Descriptive complexity for minimal time of cellular automata

Etienne Grandjean, Théo Grente

Descriptive complexity for minimal time of cellular automata

Étienne Grandjean
Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 CAEN, France
Email: etienne.grandjean@unicaen.fr

Théo Grente
Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 CAEN, France
Email: theo.grente@unicaen.fr

***Abstract*—Descriptive complexity may be useful to design programs in a natural declarative way. This is important for parallel computation models such as cellular automata, because designing parallel programs is considered difficult. Our paper establishes logical characterizations of the three classical complexity classes that model minimal time, called real-time, of one-dimensional cellular automata according to their canonical variants. Our logics are natural restrictions of the existential second-order Horn logic. They correspond to the three ways of deciding a language on a square grid circuit of side n according to the three canonical placements of an input word of length n on the grid. Our key tool is a normalization method that transforms a formula into an equivalent formula that literally mimics a grid circuit.**

1. Introduction

1.1. Descriptive complexity and programming

There are two criteria of interest of 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 [1], [2], which characterizes NP as the class of problems definable in *existential second-order logic* (ESO). Similarly, Immerman-Vardi's Theorem [2], [3] and Grädel's Theorem [4], [5] 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. Typically, 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 [6], [7]. However, the problems in concern can be defined inductively in a natural way. For instance, the product of two integers in binary notation is simply defined by the classical Horner's method and one may hope to directly derive a parallel program from such an inductive process.

1.2. Descriptive complexity and linear time on cellular automata

The present paper is in some sense the sequel of a recent paper [8] (see also [9]). First, [8] observes that the inductive processes defining the problems in concern (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 [2], [5], [10]. 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 hyperplan along some fixed line in a space of dimension $d+1$. To capture these inductive behaviors, [8] 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 [5] 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. [8] 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 also is 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*.

1.3. 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 in concern?

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* [11], [12], [13]. A cellular automaton is said to run in *real-time* if it stops, i.e., gives the answer yes or no, at the *minimal* instant when the elapsed time is 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 [14], [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[1] have been proved to be distinct [11], [16], [17], [18]:

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

The final and decisive step to establish this classification is a nice dichotomy of [18] on *admissible* neighborhoods[2] 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 2 above).

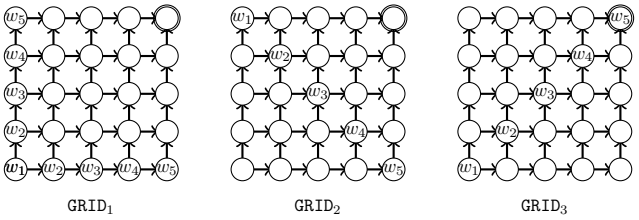
Further, it is *surprising* to notice that

- 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 $\texttt{Trellis} \cup \texttt{RealTime}_{\texttt{IA}} \subsetneqq \texttt{RealTime}_{\texttt{CA}}$ [11], [19], [20],
- while it is *unknown* whether the trivial inclusion $\texttt{RealTime}_{\texttt{CA}} \subseteq \texttt{DLIN}^1_{\texttt{CA}}$ is strict; worse, even whether the inclusion $\texttt{RealTime}_{\texttt{CA}} \subseteq \texttt{LinSpace}$ is strict is an open problem!

1.4. 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. Similarly, [21] has proved the surprising result that Trellis *is* the class of languages generated by *linear conjunctive grammars* (see also [22]) and [23] 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 an *alternating automaton with one counter*.

Logics have two nice and complementary properties: they are *flexible*, hence *expressive*; they have *normal forms*, hence 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 $n \geq 1$, let C_n be the grid circuit $n \times n$ where the state $q \in Q$ (for finite Q) 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 $(j > 1)$, so that the output cell is the site (n, n). Up to symmetries, there are three canonical ways[3] to arrange an input word $w = w_1 \dots w_n$ of length n on the grid C_n, see Figure 1:



$\texttt{GRID}_1$ $\texttt{GRID}_2$ $\texttt{GRID}_3$

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

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

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 in concern (recall: 1: $\texttt{RealTime}_{\texttt{CA}}$; 2: Trellis; 3: $\texttt{RealTime}_{\texttt{IA}}$), and conversely. More precisely, to characterize the three real-time classes, we 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, incl-ESO-HORN and pred-dio-ESO-HORN (defined in the next section), and we prove the following equalities:

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

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

- we define it the *largest 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 *real-time* complexity class in concern.

Structure of the paper: In preliminaries, we recall the classical definitions of one-dimensional cellular automata and of their real-time classes and define our three logics with an example of a problem naturally expressed in such a logic. Section 3 establishes how each of our logics can be normalized. Using these normal forms we show in Section 4 that our three logics exactly characterize the three real-time complexity classes and also – for inclusive logic – the class of linear conjunctive languages of Okhotin [21]. Section 5 gives conclusive remarks.

2. Preliminaries

2.1. Cellular automata and real-time complexity

A cellular automaton in one dimension is a tape of cells (each cell is a finite automaton) indexed by $\mathbb{Z}$. Each cell

1. 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. OIA).

2. 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.

3. 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)...

takes a value from a finite set of states and the cells evolve synchronously along a discrete time scale. The evolution of the cell c is done according to a transition function which takes as input the state of the cell itself and the states of its neighbors at the previous time step and outputs a new state for the cell.

Definition 1 (cellular automaton)**.** *A* cellular automaton *is defined by a 5-tuple $(Q, \Sigma, Q_{accept}, \mathcal{N}, \delta)$ where Q is a finite* set of states*, $\Sigma \subset Q$ is the input alphabet, $Q_{accept} \subset Q$ is the set of* accepting states*, the* neighborhood $\mathcal{N}$ *is a finite ordered subset of $\mathbb{Z}$ and δ is the* transition function *from $Q^{|\mathcal{N}|}$ to Q. The state of the cell c at time t is denoted by $\langle c, t\rangle$. The state of the cell c at time $t > 1$ is defined by the transition function: $\langle c, t\rangle = \delta(\langle c + v, t - 1\rangle : v \in \mathcal{N})$.*

Definition 2 (real time language acceptance)**.** *Cellular automata can act as* language acceptors*. In this case cellular automata work on a finite set of cells indexed by $[1, n]$ where n is the size of the input word $w = w_1 \dots w_n$, one of this cells is also chosen as the* output cell *(usually one of the border cells). A word w is said to be* accepted *by an automaton in real-time if its output cell enters an accepting state immediately after getting all the information from w. The* language accepted *by a cellular automaton $\mathcal{A}$ in real-time (denoted by $L(\mathcal{A})$) is the set of all its accepted words in real-time.*

Convention (permanent state, quiescent state).
All the cells of index outside $[1, n]$ are in the *permanent state* $\sharp$. Without information from the input a cell of index $[1, n]$ is in the *quiescent state* λ.

The real-time computation power of a CA only depends on its *communication scheme*. That is fully determined by the following three specifications: the way the input is fed to the automaton, the way the cells communicate (depending on the neighborhood) and the output cell. The input is usually fed to the automaton in a parallel way: the i^{th} bit of the input is given to the i^{th} cell at the start of the computation. The input can also be fed in a sequential way: the i^{th} bit of the input is given to the first cell at time i for which we add a specific transition function δ_{input}. Usually the output cell is the first cell for two-way communications and the last one for one-way communication.

CA have their input fed in a parallel way and the cells communicate in two-way mode ($\mathcal{N} = \{-1, 0, 1\}$). One-way cellular automata (OCA) and iterative arrays (IA) are two natural variants of CA. OCA are narrowed on the way the cells communicate: the information is only transmitted from left to right ($\mathcal{N} = \{-1, 0\}$). The input mode of IA is no more parallel but sequential.

Definition 3 ($\texttt{RealTime}_{\texttt{CA}}$)**.** *The class* $\texttt{RealTime}_{\texttt{CA}}$ *is the set of languages accepted by real-time CA with a parallel input, the neighborhood $\mathcal{N} = \{-1, 0, 1\}$ and the first cell as the output cell.*

The class $\texttt{RealTime}_{\texttt{CA}}$ is equivalent to $\texttt{RealTime}_{\texttt{OIA}}$, the set of languages accepted by one-way IA with sequential input running in real-time with neighborhood $\mathcal{N} = \{-1, 0\}$ and the last cell as the output cell.

