)$;

The potential of the node (x, y) is odd and the site $(x, y-2)$ is a node of T:

- $\texttt{Node}(x, y) \wedge \texttt{Odd}(x, y) \wedge \texttt{Node}(x, y-2) \rightarrow \texttt{Leaf}(x, y)$.

Remark: When $w \in a^+b^3a^+$ the first (or second) clause above defines the only leaf of the tree.

Clauses defining the nodes of the potential tree
A. Even case, the nodes of the left branches (upper triangles).

The upper diagonal predicate: `UpDiag`. When the site (x, y) is an even node, an upper diagonal is initialized on the site $(x, y+1)$.

- $\texttt{Node}(x, y-1) \wedge \texttt{Even}(x, y-1) \rightarrow \texttt{UpDiag}(x, y)$

Once initialized, the diagonal follows a trajectory along the vector $(-1, 1)$ until it crosses a left branch, ensuring that no node is created outside the working space of each starting node.

- $\texttt{UpDiag}(x+1, y-1) \wedge \neg\texttt{LeftBranch}(x+1, y-1) \rightarrow \texttt{UpDiag}(x, y)$.

The upper clock: `UpClock`. The first tick of the upper clock begins two sites above an even starting node. In other terms, a site (x, y) will be the start of the first tick of the upper clock if and only if the site $(x, y-2)$ is an even node:

- $\texttt{Node}(x, y-2) \wedge \texttt{Even}(x, y-2) \rightarrow \texttt{UpClock}(x, y)$

Once the first tick of the clock emitted, a new tick is initialized every two sites of the right branch of the starting node until its right son is reached:

- $\texttt{RightBranch}(x, y) \wedge \texttt{UpClock}(x, y-2) \wedge \neg\texttt{Node}(x, y-1) \wedge \neg\texttt{Node}(x, y-2) \rightarrow \texttt{UpClock}(x, y)$

An upper clock signal runs leftward until it crosses a filter signal in *delete* phase or the upper diagonal:

- $\texttt{UpClock}(x+1, y) \wedge \neg\texttt{UpDelete}(x+1, y) \wedge \neg\texttt{UpDiag}(x+1, y) \rightarrow \texttt{UpClock}(x, y)$

The upper filters: `UpCross` *and* `UpDelete`. The two phases *cross* and *delete* of an upper filter signal are symbolized by the predicates `UpCross` and `UpDelete`. An upper filter signal is initialized in *cross* phase on the site just above every intersection between an upper clock and the upper diagonal, except when the intersection takes place on the left branch:

- $\texttt{UpDiag}(x, y-1) \wedge \texttt{UpClock}(x, y-1) \wedge \neg\texttt{LeftBranch}(x, y-1) \rightarrow \texttt{UpCross}(x, y)$

By default, the filter signals move diagonally along the vector $(-1, 1)$ without changing phase until they cross an upper clock signal or the left branch:

- $\texttt{UpCross}(x+1, y-1) \wedge \neg\texttt{UpClock}(x+1, y-1) \wedge \neg\texttt{LeftBranch}(x+1, y-1) \rightarrow \texttt{UpCross}(x, y)$;
  $\texttt{UpDelete}(x+1, y-1) \wedge \neg\texttt{UpClock}(x+1, y-1) \wedge \neg\texttt{LeftBranch}(x+1, y-1) \rightarrow \texttt{UpDelete}(x, y)$

The filter signals change phase when they cross an upper clock signal, except when the intersection takes place on the left branch:

- $\texttt{UpCross}(x, y-1) \wedge \texttt{UpClock}(x, y-1) \wedge \neg\texttt{LeftBranch}(x, y-1) \rightarrow \texttt{UpDelete}(x, y)$;
  $\texttt{UpDelete}(x, y-1) \wedge \texttt{UpClock}(x, y-1) \wedge \neg\texttt{LeftBranch}(x, y-1) \rightarrow \texttt{UpCross}(x, y)$

The nodes of left branches in the upper triangles: `Node`. A new node is created on each intersection between the left branch and, the upper diagonal or a filter signal in either phase:

- $\texttt{UpDiag}(x, y) \wedge \texttt{LeftBranch}(x, y) \rightarrow \texttt{Node}(x, y)$;
- $\texttt{UpCross}(x, y) \wedge \texttt{LeftBranch}(x, y) \rightarrow \texttt{Node}(x, y)$;
- $\texttt{UpDelete}(x, y) \wedge \texttt{LeftBranch}(x, y) \rightarrow \texttt{Node}(x, y)$.

