Fractal Parallelism: Solving SAT in bounded space and time

Denys Duchier, Jérôme Durand-Lose*, and Maxime Senot

LIFO, Université d'Orléans,
B.P. 6759, F-45067 ORLÉANS Cedex 2.

Abstract. Abstract geometrical computation can solve NP-complete problems efficiently: any boolean constraint satisfaction problem, instance of SAT, can be solved in bounded space and time with simple geometrical constructions involving only drawing parallel lines on a Euclidean space-time plane. Complexity as the maximal length of a sequence of consecutive segments is quadratic. The geometrical algorithm achieves massive parallelism: an exponential number of cases are explored simultaneously. The construction relies on a fractal pattern and requires the same amount of space and time independently of the SAT formula.

Key-words. Abstract geometrical computation; Signal machine; Fractal; SAT; Massive parallelism; Model of computation.

1 Introduction

SAT, the problem of determining the satisfiability of propositional formulae, is the poster-child of combinatorial complexity and the natural representative of the classical time complexity class NP [Cook, 1971, Levin, 1973]. As such, it is a natural challenge to consider when investigating new computing machinery (quantum, NDA, membrane, hyperbolic spaces...) [Sosík, 2003, Margenstern and Morita, 2001]. In this paper, we show that *signal machines*, through fractal parallelization, are capable of solving SAT in bounded space and time, and thus by NP-completeness of SAT, signal machine can solve any NP-problem *i.e.* hard problems according to classical models like Turing-machine. We also offer a more pertinent notion of complexity, namely *depth*, which is quadratic for our proposed construction for SAT.

The geometrical context proposed here is the following: dimensionless *particles/ signals* move uniformly on the real axis. When a set of particles collide, they are replaced by a new set of particles according to a chosen collection of *collision rules*. By adjoining a temporal dimension to the continuous space-line, we can visualize the chronology of motions and collisions of these particles as a *space-time diagram* (in which both space and time are continuous). Since particles have constant speed, their trajectories in the diagram consist of line segments.

* Corresponding author `Jerome.Durand-Lose@univ-orleans.fr`

Models of computation, conventional or not, are frequently based on mathematical idealizations of physical concepts and investigate the consequences, on computational power, of such abstractions (quantum, membrane, closed time-like curves, black holes. . .) [Păun, 2001, Brun, 2003, Etesi and Németi, 2002]. However, oftentimes, the idealization is such that it must be interpreted either as allowing information to have infinite density (e.g. an oracle), or to be transmitted at infinite speed (global clock, no spatial extension. . .). On this issue, the model of signal machines stands in contradistinction with other abstract models of computation: it respects the principle of causality, density and speed of information are finite, as are the sets of objects manipulated. Nonetheless, it remains a resolutely abstract model with no apriori ambition to be physically realizable, and it deals with theoretical issues such as computational power.

Signal machines are Turing-universal [Durand-Lose, 2005] and allows to do analog computation by a systematic use of the continuity of space and time [Durand-Lose, 2008, 2009a,b]. Other *geometrical models of computation* exist and allow to compute: colored universe [Jacopini and Sontacchi, 1990], geometric machines [Huckenbeck, 1989], piece-wise constant derivative systems [Asarin and Maler, 1995, Bournez, 1997], optical machines [Naughton and Woods, 2001]. . .

Most of the work to date in this domain, called *abstract geometrical computation* (AGC), has dealt with the simulation of sequential computations even though the model, seen as a continuous extension of cellular automata, is inherently parallel. (The connexion with CA is briefly illustrated on Fig. 1) In the present paper, we describe a massively parallel evaluation of all possible valuations for a given propositional formula. This is the first time that parallelism is really used in AGC and that an algorithm is proposed to solve directly hard problems (without simulating another model).