Definition 4 ($\texttt{Trellis}$)**.** *The class* $\texttt{Trellis}$ *is the set of languages accepted by trellis automata or equivalently by OCA running in real-time with a parallel input, the neighborhood $\mathcal{N} = \{-1, 0\}$ and the last cell as the output cell.*

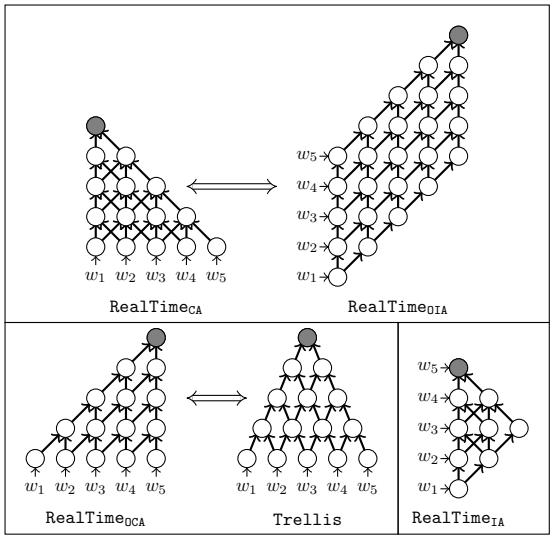


Figure 2. The space-time diagram of the three natural real-time classes

Definition 5 ($\texttt{RealTime}_{\texttt{IA}}$)**.** *The class* $\texttt{RealTime}_{\texttt{IA}}$ *is the set of languages accepted by IA running in real-time with a sequential input, the neighborhood $\mathcal{N} = \{-1, 0, 1\}$ and the first cell as the output cell.*

The space-time diagrams of these real-time classes are depicted in Figure 2.

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 $[1, n] = \{1, \dots, n\}$ only uses the *successor* and *predecessor* functions and the monadic predicates $\texttt{min}$ and $\texttt{max}$ as its *only* arithmetic functions/predicates:

Definition 6 (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*

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

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

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

Let us now define two of our logics:

Definition 7 (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 ψ is a conjunction of Horn clauses on the variables x, y,*
– of signature $\mathcal{S}_\Sigma \cup \mathbf{R}$ (resp. $\mathcal{S}_\Sigma \cup \mathbf{R} \cup \{=\}$) where $\mathbf{R}$ is a set of binary predicates called computation predicates*,*
– 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 either an* input literal/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* $U \in \{\texttt{min}, \texttt{max}\}$ *and an integer* $a \geq 0$,

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 8 (inclusion logic)**.** *An* inclusion Horn formula *is a formula of the form* $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x, y)$ *where* ψ *is a conjunction of Horn clauses of signature* $\mathcal{S}_\Sigma \cup \mathbf{R} \cup \{=, \leq, <\}$ *where* $\mathbf{R}$ *is a set of binary predicates called* computation predicates*, 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* [4] $U(x+a)$, $\neg U(x+a)$, $U(y+a)$ *or* $\neg U(y+a)$, *for* $U \in \{(Q_s)_{s\in\Sigma}, \texttt{min}, \texttt{max}\}$ *and an integer* $a \in \mathbb{Z}$,
2) *or the (in)equality* $x = y$ *or* $x < y$ [5],
3) *or a conjunction of the form*

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

for a computation atom $S(x+a, y-b)$*, with* $S \in \mathbf{R}$ *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 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 $\neg\texttt{min}(x-2)$ and $y \leq n-2$ is written in place of $\neg\texttt{max}(y+1)$.

Technical remarks about our logics: 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 $a, b \geq 0$.

Comparing the input presentation in our logics: The presentation of the input is *more restrictive* in Definition 7 of predecessor logics than in that of inclusion logic (Definition 8) because we have forbidden the use of the successor function for uniformity/aesthetics. However, allowing the same *largest* set of input literals $(\neg)U(x+a)$, $(\neg)U(y+a)$, for $U \in \{(Q_s)_{s\in\Sigma}, \texttt{min}, \texttt{max}\}$ and $a \in \mathbb{Z}$, *does not modify* the expressive power of predecessor logics: steps 5 and 6 of the normalization of inclusive logic in Section 3 can be easily adapted to predecessor logics.

4. Without loss of generality, assume that there is no negation over a predicate Q_s.

5. Then, the hypothesis $x \leq y$ is redundant.

Using our logics for programming: an example

We now give an example of a natural problem expressed in one of our logics. The language notBordered is the set of all words $w \in \Sigma^+$ with no proper prefix equal to a proper suffix:
notBordered $= \{w \mid \forall w'$ such that $w = w'u = vw'$, $w' = \epsilon$ or $w' = w\}$.
This language can be defined by $\Phi_{\texttt{notBordered}} := \exists \texttt{Border} \forall x \forall y \psi$ of pred-ESO-HORN where ψ is the conjunction of the following clauses where $\texttt{Border}(x, y)$ means $w_1 \ldots w_x = w_{y-x+1} \ldots w_y$ (see Figure 3):

1) $\texttt{min}(x) \wedge \neg\texttt{min}(y) \wedge Q_s(x) \wedge Q_s(y) \rightarrow \texttt{Border}(x, y)$
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 all $s \in \Sigma$;
3) $\texttt{max}(y) \wedge \texttt{Border}(x, y) \rightarrow \perp$.

If $\texttt{Border}(x, y)$ is true when y is maximal, then $w_1 \ldots w_x = w_{n-x+1} \ldots w_n$ and therefore $w \notin$ notBordered.

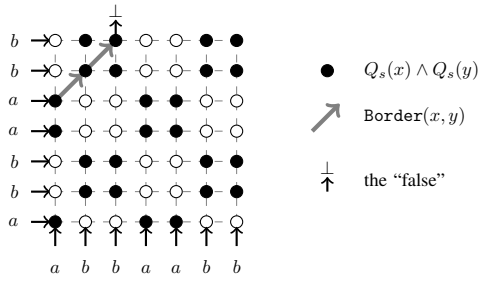


Figure 3. Computation of $\Phi_{\texttt{notBordered}}$ on the word *abbaabb*

So, as a consequence of this paper, notBordered belongs to $\texttt{RealTime}_{\texttt{CA}}$. In fact, more is known [20]:

$$\texttt{notBordered} \in \texttt{RealTime}_{\texttt{CA}} \setminus (\texttt{TREILLIS} \cup \texttt{RealTime}_{\texttt{IA}}).$$

3. Normalizing our logics

The most difficult and main parts of the proofs of our descriptive complexity results, i.e., equalities 1-3 of Subsection 1.4, are the following *normalization lemmas*. They are key ingredients because our normalized formulas can be *literally simulated* by grids and finally by CA of the corresponding real-time classes.

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*)* normalized *as follows: each clause of* Φ' *is of one of the following forms:*

- input clause *of the form, for* $s \in \Sigma$ *and* $R \in \mathbf{R}$*:* $\texttt{min}(x) \wedge \texttt{min}(y) \wedge Q_s(y) \rightarrow R(x, y)$*, 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* $x = y \wedge \neg\texttt{min}(x) \wedge Q_s(x) \rightarrow R(x, y)$*).*
- *the* contradiction clause*, for a fixed* $R_\perp \in \mathbf{R}$*:* $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\perp(x, y) \rightarrow \perp$*;*
- 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 $S(x-1,y) \wedge \neg\texttt{min}(x)$ *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 normalized *as follows: each clause of* Φ' *is of one of the following forms:*

- 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* $R_\perp \in \mathbf{R}$*,* $\texttt{min}(x) \wedge \texttt{max}(y) \wedge R_\perp(x,y) \rightarrow \perp$*;*
- computation clause *of the form*[6] $x < y \wedge \delta_1 \wedge \ldots \wedge \delta_r \rightarrow R(x,y)$*, 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 (resp. languages) so defined.

Proof of the normalization lemmas 1 and 2

The normalization processes of our three logics are quite similar 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.