Remark: If ab^3 and b^3a are not factors of the input word, all the predicates above are empty (except `Odd` and `Even`) and no nodes (and therefore no leaves) are created.

B. Even case, the nodes of the right branches (lower triangles).

The lower diagonal: `LowDiag`. When the site (x, y) is an even node, a lower diagonal is initialized on the site $(x-1, y+1)$:

- $\texttt{Node}(x+1, y-1) \wedge \texttt{Even}(x+1, y-1) \rightarrow \texttt{LowDiag}(x, y)$

Like the upper diagonal the lower diagonal follows a trajectory along the vector $(-1, 1)$, this diagonal stops when it crosses a right branch:

- $\texttt{LowDiag}(x+1, y-1) \wedge \neg\texttt{RightBranch}(x+1, y-1) \rightarrow \texttt{LowDiag}(x, y)$

The lower clock: `LowClock`. When the starting node is even, the first tick of the lower clock is always initialized on the site directly on the left of this starting node:

- $\texttt{Node}(x+1, y) \wedge \texttt{Even}(x+1, y) \rightarrow \texttt{LowClock}(x, y)$

As for the upper clock, a new tick of the lower clock is initialized every two sites of the left branch of the starting node until its left son is reached:

- $\texttt{LeftBranch}(x, y) \wedge \texttt{LowClock}(x+2, y) \wedge \neg\texttt{Node}(x+1, y) \wedge \neg\texttt{Node}(x+2, y) \rightarrow \texttt{LowClock}(x, y)$.

A lower clock signal runs upward until it crosses a filter signal in *delete* phase or the lower diagonal:

- $\texttt{LowClock}(x, y-1) \wedge \neg\texttt{LowDiag}(x, y-1) \wedge \neg\texttt{LowDelete}(x, y-1) \rightarrow \texttt{LowClock}(x, y)$

The lower filters: `LowCross` *and* `LowDelete`. The construction of the lower filter signals is similar to the upper one. A filter signal in *cross* phase is initialized on the site just to the left of an intersection between the lower clock signal and the lower diagonal only if this intersection is not on the right branch:

- $\texttt{LowDiag}(x+1,y) \wedge \texttt{LowClock}(x+1,y) \wedge \neg\texttt{RightBranch}(x+1,y) \rightarrow \texttt{LowCross}(x,y)$.

Then it moves diagonally and alternates between the *cross* and *delete* phases:

- $\texttt{LowCross}(x+1,y-1) \wedge \neg\texttt{LowClock}(x+1,y-1) \wedge \neg\texttt{RightBranch}(x+1,y-1) \rightarrow \texttt{LowCross}(x,y)$;
  $\texttt{LowDelete}(x+1,y-1) \wedge \neg\texttt{LowClock}(x+1,y-1) \wedge \neg\texttt{RightBranch}(x+1,y-1) \rightarrow \texttt{LowDelete}(x,y)$;

- $\texttt{LowCross}(x+1,y) \wedge \texttt{LowClock}(x+1,y) \wedge \neg\texttt{RightBranch}(x+1,y) \rightarrow \texttt{LowDelete}(x,y)$;
  $\texttt{LowDelete}(x+1,y) \wedge \texttt{LowClock}(x+1,y) \wedge \neg\texttt{RightBranch}(x+1,y) \rightarrow \texttt{LowCross}(x,y)$.