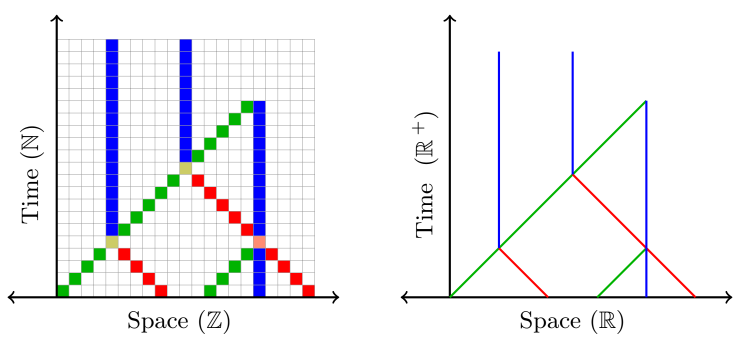


Fig. 1. From cellular automata to signal machines.

To achieve this, we follow a fractal pattern to a depth of n (for n propositional variables) in order to partition the space in 2^n regions corresponding to the 2^n possible valuations of the formula. We call the resulting geometrical construction the *combinatorial comb* of propositional assignments. With a signal machine, such an exponential construction fits in bounded space and time regardless of the number of variables.

This constant time has to be understood in the context of continuous space and time. Feynman famously remarked that “there’s plenty of room at the bottom.” This is especially the case here where, by scaling things down, room can be provided anywhere. With proper handling, this also leads to unbounded acceleration [Durand-Lose, 2009a]. The fractal pattern provides a way to automatically scale down. The one implemented here is a recursive subdivision in halves.

Once the combinatorial comb is in place, it is used to implement a binary decision tree for evaluating the formula, where all branches are explored in parallel. Finally, all the results are collected and disjunctively aggregated to yield the final answer.

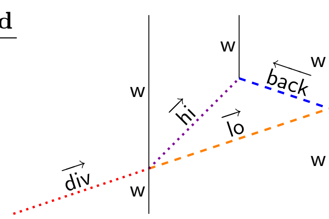
Signal machines are presented in Section 2. Sections 3 to 7 detail step by step our algorithm to solve SAT by geometrical computation: splitting the space, coding and generating the formula, broadcasting it, evaluating it and finalizing the answer by collecting the evaluations. Complexities are discussed in Section 8 and conclusion and remarks are gathered in Section 9.

2 Definitions

Signal machines are an extension of cellular automata from discrete time and space to continuous time and space. Dimensionless signals/particles move along the real line and rules describe what happens when they collide.

Signals. Each *signal* is an instance of a *meta-signal*. The associated meta-signal defines its *velocity* and what happen when signals meet. Figure 2 presents a very simple space-time diagram. Time is increasing upwards and the meta-signals are indicated as labels on the signals. Existing meta-signals are listed on the left of Fig. 2.

Meta-Signals	**Speed**
w	0
$\overrightarrow{\mathsf{div}}$	3
$\overrightarrow{\mathsf{hi}}$	1
$\overrightarrow{\mathsf{lo}}$	3
$\overleftarrow{\mathsf{back}}$	-3



Collision rules

$\{\ \mathsf{w},\ \overrightarrow{\mathsf{div}}\ \} \rightarrow \{\ \mathsf{w},\ \overrightarrow{\mathsf{hi}},\ \overrightarrow{\mathsf{lo}}\ \}$

$\{\ \overrightarrow{\mathsf{lo}},\ \mathsf{w}\ \} \rightarrow \{\ \overleftarrow{\mathsf{back}},\ \mathsf{w}\ \}$

$\{\ \overrightarrow{\mathsf{hi}},\ \overleftarrow{\mathsf{back}}\ \} \rightarrow \{\ \mathsf{w}\ \}$

Fig. 2. Geometrical algorithm for computing the middle

Generally, we use over-line arrows to indicate the direction of propagation of a meta-signal. For example, $\overleftarrow{\mathsf{a}}$ and $\overrightarrow{\mathsf{a}}$ denotes two different meta-signals; but as can be expected, they have similar uses and behaviors. Similarly $\mathsf{b_r}$ and $\mathsf{b_l}$ are different; both are stationary, but one is meant to be the version for right and the other for left.