Proof of Lemma 1: Normalization of predecessor Horn formulas. 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 of 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 these 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 the *only* 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 $R \in \mathbf{R}$: typically, we define the predicate $R^{x-a,y-b}$ that intuitively satisfies the equivalence $R^{x-a,y-b}(x,y) \iff R(x-a,y-b)$. Let us suggest the method by an example. Assume we have initially the Horn clause

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

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 1. *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,* $x = a+1$ *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* $\texttt{min}(x)$*,* $\neg\texttt{min}(x)$*,* $\texttt{max}(x)$*,* $\neg\texttt{max}(x)$*,* $x = a$*,* $x > a$*, for an integer* $a > 1$*, and similarly, for* y*.*

4) Elimination of atoms $x > a$**,** $x = a$**,** $y > a$**,** $y = a$**:** By recurrence on integer $a \geq 1$, let us define the *binary* predicates $R^{x>a}$ (and, similarly, $R^{x=a}$, $R^{y>a}$, $R^{y=a}$) whose intuitive meaning is $x > a$ (resp. $x = a$, $y > a$, $y = a$). The predicate $R^{x>1}$ is defined by the clause $\neg\texttt{min}(x) \rightarrow R^{x>1}(x,y)$. For

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

$a > 1$, 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 $x = a$, for $a > 1$, by $R^{x>a}(x,y)$ and $R^{x=a}(x,y)$, 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 we replace 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:
   • *initialization* for $x = 1$: $\texttt{min}(x) \wedge \eta(y) \rightarrow R(x,y)$ with $\eta(y) \in \{(Q_s(y))_{s\in\Sigma}, (\neg)\texttt{min}(y), (\neg)\texttt{max}(y)\}$;
   • *initialization* for $y = 1$: $\texttt{min}(y) \wedge \eta(x) \rightarrow R(x,y)$ 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 *over-diagonal 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 $R_{\texttt{pred}}$ of intuitive meaning $R_=(x,y) \iff x = y$ and $R_{\texttt{pred}}(x,y) \iff x - 1 = y$ by the following clauses: $\texttt{min}(x) \wedge \texttt{min}(y) \rightarrow R_=(x,y)$; $\neg\texttt{min}(x) \wedge R_=(x-1,y) \rightarrow R_{\texttt{pred}}(x,y)$; $\neg\texttt{min}(y) \wedge R_{\texttt{pred}}(x,y-1) \rightarrow R_=(x,y)$.

Then define the predicate $R_<$ such that $R_<(x,y) \iff x < y$ with the two clauses $\neg\texttt{min}(y) \wedge R_=(x,y-1) \rightarrow R_<(x,y)$ and $\neg\texttt{min}(y) \wedge R_<(x,y-1) \rightarrow R_<(x,y)$. Define similarly the predicate $R_\leq$ such that $R_\leq(x,y) \iff x \leq y$ with the two clauses $\texttt{min}(x) \wedge \texttt{min}(y) \rightarrow R_\leq(x,y)$ and $\neg\texttt{min}(x) \wedge R_<(x-1,y) \rightarrow R_\leq(x,y)$.

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 of the domain: Let us fold the square domain $\{(x,y) \in [1,n]^2\}$ along the diagonal $x = y$ on the over-diagonal 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 $R^{\texttt{inv}}$ whose intuitive meaning is the following: for each $x \leq y$, we have $R^{\texttt{inv}}(x,y) \iff R(y,x)$. 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 *under-diagonal triangle* using predicates $R^{\texttt{inv}}$. Finally, we will express that each $R \in \mathbf{R}$ coincides with its inverse $R^{\texttt{inv}}$ on the fold $x = y$.

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

Folding the computation clauses: Let us describe how to fold the clauses (a) or (b) (folding clauses (c) is easy):
• *Folding of clauses* (a): A clause of the form $S(x-1,y) \wedge \neg\texttt{min}(x) \rightarrow R(x,y)$ is equivalent to the conjunction of clauses i) $x \leq y \wedge S(x-1,y) \wedge \neg\texttt{min}(x) \rightarrow R(x,y)$ and ii) $x > y \wedge S(x-1,y) \wedge \neg\texttt{min}(x) \rightarrow R(x,y)$. 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 $y > x \wedge S(y-1,x) \wedge \neg\texttt{min}(y) \rightarrow R(y,x)$ whose folded (equivalent) form on T_n is $x < y \wedge S^{\texttt{inv}}(x,y-1) \wedge \neg\texttt{min}(y) \rightarrow R^{\texttt{inv}}(x,y)$ since $x < y$ implies $x \leq y - 1$.
• *Folding of clauses* (b): Similarly, a clause of the form $S(x,y-1) \wedge \neg\texttt{min}(y) \rightarrow R(x,y)$ is equivalent to the conjunction of clauses $x < y \wedge S(x,y-1) \wedge \neg\texttt{min}(y) \rightarrow R(x,y)$ and $x \leq y \wedge S^{\texttt{inv}}(x-1,y) \wedge \neg\texttt{min}(x) \rightarrow R^{\texttt{inv}}(x,y)$.

Folding the contradiction clause: Clearly, it is harmless to confuse the (contradiction) predicate $R_\perp$ and its inverse $(R_\perp)^{\texttt{inv}}$; consequently, the contradiction clause itself $\texttt{max}(x) \wedge \texttt{max}(y) \wedge R_\perp(x,y) \rightarrow \perp$ 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 $R^{\texttt{inv}}$ on the diagonal: $x = y \wedge R(x,y) \rightarrow R^{\texttt{inv}}(x,y)$; $x = y \wedge R^{\texttt{inv}}(x,y) \rightarrow R(x,y)$.

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

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

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

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

• Each clause (a) $x \leq y \wedge S(x-1,y) \wedge \neg\texttt{min}(x) \rightarrow R(x,y)$ is replaced by the "equivalent" conjunction of the following two clauses $x \leq y \wedge S^y_{\leftarrow\texttt{max}}(x-1,y) \wedge \neg\texttt{min}(x) \rightarrow R^y_{\leftarrow\texttt{max}}(x,y)$ and $x \leq y \wedge S^y_{\leftarrow\neg\texttt{max}}(x-1,y) \wedge \neg\texttt{min}(x) \rightarrow R^y_{\leftarrow\neg\texttt{max}}(x,y)$.

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

• 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: $x \leq y \wedge S^y_{\leftarrow\texttt{max}}(x,y) \wedge T^y_{\leftarrow\texttt{max}}(x,y) \rightarrow R^y_{\leftarrow\texttt{max}}(x,y)$ and $x \leq y \wedge S^y_{\leftarrow\neg\texttt{max}}(x,y) \wedge T^y_{\leftarrow\neg\texttt{max}}(x,y) \rightarrow R^y_{\leftarrow\neg\texttt{max}}(x,y)$.

• Make a similar substitution for each above clause (d).

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

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

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: The first idea is to group together in each computation clause the hypothesis atoms of the form $R(x,y)$ and the conclusion of the clause. Accordingly, the formula can be rewritten in the form $\Phi := \exists\mathbf{R}\forall x\forall y[\bigwedge_i C_i(x,y) \wedge \bigwedge_{i\in[1,k]}(\alpha_i(x,y) \rightarrow \theta_i(x,y))]$ 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 $R(x-1,y) \wedge \neg\texttt{min}(x)$, $R(x,y-1) \wedge \neg\texttt{min}(y)$, *but not* $R(x,y)$, and $\theta_i(x,y)$ is a Horn clause whose *all* the 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 are of the only forms $R(x-1,y) \wedge \neg\texttt{min}(x)$, $R(x,y-1) \wedge \neg\texttt{min}(y)$. So, we can prove by induction on the sum $x+y$ that the obtained formula Φ' which is a conjunction of clauses (whose hypotheses do include no atom of the form $R(x,y)$ anymore) is equivalent to the above formula Φ. For a detailed proof, see the appendix.

General case: Steps 1-7 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 $R(x,y-1)$; step 7 (folding the domain) is adapted so that it eliminates the atoms of the form $R(y,x)$ by using the following equivalence for $x \leq y$, $R(y,x) \iff R^{\texttt{inv}}(x,y)$. See the details in the appendix.

This achieves the proof of the normalization result `pred-ESO-HORN` = `normal-pred-ESO-HORN`.

Adaptation of steps 1-10 for normalization of predecessor Horn formulas with diagonal input-output: Step 1 is not modified. In step 2, the initialization clauses are now $x = y \wedge Q_s(x) \rightarrow W^x_s(x,y)$ and $x = y \wedge Q_s(y) \rightarrow W^y_s(x,y)$ 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) $x = y \wedge Q_s(x) \rightarrow R(x,y)$; 2) $\texttt{min}(x) \wedge \eta(y) \rightarrow R(x,y)$, with $\eta(y) \in \{(\neg)\texttt{min}(y), (\neg)\texttt{max}(y)\}$; 3) $\texttt{min}(y) \wedge \eta(x) \rightarrow R(x,y)$, with $\eta(x) \in \{(\neg)\texttt{min}(x), (\neg)\texttt{max}(x)\}$. Steps 6 and 7 (folding of 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) $x = y \wedge Q_s(x) \rightarrow R(x,y)$; 2) $\texttt{min}(x) \wedge \texttt{min}(y) \rightarrow R(x,y)$; 3) $\texttt{min}(x) \wedge \neg\texttt{min}(y) \rightarrow R(x,y)$. Clause 2 can be replaced by the equivalent clause 2') $x = y \wedge \texttt{min}(x) \rightarrow R(x,y)$.