The nodes of right branches in the lower triangles: `Node`. Contrary to the left branch case, the intersection between the lower diagonal and the right branch does not create a new node. Therefore there are only two clauses defining the creation of a node at each intersection between a filter signal and the right branch.

- $\texttt{LowCross}(x,y) \wedge \texttt{RightBranch}(x,y) \rightarrow \texttt{Node}(x,y)$;

- $\texttt{LowDelete}(x,y) \wedge \texttt{RightBranch}(x,y) \rightarrow \texttt{Node}(x,y)$.

C. Odd case, the nodes of both left and right branches (upper and lower triangles) . In this section we detail the construction of nodes on both right and left branches when the starting node is odd. Since we do not want leaves to create new nodes, we will use the conjunction $\texttt{Node}(\delta) \wedge \texttt{Odd}(\delta) \wedge \neg\texttt{Leaf}(\delta)$ to refer to odd nodes which are not leaves.

Initialization of diagonals: `UpDiag` *and* `LowDiag`. The diagonal behavior of both upper and lower diagonals stays the same when the starting node is odd. The only change is their initialization: both diagonals are initialized a site higher than in the even starting node case:

- $\texttt{Node}(x,y-2) \wedge \texttt{Odd}(x,y-2) \wedge \neg\texttt{Leaf}(x,y-2) \rightarrow \texttt{UpDiag}(x,y)$

- $\texttt{Node}(x+1,y-2) \wedge \texttt{Odd}(x+1,y-2) \wedge \neg\texttt{Leaf}(x+1,y-2) \rightarrow \texttt{LowDiag}(x,y)$

Initialization of clocks: `UpClock`, `LowClock` *and* `LowCross`. The general behavior of upper clocks and lower clocks does not change when the starting node is odd. The initialization change of the upper clock is the same as the upper diagonal, the first tick of the upper clock begins one site higher than in the even starting node case.

- $\texttt{Node}(x,y-3) \wedge \texttt{Odd}(x,y-3) \wedge \neg\texttt{Leaf}(x,y-3) \rightarrow \texttt{UpClock}(x,y)$

The initialization change of the lower clock is a bit different, in order to simulate a first clock tick on the starting node itself, the first clock tick of the lower diagonal begins two sites to the left of the starting node and a lower filter signal in *cross* phase is initialized on the site just to the up left of the starting node.

- $\texttt{Node}(x+2,y) \wedge \texttt{Odd}(x+2,y) \wedge \neg\texttt{Leaf}(x+2,y) \rightarrow \texttt{LowClock}(x,y)$

- $\texttt{Node}(x+1,y-1) \wedge \texttt{Odd}(x+1,y-1) \wedge \neg\texttt{Leaf}(x+1,y-1) \rightarrow \texttt{LowCross}(x,y)$

To recap, after this long sequence, we have all the clauses defining the potential tree and its leaves. Next, from the leaves, we will characterize the language `Culik`.

Clause defining the language `Culik` *within the language* $a^+b^+ba^+$

Lemma 13. *Let w be a word of $a^+b^+ba^+$. Then, $w \in$ `Culik` $\iff \langle w\rangle \models n \leq 4 \vee$ `Leaf`$(2,n) \vee$ `Leaf`$(1,n-1)$*

Proof. If w is of size $n \leq 4$, then $w = abba$ is both the only word of $a^+b^+ba^+$ and the only word of `Culik`'s language.

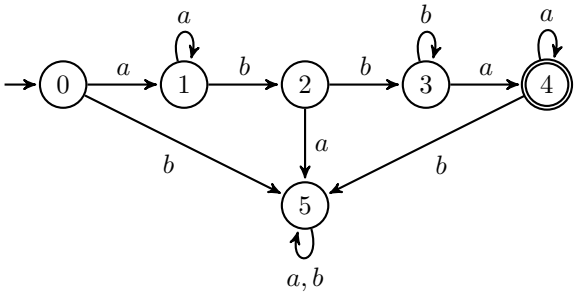
If w is of size $n > 4$, then

- either w is of the form $a^ib^2a^j$, for $i + j > 2$, and therefore w is not a word of `Culik` and does not generate any nodes or leaves because ab^3 and b^3a are not factors of w,
- or w is of the form $a^ib^ka^j$, for $k \geq 3$ and $i, j \geq 1$, and we have the equivalence: $w \in$ `Culik` $\iff \langle w\rangle \models$ `Leaf`$(2,n) \vee$ `Leaf`$(1,n-1)$.