Collision rules. When a set of signals collide, they are replaced by a new set of signals according to a matching collision rule. A rule has the form:

$$\{\sigma_1, \ldots, \sigma_n\} \rightarrow \{\sigma'_1, \ldots, \sigma'_p\}$$

where all σ_i are meta-signals of distinct speeds as well as σ'_j (two signals cannot collide if they have the same speed and outcoming signals must have different speeds). A rule matches a set of colliding signals if its left-hand side is equal to the set of their meta-signals. By default, if there is no exactly matching rule for a collision, the behavior is defined to regenerate exactly the same meta-signals. In such a case, the collision is called *blank*. Collision rules can be deduced from space-time diagram as on Fig. 2. They are also listed on the right of this figure.

Signal machine. A signal machine is defined by a set of meta-signals, a set of collision rules, and and initial configuration, i.e. a set of particles placed on the real line. The evolution of a signal machine can be represented geometrically as a *space-time diagram*: space is always represented horizontally, and time vertically, growing upwards. The geometrical algorithm displayed in Fig. 2 computes the middle: the new w is located exactly half way between the initial two w.

3 Combinatorial comb

In order to determine by brute force whether a propositional formula with n variables is satisfiable, 2^n cases must be considered. These cases can be recursively enumerated using a binary decision tree. In this section, we explain how to construct in parallel the full decision tree in constant space and time. This is done for a fixed formula, so that n is a constant, and the construction of the signal machine depends on it. In later sections we will use this tree to evaluate the formula.

The intuition is that the decision for variable x_i will be represented by a stationary signal: the space on the left should be interpreted as $x_i = \texttt{false}$, and the space on the right as $x_i = \texttt{true}$. Then we will similarly subdivide the spaces to the left and to the right, with stationary signals for x_{i+1}, and so on recursively for all variables as illustrated in Fig. 3(a).

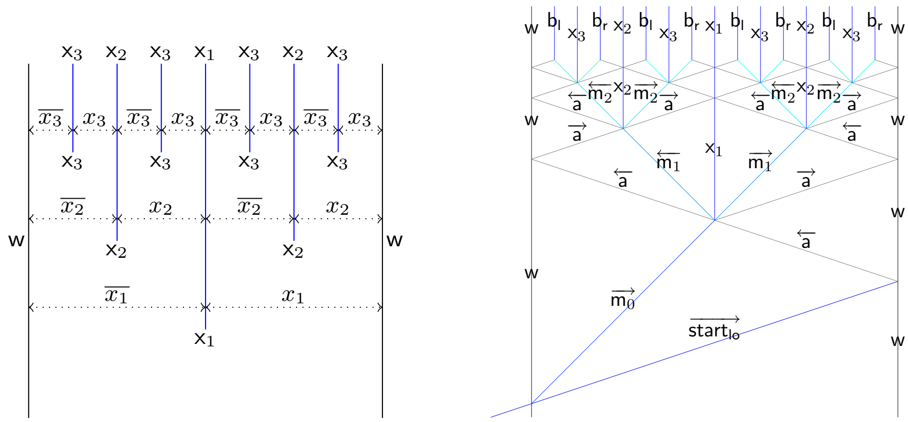
Starting with two bounding signals w and an initiator $\overrightarrow{\mathsf{start}}$, space is recursively divided as shown in Fig. 3(b). The first step works exactly as in Fig. 2, but then continues on to a depth of n: the counting is realized by using successively $\overrightarrow{\mathsf{m}_0}$, $\overrightarrow{\mathsf{m}_1}$, $\overrightarrow{\mathsf{m}_2}$ ... The necessary rules and meta-signals are summarized in Tab. 1

Since each level of the tree is half the height of the previous one, the full tree can be constructed in bounded time regardless of its size. Also, note that the bottom level of the tree is not x_n but $\mathsf{b_r}$ and $\mathsf{b_l}$. These are used both to evaluate the formula and to aggregate the results as explained later.