Step 9 (from initialization clauses to input clauses) is modified as follows. Define the predicates $R^{\texttt{min}(x)}$, $R^{\texttt{min}(y)}$ and $R^{\neg\texttt{min}(y)}$ by the initialization clauses 4) $x = y \wedge \texttt{min}(x) \rightarrow R^{\texttt{min}(x)}(x,y)$ and 5) $x = y \wedge \texttt{min}(x) \rightarrow R^{\texttt{min}(y)}(x,y)$, and the computation clauses $\neg\texttt{min}(y) \wedge R^{\texttt{min}(x)}(x,y-1) \rightarrow R^{\texttt{min}(x)}(x,y)$, $\neg\texttt{min}(y) \wedge R^{\texttt{min}(y)}(x,y-1) \rightarrow R^{\neg\texttt{min}(y)}(x,y)$, and $\neg\texttt{min}(y) \wedge R^{\neg\texttt{min}(y)}(x,y-1) \rightarrow R^{\neg\texttt{min}(y)}(x,y)$. This allows to replace the initialization clause 3 by the computation clause $R^{\texttt{min}(x)}(x,y) \wedge R^{\neg\texttt{min}(y)}(x,y) \rightarrow R(x,y)$. After those transformations all the initialization clauses are of the form $x = y \wedge Q_s(x) \rightarrow R(x,y)$ (clause 1 above) or $x = y \wedge \texttt{min}(x) \rightarrow R(x,y)$ (clauses 2', 4 and 5 above). By a separation in cases, it is easy to transform each of these clauses into an equivalent conjunction of *input clauses* of the required (normalized) forms: $x = y \wedge \texttt{min}(x) \wedge Q_s(x) \rightarrow R(x,y)$, or $x = y \wedge \neg\texttt{min}(x) \wedge Q_s(x) \rightarrow R(x,y)$.

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. Lemma 1 is proved. □

Proof of Lemma 2: Normalization of inclusion Horn formulas. It divides into seven steps.

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

2) Processing the input: We make available the letters of the input word on the only diagonal $x = y$ by introducing new predicates W_s^{x+a} and W_s^{y+a}, for $a \in \mathbb{Z}$, 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):
$x = y \wedge Q_s(x) \rightarrow W_s^x(x, y)$; $(x = y) \wedge Q_s(x) \rightarrow W_s^y(x, y)$;
$x = y \wedge Q_s(x-a) \wedge x > a \rightarrow W_s^{x-a}(x, y)$, and
$x = y \wedge Q_s(x-a) \wedge x > a \rightarrow W_s^{y-a}(x, y)$, for $a > 0$;
$x = y \wedge Q_s(y+a) \wedge y \leq n-a \rightarrow W_s^{x+a}(x, y)$, and
$x = y \wedge Q_s(y+a) \wedge y \leq n-a \rightarrow W_s^{y+a}(x, y)$, for $a > 0$.
• *Transport clauses*, for $a \in \mathbb{Z}$:
$x < y \wedge W_s^{x+a}(x, y-1) \rightarrow W_s^{x+a}(x, y)$, and
$x < y \wedge W_s^{y+a}(x+1, y) \rightarrow W_s^{y+a}(x, y)$.

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 (see the initialization clauses above) always jointly with $x = y$ and in the only three forms $Q_s(x)$, $Q_s(x-a)$ (jointly with $x > a$), or $Q(y+a)$ (jointly with $y \leq n-a$), for $a > 0$.

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.

As we have done for the Q_s, we want to make available the information about min and max, i.e., about extrema, on the only diagonal $x = y$. 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, $R^{y>a}(x, y) \iff y > a$. For example, define the predicate $R^{x=a}$ (resp. $R^{y=a}$) by the two clauses $x = y \wedge x = a \rightarrow R^{x=a}(x, y)$ and $x < y \wedge R^{x=a}(x, y-1) \rightarrow R^{x=a}(x, y)$ (resp. $x = y \wedge x = a \rightarrow R^{y=a}(x, y)$ and $x < y \wedge R^{y=a}(x+1, y) \rightarrow R^{y=a}(x, y)$). As another example, define $R^{x<n-a}$ by the clauses $x = y \wedge y < n-a \rightarrow R^{x<n-a}(x, y)$ and $x < y \wedge R^{x<n-a}(x, y-1) \rightarrow R^{x<n-a}(x, y)$.

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 now 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 $x < y \wedge S^{x+2,y-2}(x+1, y) \rightarrow S^{x+3,y-2}(x, y)$ 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* $x = y \wedge \delta \rightarrow R(x, y)$, 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) $x < y \wedge S(x+1, y) \rightarrow R(x, y)$;
   b) $x < y \wedge S(x, y-1) \rightarrow R(x, y)$;
   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 separation in cases, one easily obtains the above three forms of initialization clauses.

Justification for computation clauses: Here again, "decompose" each computation clause in clauses of above forms (a,b,c) 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 using 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 lie 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 a 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 $y = n-b$ ($b \geq 0$), except in the contradiction clause: 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: To each clause (a) $x < y \wedge S(x+1, y) \rightarrow R(x, y)$ add the clauses

$x < y \wedge S_{\leftarrow}^{x=a,y=n-b}(x+1,y) \rightarrow R_{\leftarrow}^{x=a-1,y=n-b}(x,y)$, 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 $x < y \wedge S_{\leftarrow}^{x=a,y=n-b}(x,y-1) \rightarrow R_{\leftarrow}^{x=a,y=n-(b-1)}(x,y)$, for all $a \in [1,A]$ and $b \in [1,B]$. Also add for clauses (a,b) the similar (simplified) clauses with only one (instead of two) equality hypothesis. For example, for clause (a) we add the clauses $x < y \wedge S_{\leftarrow}^{x=a}(x+1,y) \rightarrow R_{\leftarrow}^{x=a-1}(x,y)$, for all $a \in [2,A]$, and $x < y \wedge S_{\leftarrow}^{y=n-b}(x+1,y) \rightarrow R_{\leftarrow}^{y=n-b}(x,y)$, 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 provided 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 $x \preceq y \wedge S_{\leftarrow}^{\eta}(x,y) \wedge T_{\leftarrow}^{\theta}(x,y) \rightarrow R_{\leftarrow}^{\eta \cup \theta}(x,y)$. For example, $x \preceq y \wedge S_{\leftarrow}^{x=a}(x,y) \wedge T_{\leftarrow}^{y=n-b}(x,y) \rightarrow R_{\leftarrow}^{x=a,y=n-b}(x,y)$ and $x \preceq y \wedge S_{\leftarrow}^{y=n-b}(x,y) \wedge T_{\leftarrow}^{x=a,y=n-b}(x,y) \rightarrow R_{\leftarrow}^{x=a,y=n-b}(x,y)$.

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 $x=1 \wedge y=n \wedge (R_{\perp})_{\leftarrow}^{x=1,y=n}(x,y) \rightarrow \perp$, which is the contradiction clause required if the predicate $(R_{\perp})_{\leftarrow}^{x=1,y=n}$ is renamed $R_{\perp}$.

6) Elimination of atoms $Q_s(x-a)$, $Q_s(y+b)$ $(a,b>0)$: This step is quite similar to previous step 5: it is described in the appendix.

Recapitulation: After step 6, all the initialization clauses are of the form $x=y \wedge Q_s(x) \rightarrow R(x,y)$ as required[7].

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

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

4. Equivalence between our logics and CA complexity classes

The communication scheme of real-time classes finds a natural expression in our normalized logics. The *input clauses*, the only clauses using unary predicates Q_s, express the way the input is fed to the automaton. The *computation clauses* with a computation atom $R(x,y)$, $R \in \mathbf{R}$, as their conclusion simulate the computation of the CA. Deducing or not deducing the "false" by contradiction clauses means rejection or acceptance. Each of our normalized logics can be described graphically on a grid, indexed by $[1,n]^2$ (see Figure 1).