□

It remains to prove that $a^+b^+ba^+$ belongs to `incl-ESO-IND` (actually, to `incl-ESO-HORN`).

Clauses defining the language $a^+b^+ba^+$. Essentially, the clauses mimic the finite automaton $\mathcal{A}$ recognizing the regular expression $a^+b^+ba^+$



$\Sigma = \{a, b\}$ the input alphabet
$\mathtt{S} = \{0, 1, 2, 3, 4, 5\}$ the states set
0 the start state
$F = \{4\}$ the accept states set
$\delta : \mathtt{S} \times \Sigma \to \mathtt{S}$ the transition function given by its diagram

Figure 25: The automaton $\mathcal{A} = (\Sigma, \mathtt{S}, 0, F, \delta)$ that accepts $a^+b^+ba^+$

Predicates that transport the input through the grid: T_s for $s \in \Sigma$.

- $x = y \wedge Q_s(x) \to T_s(x, y)$; $T_s(x + 1, y) \to T_s(x, y)$, for $s \in \{a, b\}$.

Predicates that simulate the automaton $\mathcal{A} = (\Sigma, \mathtt{S}, 0, F, \delta)$: R_q for $q \in \mathtt{S}$.

- $x = 1 \wedge x = y \wedge Q_s(x) \to R_{\delta(0,s)}(x, y)$, for $s \in \Sigma$.
- $R_q(x, y - 1) \wedge T_s(x + 1, y) \to R_{\delta(q,s)}(x, y)$, for $(q, s) \in \mathtt{S} \times \Sigma$
- $x = 1 \wedge y = n \wedge R_q(x, y) \to \bot$, for $q \in \mathtt{S} \setminus F$.

The conjunction of these clauses ensures that the input word belongs to $a^+b^+ba^+$.

Clause defining the language `Culik`

Let ψ denote the conjunction of the previous clauses and $\sigma_{\texttt{Culik}}$ denote its set of computation predicates {`Even`, `Odd`, `LeftBorder`, `RightBorder`, `LeftGeneral`, `RightGeneral`, `StartLeftBranch`, `StartRightBranch`, `Node`, `Leaf`, `LeftBranch`, `RightBranch`, `UpDiag`, `LowDiag`, `UpClock`, `LowClock`, `UpCross`, `UpDelete`, `LowCross`, `LowDelete`, $T_a, T_b, R_0, R_1, R_2, R_3, R_4, R_5$}. Let $\Phi_{\texttt{Culik}}$ denote the formula $\exists \sigma_{\texttt{Culik}}\, \forall x, y\ \psi$. By construction, $\Phi_{\texttt{Culik}}$ belongs to `incl-ESO-IND`. From Lemma 13 and the previous characterization of $a^+b^+ba^+$, we deduce the following proposition:

Proposition 4. *For any word $w \in \{a, b\}^+$, we have the equivalence: $w \in$ `Culik` iff $\langle w\rangle \models \Phi_{\texttt{Culik}}$. This implies* `Culik` $\in$ `incl-ESO-IND` *and* `Culik` $\in$ `Trellis` *[6].*

9. Conclusion

We believe that this paper contributes to the knowledge of cellular automata, their complexity classes and the design of their programs in two ways:

Descriptive complexity: We establish the first logical characterizations of the classical real-time complexity classes of cellular automata.

Programming: We give a methodology for programming problems on cellular automata from the inductive logical definitions of those problems.

Regarding this last point, we recall that it is difficult to design programs for parallel computers. We lack general tools to design them. Our programming methodology seems to be the first general method for a parallel and local computation model such as the cellular automaton.