Encoding automata states by predicates

The set $\mathbf{R}$ of predicates that will be used in formulas expressing the computation of an automaton $\mathcal{A}$, with Q the set of its states, is $\mathbf{R} = \{R_q \mid q \in Q\}$. The intuitive meaning of this predicates is the following: $R_q(c,t)$ is true $\iff$ the cell c is in the state q at time t.

7. Note that an initialization clause of the form $x=y \rightarrow R(x,y)$ can be rewritten $\bigwedge_{s \in \Sigma}(x=y \wedge Q_s(x) \rightarrow R(x,y))$ (separation in cases).

Encoding predicates by automata states

Let Φ be a formula defining a language L, in one of our logics, of the form $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x,y)$ with $\mathbf{R} = (R_1, \ldots, R_m)$ and $R_1 = R_{\perp}$. The set of states that will be used by an automaton $\mathcal{A}$ accepting L is $Q = \{\sharp, \lambda\} \cup \{0,1\}^m$ with $\sharp$ the permanent state and λ the quiescent state. We denote by $(q)_i$ $(i \in [1,m])$, the i^{th} bit of a state $q \in \{0,1\}^m$. Intuitively, $(q)_i$ refers to the predicate R_i. Since the predicate $R_1 = R_{\perp}$ represents the "false", the set of accepting states is the set of states $q \in \{0,1\}^m$ with the first bit $(q)_1$ equal to 0, that is $Q_{accept} = \{0\} \times \{0,1\}^{m-1}$.

4.1. Logical characterization of RealTime$_{\text{CA}}$

In this section, we will show that the languages accepted in real-time by two-way CA with input fed in a parallel way and output read on the first cell are exactly the languages defined by a formula of pred-ESO-HORN.

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

The proof will use the following inclusion scheme:

$$\text{pred-ESO-HORN} = \text{normal-pred-ESO-HORN} \subseteq \text{RealTime}_{\text{OIA}} = \text{RealTime}_{\text{CA}} \subseteq \text{pred-ESO-HORN}$$

The equality pred-ESO-HORN = normal-pred-ESO-HORN has already been proved in Section 3 and the other equality RealTime$_{\text{OIA}}$ = RealTime$_{\text{CA}}$ is folklore in automata theory. Two inclusions are left to be proved.

Lemma 3. RealTime$_{\text{CA}} \subseteq$ pred-ESO-HORN

Proof. We will show that for each automaton $\mathcal{A} \in$ RealTime$_{\text{CA}}$ we can build a formula $\Phi \in$ pred-ESO-HORN such that: $w \in L(\mathcal{A}) \iff \langle w \rangle$ satisfies the formula Φ.

First of all, one can be easily convinced that the computation power of a CA $\mathcal{A} \in$ RealTime$_{\text{CA}}$ with neighborhood $\mathcal{N}_{\mathcal{A}} = \{-1,0,1\}$ is equal to the computation power of a OCA $\mathcal{A}'$ with neighborhood $\mathcal{N}_{\mathcal{A}'} = \{-2,-1,0\}$ running within the same computation time n where n is the size of the input (see Figure 4). This transformation can be seen on the space-time diagram of $\mathcal{A}$ with the variable change: $c \mapsto c+t-1$, where c is the index of the cell and t the time step of the computation. This geometrical transformation does not change the computation power so that: $L(\mathcal{A}) = L(\mathcal{A}')$.

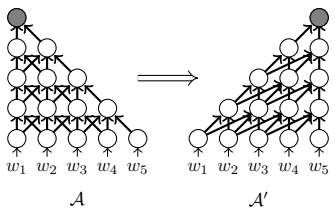


Figure 4. Neighborhood's change: from $\{-1,0,1\}$ to $\{-2,-1,0\}$

Input: The parallel input of $\mathcal{A}'$ is expressed by the conjunction $\psi_{input} = \bigwedge_{s \in \Sigma}(\texttt{min}(t) \wedge Q_s(c) \rightarrow R_s(c,t))$ for Σ the input alphabet: each cell c is in the state $w_c \in \Sigma$ at the start of the computation.

Computation: The state evolution of a cell of $\mathcal{A}'$ is given by the transition function:
$\langle c,t \rangle = \delta_{\mathcal{A}'}(\langle c-2,t-1 \rangle, \langle c-1,t-1 \rangle, \langle c,t-1 \rangle)$.
A transition rule $\delta_{\mathcal{A}'}(q_2,q_1,q_0) = q$ for $q_2 \neq \sharp$ is expressed by the computation clause:
$c > t \wedge \neg \texttt{min}(t) \wedge R_{q_2}(c-2,t-1) \wedge R_{q_1}(c-1,t-1)$

$\wedge R_{q_0}(c,t-1) \to R_q(c,t)$.
The specific case where q_2 is the permanent state $\sharp$ ($\delta_{\mathcal{A}'}(\sharp,q_1,q_0)=q$) is handled by the clause:
$c=t \wedge \neg\texttt{min}(t) \wedge R_{q_1}(c-1,t-1) \wedge R_{q_0}(c,t-1) \to R_q(c,t)$.
We denote $\psi_{compute}$ the conjunction of the above two sets of clauses.

Remark: By construction of ψ_{input} and $\psi_{compute}$, we have the following equivalence for $\psi' \coloneqq \psi_{input} \wedge \psi_{compute}$. 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 the state q at time t.

Output: The contradiction clauses express the output on the last cell:
$\psi_{output} \coloneqq \bigwedge_{q\in Q\setminus Q_{accept}} (\texttt{max}(c) \wedge \texttt{max}(t) \wedge R_q(c,t) \to \bot)$

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:

The cell 1 of $\mathcal{A}$ is in an accepting state q at time n
$\iff$
The cell n of $\mathcal{A}'$ is in an accepting state q at time n
$\iff$
The conjunction $\bigwedge_{q\in Q\setminus Q_{accept}} R_q(n,n)$ is false 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 $L(\mathcal{A}) \in$ pred-ESO-HORN. □

Lemma 4. normal-pred-ESO-HORN $\subseteq \texttt{RealTime}_{\texttt{OIA}}$

Proof. We will show that for every language $L \subseteq \Sigma^+$ defined by a formula $\Phi \in$ normal-pred-ESO-HORN, a one-way iterative array $\mathcal{A} \in \texttt{RealTime}_{\texttt{OIA}}$ can be constructed such that $L = L(\mathcal{A})$.

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

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

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

We first translate the formula Φ into a dependency graph of type $\texttt{GRID}_1$ (see Figure 1, with input exclusively on the left side). It will then be turned into a real-time OIA $\mathcal{A}$. The transition function is composed by an input transition function $\delta_{\texttt{input}}$ only applied on the first cell and the general transition function δ applied on the other cells c.

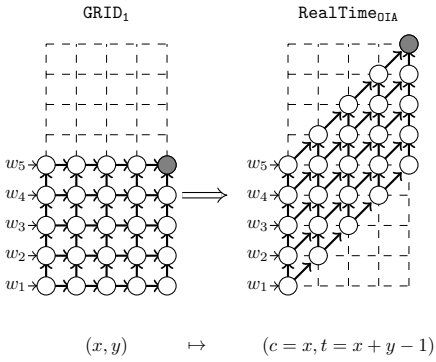


Figure 5. Variable change

For all $i \in [1,m]$; $s \in \Sigma$; $q,l,r \in Q \setminus \{\sharp,\lambda\}$:

Input transition function: $\delta_{\texttt{input}} : \Sigma \times Q \to Q$. The bit $(\delta_{input}(s,\sharp))_i$ is 1 if and only if there is an input clause $\texttt{min}(x) \wedge \texttt{min}(y) \wedge Q_s(y) \to R_i(x,y)$ in ψ. $(\delta_{input}(s,q))_i$ is 1 if and only if there exists in ψ an input clause $\texttt{min}(x) \wedge \neg\texttt{min}(y) \wedge Q_s(y) \to R_i(x,y)$ or a computation clause $\neg\texttt{min}(y) \wedge \bigwedge_{h\in H} R_h(x,y-1) \to R_i(x,y)$ with $(q)_h = 1$, for all $h \in H$.

Transition function: $\delta : Q \times Q \to Q$ applied on all cells $c \in [2,n]$. The bit $(\delta(l,r))_i$ is 1 if and only if there exists in ψ a computation clause $\bigwedge_{h\in H} R_h(x-1,y) \wedge \neg\texttt{min}(x) \wedge \bigwedge_{h\in H'} R_h(x,y-1) \wedge \neg\texttt{min}(y) \to R_i(x,y)$ such that $(l)_h = 1$ for all $h \in H$ and $(r)_h = 1$ for all $h \in H'$.
The bit $(\delta(l,\lambda))_i$ is 1 if and only if there is in ψ a computation clause $\neg\texttt{min}(x) \wedge \bigwedge_{h\in H} R_h(x-1,y) \to R_i(x,y)$ such that $(l)_h = 1$, for all $h \in H$.

Let $\mathcal{A} = (Q, \Sigma, Q_{accept}, \mathcal{N}, \delta_{\texttt{input}}, \delta)$ be the OIA defined above after making the change of variables of Figure 5: $c = x$; $t = x + y - 1$. By construction of $\mathcal{A}$, we have the following equivalences concluding the proof.
$\forall w \in \Sigma^n$:

1) $\forall (a,b) \in [1,n]^2, \forall i \in [1,m]$, 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 at time $t = a + b - 1$ in a state q with $(q)_i = 1$.
2) $\mathcal{A}$ accepts w in real-time $\iff \langle w\rangle$ satisfies Φ.

□

4.2. Logical characterization of $\texttt{RealTime}_{\texttt{IA}}$

In this section, we will show that the languages accepted in real-time by IA are exactly the languages defined by formulas of pred-dio-ESO-HORN.

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

The proof of Theorem 2 is close to the one of Theorem 1 and is obtained by the following inclusion scheme:
pred-dio-ESO-HORN = normal-pred-dio-ESO-HORN
$\subseteq \texttt{RealTime}_{\texttt{IA}} \subseteq$ pred-dio-ESO-HORN.

The equality pred-dio-ESO-HORN = normal-pred-dio-ESO-HORN has been already proved in Section 3. We will now prove the two remaining inclusions.

Lemma 5. $\texttt{RealTime}_{\texttt{IA}} \subseteq$ pred-dio-ESO-HORN.

Proof. Let $\mathcal{A}$ be an automaton in $\texttt{RealTime}_{\texttt{IA}}$, we apply the transformation $c \mapsto c + t - 1$ on its time-space diagram.

This transformation gives us a new automaton $\mathcal{A}'$ with the neighborhood $\mathcal{N}' = \{-2, -1, 0\}$ and the input still fed sequentially but in the following way: the i^{th} bit of the input is given to the cell i at time i. Since the input presentation is the only change between the computation of an automaton in $\texttt{RealTime}_{\texttt{IA}}$ and an automaton in $\texttt{RealTime}_{\texttt{CA}}$, the computation clauses and the contradiction clauses will stay the same.

Input: The diagonal input of $\mathcal{A}'$ is expressed by the conjunction of the input clauses:
$\psi_{input} \coloneqq \bigwedge_{s\in\Sigma}(t = c \wedge Q_s(c) \to R_s(c,t))$ for $\Sigma \subset Q$.

Computation: The conjunction $\psi_{compute}$ is defined from the transition rules of $\mathcal{A}'$ as in the previous section.

Let ψ' be the conjunction $\psi_{input} \wedge \psi_{compute}$.

Output: The output reading is done on the same cell as in the previous section.
$\psi_{output} \coloneqq \bigwedge_{q\in Q\setminus Q_{accept}}(\texttt{max}(c) \wedge \texttt{max}(t) \wedge R_q(c,t) \to \bot)$

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:

The cell 1 of $\mathcal{A}$ is in an accepting state q at time n
$\iff$
The cell n of $\mathcal{A}'$ is in an accepting state q at time n
$\iff$
The conjunction $\bigwedge_{q\in Q\setminus Q_{accept}} R_q(n,n)$ is false 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 $L(\mathcal{A}) \in$ pred-dio-ESO-HORN.

□

Lemma 6. $\texttt{normal-pred-dio-ESO-HORN} \subseteq \texttt{RealTime}_{\texttt{IA}}$

Proof sketch. Since the proof is similar to that of Lemma 4, we will here just give an hint on how to associate to each site $(x,y) \in [1,n]^2$ such that $x \leq y$ a site (c,t) of the space-time diagram of an iterative array $\mathcal{A}$ running in real-time (see Figure 6). The sites $(x,y) \in [1,n]^2$ such that $x \geq y$ are similarly handled.

First, we apply to the set of sites (x,y) of the dependency graph of Φ the variable change $c' = y - x + 1; t' = x + y - 1$. This variable change turns respectively the dependencies $(x-1,y) \to (x,y)$ and $(x,y-1) \to (x,y)$ into $(c'+1,t'-1) \to (c',t')$ and $(c'-1,t'-1) \to (c',t')$ 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 6).

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 6).

□

4.3. Logical characterization of Trellis and linear conjunctive grammars

Conjunctive grammars, introduced by Okhotin [21], are an extension of context-free grammars. This class of formal

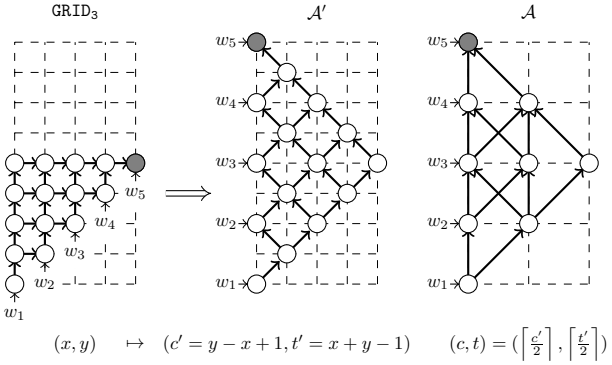


Figure 6. Variable change and grouping

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 [21] that languages obtained by the linear restriction of conjunctive grammars are the same as the ones recognized by trellis automata. The class of languages generated by linear conjunctive grammars is denoted $LinConj$.

Each rule of a linear conjunctive grammar $\mathcal{G} = (\Sigma, N, P, S)$ in normal form is a rule of the form:
$A \to sB_1\& \dots \& sB_\ell \& C_1 t \& \dots \& C_p t$ 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 by least solution of a system of language equations with concatenation, union and intersection. For example, the rule
$A \to sB_1\& \dots \& sB_\ell \& C_1 t \& \dots \& C_p t$
gives us the language equation:
$L(A) = sL(B_1) \cap \dots \cap sL(B_\ell) \cap L(C_1)t \cap \dots \cap L(C_p)t$
where $L(A)$ is the set of all words having the syntactic property A.

Our inclusive logic characterizes the class of (complements of) linear conjunctive languages:

Theorem 3. *For any $L \in \Sigma^+$ we have:*
$L \in \texttt{incl-ESO-HORN} \iff (\Sigma^+ \setminus L) \in LinConj$.

The theorem is proved in two steps.

Lemma 7. $L \in \texttt{incl-ESO-HORN} \Rightarrow (\Sigma^+ \setminus L) \in LinConj$

Proof. Let us show that for each language $L \in$ incl-ESO-HORN defined by a formula $\Phi \in$ normal-incl-ESO-HORN, $\Phi = \exists \mathbf{R} \forall x \forall y \psi(x,y)$, we can build a grammar $G_\Phi = (\Sigma, N, P, S)$ such that: $w \in L \iff w \notin L(G_\Phi)$
The grammar G_Φ is defined as follows.
The set of non-terminal symbols of G_Φ is $N = \mathbf{R}$ and $S = R_\perp$ is the start symbol. The rules of $\mathcal{G}_\Phi$ are the following:
To each input clause $x = y \wedge Q_s(x) \to R(x,y)$ of Φ we associate the rule $R \to s$.
To each computation clause of Φ
$x < y \wedge S_1(x+1,y) \wedge \dots \wedge S_\ell(x+1,y) \wedge T_1(x,y-1) \wedge \dots \wedge T_p(x,y-1) \to R(x,y)$ we associate the rules
$R \to sS_1\& \dots \& sS_\ell \& T_1 t \& \dots \& T_p t$, for all $s, t \in \Sigma$.