Descriptive versus computational complexity of cellular automata

It was known that the three complexity classes studied in this paper, $\texttt{RealTime}_{\texttt{CA}}$, $\texttt{RealTime}_{\texttt{IA}}$ and `Trellis`, are the only distinct and natural complexity classes for minimal time, so-called real-time, of one-dimensional cellular automata. In various articles from the 1960s to 2000s, it has been established that each of those classes is robust, in particular with respect to the chosen neighborhood [31], and has several equivalent characterizations in other frameworks: e.g, `Trellis` is the class of linear conjunctive languages [28] and also the class of linear Boolean languages [29].

In this paper, we have presented a unified view of the three real-time classes as part of descriptive complexity. More precisely, we have introduced three logics, `pred-ESO-HORN`, `pred-dio-ESO-HORN` and `incl-ESO-HORN`, which are essentially Horn logics with two first-order variables, applied to a square grid structure, i.e., a structure using the successor and predecessor functions and the minimum and maximum predicates, as its only built-in objects. We have defined the syntax of each of our logics as large (flexible) as possible so that it can express problems in the most natural way. At the same time, a normal form of each logic has been defined so that each normalized formula "mimics" a grid automaton. For each logic, we have given a general algorithm which transforms a formula into its equivalent normal form. Finally, we have proved that those three "grid" logics (or grid automata) whose only difference is the placement of the input word on the square grid, on a side which contains the output vertex, or on the diagonal that contains or does not contain the output vertex, exactly characterize $\texttt{RealTime}_{\texttt{CA}}$, $\texttt{RealTime}_{\texttt{IA}}$ and `Trellis`, respectively. In this paper, the successive steps of the normalization algorithms of the logics and the equivalences of these logics with the real-time complexity classes are fully described and proved while they are sometimes more sketchy in [18].

Logic as a programming language of cellular automata

We believe that the main contribution of this long paper compared to its conference version [18] is the following:

- We describe a general methodology for programming problems on cellular automata from the inductive logical definitions of these problems;
- The method is applied to a number of specific problems.

For this purpose, we have extended the *Horn logics* introduced in [18] as (now called) *inductive logics* by allowing limited use of *negation* on hypothesis computation atoms (it is comparable to *Stratified Datalog* which extends *Datalog*). Note two essential points: our inductive logics characterize exactly the *same* real-time complexity classes as the Horn logics; they still have normal forms that mimic grid automata. Regarding our method, the reader may ask the question: What is the real interest of logic to program cellular automata?

The interest of our programming method is evidenced in this paper by the application that we make of it for a representative choice of classical problems (seven examples) that we explicitly express in our logics (Sections 3, 7 and 8) and translate in normal form to illustrate the normalization method for a specific problem (Subsection 3.2). Let us now argue for our logic programming method.

First, in a general setting, logic programming has proven to be a useful paradigm to program several tasks, for example, compiling or database queries, as evidenced in the languages *Prolog* and

Datalog. It allows to define concepts (represented by predicates) inductively, as we have shown here, e.g., for the problems `Palindrome`, `Unbordered` and `Product`.

Second, note that the programming method most used in the literature to program cellular automata is the geometric method of signals and collisions [2, 12, 5, 33, 10, 26, 6, 8, 9]. We think that our logical description of signals (by predicates) and of their collisions and transformations (by implication clauses often using negation) is both more intuitive (more readable) and more precise (more formal) than their direct encoding by states of the final automaton that literature only sketches in general. Logic both is flexible – it is a high-level programming language – and allows for formal proofs as we have shown in the seven examples that we have studied in detail. This is especially important for the most elaborate problems: the multi-parenthesis language $\texttt{Dyck}_k$ recognition (Subsection 7.3) and overall the `Culik` language recognition (Section 8), which involves synchronization by the "divide and conquer" method.