Let ψ' be the conjunction ψ without the contradiction clause. By an easy induction, the following equivalence can be proved, for all $R \in \mathbf{R}$, all words $w = w_1 \dots w_n \in \Sigma^+$, and all intervals $[a,b] \subseteq [1,n]$:

$w_a \dots w_b \in L(R) \iff$
$\Big(\forall x \forall y \psi' \wedge \bigwedge_{i \in [a,b]} Q_{w_i}(i)\Big) \to R(a,b)$ is a tautology.

Taking $R = R_\perp$ and $[a,b] = [1,n]$ we obtain:
$w \in L(G_\Phi) \iff \langle w \rangle \not\models \Phi$ as claimed. □

Lemma 8. $(\Sigma^+ \setminus L) \in LinConj \Rightarrow L \in$ `incl-ESO-HORN`

Proof. Let L be a language, subset of Σ^+, such that $(\Sigma^+ \setminus L) \in LinConj$. We associate to the linear grammar $\mathcal{G} = (\Sigma, N, P, S)$ defining $(\Sigma^+ \setminus L)$ with $N = \{A_1, \dots, A_k\}$ in normal form, as presented above, the formula $\Phi_\mathcal{G} := \exists \mathbf{R} \forall x \forall y \psi(x,y)$ with $\mathbf{R} = \{A_1, \dots, A_k\}$ and ψ the conjunction of the following Horn clauses:
the computation clause $x < y \wedge Q_s(x) \wedge \bigwedge_{i=1}^{\ell} B_i(x+1,y) \wedge \bigwedge_{i=1}^{p} C_i(x, y-1) \wedge Q_t(y) \to A(x,y)$ for each rule $A \to sB_1 \& \dots \& sB_\ell \& C_1 t \& \dots \& C_p t$;
the clause $x = y \wedge Q_s(x) \to A(x,y)$ for each rule $A \to s$;
and the contradiction clause $\mathtt{min}(x) \wedge \mathtt{max}(y) \wedge S(x,y) \to \perp$.

The proof of the equivalence $w \in L(\mathcal{G}) \iff \langle w \rangle \models \Phi_\mathcal{G}$ is the same as for the previous Lemma 7. □

Logical characterization of `Trellis`

Clearly, the final state (result) q of a trellis automaton $\mathcal{A} = (Q, \Sigma, Q_{accept}, \delta)$ acting on a word $w_x \dots w_y$ of length greater than 1 ($x < y$) is completely determined by the final state q_l of $\mathcal{A}$ acting on the prefix $w_x \dots w_{y-1}$ and the final state q_r of $\mathcal{A}$ acting on the suffix $w_{x+1} \dots w_y$: $q = \delta(q_l, q_r)$. This functional dependence $((x, y-1), (x+1, y)) \to (x,y)$ is exactly expressed by the computation clauses of the normalized inclusive logic. This suggests the following equality.

Theorem 4. `incl-ESO-HORN` = `Trellis`

The theorem is proved in two steps.

Lemma 9. `Trellis` ⊆ `incl-ESO-HORN`

Proof. Let $\mathcal{A}$ be a trellis automaton that accepts a language $L \subseteq \Sigma^+$ and let be a word $w = w_1 \dots w_n \in \Sigma^+$. Let us introduce the set of binary predicates $\mathbf{R} = \{R_q \mid q \in Q\}$ with intuitive meaning: $R_q(x,y)$ is true $\iff$ the final state of $\mathcal{A}$ acting on the subword $w_x \dots w_y$ is q. The following clauses describe the computation of $\mathcal{A}$.
Input clauses:
$\psi_{input} \coloneqq \bigwedge_{s \in \Sigma} (x = y \wedge Q_s(x) \to R_s(x,y))$, for $s \in \Sigma$.
Computation clauses:
$\psi_{compute} \coloneqq \bigwedge_{(q_l, q_r) \in Q^2} (x < y \wedge R_{q_l}(x, y-1) \wedge R_{q_r}(x+1, y) \to R_q(x,y))$ where $q = \delta(q_l, q_r)$.
Contradiction clauses:
$\psi_{output} \coloneqq \bigwedge_{q \in Q \setminus Q_{accept}} (\mathtt{min}(x) \wedge \mathtt{max}(y) \wedge R_q(x,y) \to \perp)$

Let $\psi' \coloneqq \psi_{input} \wedge \psi_{compute}$ and $\psi \coloneqq \psi' \wedge \psi_{output}$.

For each word $w = w_1 \dots w_n \in \Sigma^+$ we have the following equivalences:

$\mathcal{A}$ accepts w in time n
$\iff$
The conjunction $\bigwedge_{q \in Q \setminus Q_{accept}} R_q(1,n)$ is false 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 $L(\mathcal{A}) \in$ `incl-ESO-HORN`. □

Lemma 10. `incl-ESO-HORN` ⊆ `Trellis`

Proof sketch. Let Φ be a formula of `normal-incl-ESO-HORN`, 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 7). The transition function of this automaton is then deduced from the computation clauses of Φ as in sections 4.1 and 4.2.

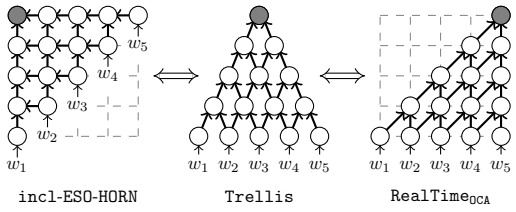


Figure 7. The bijection between `incl-ESO-HORN` and `Trellis`

□

5. Concluding remarks

It was known that the three complexity classes studied in this paper 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 [18], and has several equivalent characterizations in other frameworks: e.g, `Trellis` is the class of linear conjunctive languages [21].

In this paper, we have presented a unified view of these three real-time classes as part of descriptive complexity. We hope to have convinced the reader that the logics we have defined are useful tools to express problems in a natural way and to deduce automatically from any formula in such a logic an equivalent cellular automaton running in real-time. This is exemplified by languages `notBordered` and `Palindrome`, which can be succinctly defined in `pred-ESO-HORN` and `incl-ESO-HORN`, respectively, and hence, belong to RealTime$_{\mathtt{CA}}$ and `Trellis`, respectively. Conversely, the problem `Prime` $:= \{w \in \Sigma^* \mid |w| \text{ is a prime integer}\}$ belongs to RealTime$_{\mathtt{CA}}$ (and to RealTime$_{\mathtt{IA}}$) by Fischer's algorithm [6], [7]; therefore, it belongs to `pred-ESO-HORN`.

Further, we believe that the normalized versions of our three logics, identified to square grid circuits – a natural concept – offer a fresh view of the real-time complexity classes and an additional argument for their robustness. In addition, we are thinking about broadening our logics by allowing a limited use of negation on computation atoms like in *Stratified Datalog* [10], for easier programming inside these logics and without changing their real-time complexity.

Acknowledgments: This paper would not exist without the inspiration and guidance of Véronique Terrier. Her teaching, her deep insights into cellular automata, the references and advice she generously gave us, as well as her careful reading, were essential in designing and finalizing our concepts and results.

References

[1] R. Fagin, "Generalized first-order spectra and polynomial-time recognizable sets'," in *Complexity of Computation, SIAM-AMS Proceedings*, 1974, pp. 43–73.

[2] L. Libkin, *Elements of Finite Model Theory*, ser. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[3] N. Immerman, *Descriptive complexity*. Springer, 1999.

[4] E. Grädel, "Capturing complexity classes by fragments of second-order logic," *Theoretical Computer Science*, vol. 101, no. 1, pp. 35–57, 1992.

[5] E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, and S. Weinstein, *Finite Model Theory and Its Applications*. Springer, 2007.

[6] P. C. Fischer, "Generation of primes by one-dimensional real-time iterative array," *Journal of the ACM*, vol. 12, pp. 388–394, 1965.

[7] M. Delorme and J. Mazoyer, "Signals on cellular automata," in *Collision-based Computing*, A. Adamatzky, Ed. Springer, 2002, pp. 231–275.

[8] N. Bacquey, E. Grandjean, and F. Olive, "Definability by Horn Formulas and Linear Time on Cellular Automata," in *ICALP 2017*, vol. 80, 2017, pp. 99:1–99:14.

[9] E. Grandjean and F. Olive, "A logical approach to locality in pictures languages," *Journal of Computer and System Science*, vol. 82, no. 6, pp. 959–1006, 2016.

[10] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[11] S. N. Cole, "Real-time computation by n-dimensional iterative arrays of finite-state machine," *IEEE Transactions on Computing*, vol. 18, pp. 349–365, 1969.