Note that logic is a more synthetic language to describe a problem than the cellular automaton that decides it. Indeed, it is shown in Section 6 that *one* state of the automaton is translated into *one* predicate of the corresponding formula, so that each transition rule of the automaton is expressed by one Horn clause; conversely, a logical formula is converted to an automaton whose *set* of states is the *power set* of the set of predicates of the formula, so that the size of the automaton is *exponential* in the size of the initial formula.

In summary, logic is a nice intermediate language: it links the geometry of signals to cellular automata, while preserving complexity.

Solved questions versus open problems

It seems that the main questions concerning the mutual relationships between the three *real-time* complexity classes $\texttt{RealTime}_{\texttt{CA}}$, $\texttt{RealTime}_{\texttt{IA}}$ and `Trellis` of *one-dimensional* cellular automata have been solved and their descriptive complexity is determined by this paper. The main open problem that remains is the old and difficult question of whether the inclusion $\texttt{RealTime}_{\texttt{CA}} \subseteq \texttt{DLIN}^1_{\texttt{CA}}$ is strict, where $\texttt{DLIN}^1_{\texttt{CA}}$ is the class of languages recognized by *one-dimensional* cellular automata in *linear time*.

What about the computational complexity and descriptive complexity of cellular automata of *dimension 2 or more*? Regarding *linear time*, let us recall the equality $\texttt{DLIN}^d_{\texttt{CA}} = \texttt{monot-ESO-HORN}^d(\forall^{d+1}, \texttt{arity}^{d+1})$, for each dimension $d \geq 1$. This result proved in [3] establishes the descriptive complexity of the class $\texttt{DLIN}^d_{\texttt{CA}}$ of d-dimensional picture languages recognized in linear time on d-dimensional cellular automata and confirms the robustness of this class: for example, the definition of the class $\texttt{DLIN}^d_{\texttt{CA}}$ does not depend on the neighborhood chosen for the d-dimensional cellular automata provided it is *complete* [17], i.e., allows to reach by iteration all the points of $\mathbb{Z}^d$.

How do the results of this paper extend to the *real-time* complexity classes of cellular automata of dimension 2 or more? Considering that the normalized versions of our logics, identified to square grid circuits, offer a new view of the real-time complexity classes of dimension 1, it is natural to ask the following questions: What are the complexity classes that correspond to *cubic* grid circuits? How robust are those classes?

They seem difficult questions in case the input is a 2-dimensional picture. Indeed, a number of papers [35, 37, 15, 16] have shown that real-time complexity of 2-dimensional languages is very sensitive to the neighborhood chosen. This contrasts with the robustness of linear time complexity in any dimension and of real-time complexity classes for dimension 1.

Therefore, a more promising line of research would be to study, starting from the results of [38], the descriptive complexity of the classes of languages (sets of words) decided in real-time on a cubic grid (or, more generally, on a grid of dimension $d \geq 3$) in comparison to real-time on multi-dimensional cellular automata.

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] A. J. Atrubin. A one-dimensional real-time iterative multiplier. *IEEE Trans. Electronic Computers*, 14(3):394–399, 1965.

[3] Nicolas Bacquey, Etienne Grandjean, and Frédéric Olive. Definability by Horn Formulas and Linear Time on Cellular Automata. In *ICALP 2017*, volume 80, pages 99:1–99:14, 2017.

[4] Christian Choffrut and Karel Culik II. On real-time cellular automata and trellis automata. *Acta Informatica*, 21(4):393–407, November 1984.

[5] S. N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machine. *IEEE Transactions on Computing*, 18:349–365, 1969.

[6] Karel Culik. Variations of the firing squad problem and applications. *Inf. Process. Lett.*, 30(3):153–157, 1989.

[7] Karel Culik, Jozef Gruska, and Arto Salomaa. Systolic trellis automata. I. *International Journal Computer Mathematics*, 15(3-4):195–212, 1984.

[8] Marianne Delorme and Jacques Mazoyer. Signals on cellular automata. In Andrew Adamatzky, editor, *Collision-based Computing*, pages 231–275. Springer, 2002.

[9] Marianne Delorme and Jacques Mazoyer. Algorithmic tools on cellular automata. In Rozenberg et al. [32], pages 77–122.

[10] Charles R. Dyer. One-way bounded cellular automata. *Information and Control*, 44(3):261–281, March 1980.

[11] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets'. In *Complexity of Computation, SIAM-AMS Proceedings*, pages 43–73, 1974.

[12] Patrick C. Fischer. Generation of primes by one-dimensional real-time iterative array. *Journal of the ACM*, 12:388–394, 1965.

[13] Erich Grädel. Capturing complexity classes by fragments of second-order logic. *Theoretical Computer Science*, 101(1):35–57, 1992.

[14] Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications*. Springer, 2007.

[15] Anaël Grandjean. Differences between 2d neighborhoods according to real time computation. In *Developments in Language Theory*, pages 198–209, 2017.

[16] Anaël Grandjean and Victor Poupet. Comparing 1d and 2d real time on cellular automata. *CoRR*, abs/1610.00331, 2016.

[17] Anaël Grandjean and Victor Poupet. A linear acceleration theorem for 2d cellular automata on all complete neighborhoods. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 115:1–115:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[18] Etienne Grandjean and Théo Grente. Descriptive complexity for minimal time of cellular automata. In *LICS, 2019*, pages 1–13, 2019.

[19] Etienne Grandjean and Frédéric Olive. A logical approach to locality in pictures languages. *Journal of Computer and System Science*, 82(6):959–1006, 2016.

[20] Oscar H. Ibarra and Tao Jiang. On one-way cellular arrays. *SIAM Journal on Computing*, 16(6):1135–1154, December 1987.

[21] Neil Immerman. *Descriptive complexity*. Springer, 1999.

[22] Hans Kleine Büning and Theodor Lettmann. *Propositional logic - deduction and algorithms*, volume 48 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1999.

[23] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

[24] F. Thomson Leighton. *Introduction to Parallel Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, 1991.

[25] Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[26] Jacques Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2):183 – 238, 1987.

[27] Jacques Mazoyer and Jean-Baptiste Yunès. Computations on cellular automata. In Rozenberg et al. [32], pages 159–188.

[28] Alexander Okhotin. On the equivalence of linear conjunctive grammars and trellis automata. *Theoretical Informatics and Applications*, 38(1):69–88, 2004.

[29] Alexander Okhotin. Conjunctive and boolean grammars: The true general case of the context-free grammars. *Computer Science Review*, 9:27–59, 2013.

[30] David Peleg. *Distributed Computing : A Locality-Sensitive Approach*. SIAM Monographs on Discrete Maths and Applications, 2000.

[31] Victor Poupet. Cellular automata: Real-time equivalence between one-dimensional neighborhoods. In *STACS 2005*, pages 133–144, 2005.

[32] Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors. *Handbook of Natural Computing*. Springer, 2012.

[33] A. R. Smith. Real-time language recognition by one-dimensional cellular automata. *Journal of Computer and System Science*, 6:233–253, 1972.

[34] Véronique Terrier. Language not recognizable in real time by one-way cellular automata. *Theoretical Computer Science*, 156(1–2):281–287, March 1996.

[35] Véronique Terrier. Two-dimensional cellular automata recognizer. *Theoretical Computer Science*, 218(2):325–346, May 1999.

[36] Véronique Terrier. Characterization of real time iterative array by alternating device. *Theoretical Computer Science*, 290(3):2075–2084, 2003.

[37] Véronique Terrier. Two-dimensional cellular automata and deterministic on-line tessalation automata. *Theoretical Computer Science*, 301(1-3):167–186, 2003.

[38] Véronique Terrier. Low complexity classes of multidimensional cellular automata. *Theor. Comput. Sci.*, 369(1-3):142–156, 2006.

[39] Véronique Terrier. Language recognition by cellular automata. In Rozenberg et al. [32], pages 123–158.

[40] Véronique Terrier. Some computational limits of trellis automata. In *Cellular Automata and Discrete Complex Systems*, pages 176–186, 2017.