[12] A. R. Smith, "Real-time language recognition by one-dimensional cellular automata," *Journal of Computer and System Science*, vol. 6, pp. 233–253, 1972.

[13] C. R. Dyer, "One-way bounded cellular automata," *Information and Control*, vol. 44, no. 3, pp. 261–281, Mar. 1980.

[14] V. Terrier, "Two-dimensional cellular automata recognizer," *Theoretical Computer Science*, vol. 218, no. 2, pp. 325–346, May 1999.

[15] A. Grandjean, "Differences between 2d neighborhoods according to real time computation," in *Developments in Language Theory*, 2017, pp. 198–209.

[16] C. Choffrut and K. Čulík II, "On real-time cellular automata and trellis automata," *Acta Informatica*, vol. 21, no. 4, pp. 393–407, Nov. 1984.

[17] O. H. Ibarra and T. Jiang, "On one-way cellular arrays," *SIAM Journal on Computing*, vol. 16, no. 6, pp. 1135–1154, Dec. 1987.

[18] V. Poupet, "Cellular automata: Real-time equivalence between one-dimensional neighborhoods," in *STACS 2005*, 2005, pp. 133–144.

[19] K. Čulík II, J. Gruska, and A. Salomaa, "Systolic trellis automata. I," *International Journal Computer Mathematics*, vol. 15, no. 3-4, pp. 195–212, 1984.

[20] V. Terrier, "Language not recognizable in real time by one-way cellular automata," *Theoretical Computer Science*, vol. 156, no. 1–2, pp. 281–287, Mar. 1996.

[21] A. Okhotin, "On the equivalence of linear conjunctive grammars and trellis automata," *Theoretical Informatics and Applications*, vol. 38, no. 1, pp. 69–88, 2004.

[22] ——, "Conjunctive and boolean grammars: The true general case of the context-free grammars," *Computer Science Review*, vol. 9, pp. 27–59, 2013.

[23] V. Terrier, "Characterization of real time iterative array by alternating device," *Theoretical Computer Science*, vol. 290, no. 3, pp. 2075–2084, 2003.

APPENDICES

Appendix A: Expressing natural problems in our logics

Expressing problems `Palindrome` **and** `notPalindrome` **in inclusion logic.** The problem `Palindrome` is expressed by the conjunction of the following clauses:

$x < y \wedge Q_s(x) \wedge Q_t(y) \to \texttt{notPal}(x,y)$, for all $s,t \in \Sigma$ with $s \neq t$;

$x < y \wedge \texttt{notPal}(x+1, y-1) \to \texttt{notPal}(x,y)$;

$x \leq y \wedge \texttt{min}(x) \wedge \texttt{max}(y) \wedge \texttt{notPal}(x,y) \to \bot$.

The problem `notPalindrome` is expressed by the conjunction of the following clauses: $x = y \to \texttt{Pal}(x,y)$, $x \leq y \wedge \texttt{Successor}(x,y) \wedge Q_s(x) \wedge Q_s(y) \to \texttt{Pal}(x,y)$, $x < y \wedge \texttt{Pal}(x+1, y-1) \wedge Q_s(x) \wedge Q_s(y) \to \texttt{Pal}(x,y)$, for all $s \in \Sigma$, and $x \leq y \wedge \texttt{Pal}(x,y) \to \bot$.

Here, $\texttt{Successor}(x,y)$ intuitively means $x+1 = y$ and is defined by the following clauses: $x = y \to \texttt{Equal}(x,y)$ and $x < y \wedge \texttt{Equal}(x+1, y) \to \texttt{Successor}(x,y)$.

Appendix B: Complements of the proofs of Lemmas 1 and 2

Step 10 of Lemma 1: Elimination of atoms $R(x,y)$ as hypotheses: The first idea is to group together in each computation clause the hypothesis atoms of the form $R(x,y)$ and the conclusion of the clause. Accordingly, the formula obtained Φ can be rewritten in the form $\Phi := \exists \mathbf{R} \forall x \forall y [\bigwedge_i C_i(x,y) \wedge \bigwedge_{i \in [1,k]} (\alpha_i(x,y) \to \theta_i(x,y))]$, where the C_i's are the input clauses and the contradiction clause and each computation clause is written in the form $\alpha_i(x,y) \to \theta_i(x,y)$ where $\alpha_i(x,y)$ is a conjunction of formulas of the only forms $R(x-1,y) \wedge \neg\texttt{min}(x)$, $R(x,y-1) \wedge \neg\texttt{min}(y)$, *but not* $R(x,y)$, and $\theta_i(x,y)$ is a Horn clause *all* the atoms of which are of the form $R(x,y)$.

We 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*

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

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

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

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

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

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 be proved:

Lemma 11 (Horn property: folklore)**.** *Let F be a* strict Horn formula *of propositional calculus, that is a conjunction of clauses of the form $p_1 \wedge \ldots \wedge p_k \rightarrow p_0$ 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 $F \rightarrow q$ is a tautology. F has the same minimal model*[8] *as F'.*

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

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

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

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

As each α_i is a conjunction of formulas of the form $R(x-1,y) \wedge \neg \mathtt{min}(x)$, or $R(x,y-1) \wedge \neg \mathtt{min}(y)$, 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

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

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 $a'+b' = t-1$. Let any 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 $\bigwedge_{h \in K_J} (\bigwedge_{i \in J} \alpha_i(a,b) \rightarrow R_h(a,b))$ 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 $(\langle w \rangle, \mathbf{R})$.

8. For example, for $F := p_1 \wedge p_3 \wedge (p_1 \wedge p_3 \rightarrow p_5) \wedge (p_1 \wedge p_2 \rightarrow p_4)$, 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 $I(p_2) = I(p_4) = 0$.

From (1) and (2), we deduce that in $(\langle w \rangle, \mathbf{R})$ the conjunction $\bigwedge_{J \subseteq [1,k]} \bigwedge_{h \in K_J} (\bigwedge_{i \in J} \alpha_i(a,b) \rightarrow R_h(a,b))$ 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,

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

or, equivalently,

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

As a consequence of Lemma 11, 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 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 $a+b = t$:

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

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 converse implication $\Phi' \Rightarrow \Phi$. Claim 1 is demonstrated. □

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) $S(x-1,y) \wedge \neg \mathtt{min}(x) \rightarrow R(x,y)$;
(b) $S(x,y-1) \wedge \neg \mathtt{min}(y) \rightarrow R(x,y)$;
(c) $S(x,y) \wedge T(x,y) \rightarrow R(x,y)$; (d) $S(y,x) \rightarrow R(x,y)$.

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 $R_1(x-1,y) \wedge \neg \mathtt{min}(x) \wedge R_2(x,y-1) \wedge \neg \mathtt{min}(y) \wedge R_3(y,x) \rightarrow R_4(x,y)$ is "equivalent" to the conjunction of the following clauses using the new predicates R_5, R_6, R_7, R_8: $R_1(x-1,y) \wedge \neg \mathtt{min}(x) \rightarrow R_5(x,y)$; $R_2(x,y-1) \wedge \neg \mathtt{min}(y) \rightarrow R_6(x,y)$; $R_5(x,y) \wedge R_6(x,y) \rightarrow R_7(x,y)$; $R_3(y,x) \rightarrow R_8(x,y)$; $R_7(x,y) \wedge R_8(x,y) \rightarrow R_4(x,y)$.

The folding of clauses (a-c) is not modified. Let us describe how to fold the (new) clauses (d): $S(y,x) \rightarrow R(x,y)$. Obviously, such a clause is equivalent to the conjunction of the two clauses (i) $x \leq y \wedge S(y,x) \rightarrow R(x,y)$ and (ii) $y \leq x \wedge S(y,x) \rightarrow R(x,y)$. The equivalent "folded" form of clause (i) is $x \leq y \wedge S^{\mathtt{inv}}(x,y) \rightarrow R(x,y)$. The clause (ii) is equivalent to the clause $x \leq y \wedge S(x,y) \rightarrow R(y,x)$ the equivalent "folded" form of which is $x \leq y \wedge S(x,y) \rightarrow R^{\mathtt{inv}}(x,y)$. Finally, steps 8-10 are not modified.

Step 6 of Lemma 2: 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 $x = y \wedge R^{x-a}_{\leftarrow,s} \rightarrow R(x,y)$ (resp. $x = y \wedge R^{y+b}_{\leftarrow,s} \rightarrow R(x,y)$).

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

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