4 Formula encoding

In this section, we will explain how to represent the formula as a set of signals. This is illustrated with the following example:

$$\phi = (x_1 \vee \neg x_2) \wedge x_3$$



(a) Cases identification (b) Division process

Fig. 3. Combinatorial comb.

Meta-Signal	Speed
$\overrightarrow{\mathsf{start}}$, $\overrightarrow{\mathsf{start_{lo}}}$, $\overrightarrow{\mathsf{a}}$	3
$\overrightarrow{\mathsf{m}_0}$, $\overrightarrow{\mathsf{m}_1}$, $\overrightarrow{\mathsf{m}_2}$ ...	1
x_1, x_2, x_3 ...	0
$\overleftarrow{\mathsf{m}_0}$, $\overleftarrow{\mathsf{m}_1}$, $\overleftarrow{\mathsf{m}_2}$ ...	-1
$\overleftarrow{\mathsf{a}}$	-3
$\mathsf{b_l}$, $\mathsf{b_r}$	0

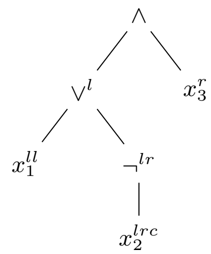
Collision rules
$\{\ \overrightarrow{\mathsf{start}},\ \mathsf{w}\ \} \rightarrow \{\ \mathsf{w},\ \overrightarrow{\mathsf{start_{lo}}},\ \overrightarrow{\mathsf{m}_0}\ \}$
$\{\ \overrightarrow{\mathsf{start_{lo}}},\ \mathsf{w}\ \} \rightarrow \{\ \overleftarrow{\mathsf{a}},\ \mathsf{w}\ \}$
$\{\ \mathsf{w},\ \overleftarrow{\mathsf{a}}\ \} \rightarrow \{\ \mathsf{w},\ \overrightarrow{\mathsf{a}}\ \}$
$\{\ \overrightarrow{\mathsf{a}},\ \mathsf{w}\ \} \rightarrow \{\ \overleftarrow{\mathsf{a}},\ \mathsf{w}\ \}$
$\{\ \overrightarrow{\mathsf{m}_i},\ \overleftarrow{\mathsf{a}}\ \} \rightarrow \{\ \overleftarrow{\mathsf{a}},\ \overleftarrow{\mathsf{m}_{i+1}},\ \mathsf{x}_i,\ \overrightarrow{\mathsf{m}_{i+1}},\ \overrightarrow{\mathsf{a}}\ \}$
$\{\ \overrightarrow{\mathsf{a}},\ \overleftarrow{\mathsf{m}_i}\ \} \rightarrow \{\ \overleftarrow{\mathsf{a}},\ \overleftarrow{\mathsf{m}_{i+1}},\ \mathsf{x}_i,\ \overrightarrow{\mathsf{m}_{i+1}},\ \overrightarrow{\mathsf{a}}\ \}$
$\{\ \overrightarrow{\mathsf{m}_n},\ \overleftarrow{\mathsf{a}}\ \} \rightarrow \{\ \mathsf{b_r}\ \}$
$\{\ \overrightarrow{\mathsf{a}},\ \overleftarrow{\mathsf{m}_n}\ \} \rightarrow \{\ \mathsf{b_l}\ \}$

Table 1. Meta-Signals and collision rules to build the comb.

A formula can be viewed as a tree whose nodes are labeled by symbols (connectives and variables). The evaluation of the formula for a given assignment is a bottom-up process that percolates from the leaves toward the root. In order to model that process, we shall represent each node of the tree by a signal. In Fig. 4(a), each node is additionally decorated with a *path* from the root uniquely identifying its position in the tree: thus we are able to conveniently distinguish multiple occurrences of the same symbol. These decorated symbols provide convenient names for the required meta-signals (see Fig. 4(b)). Thus a formula of size l requires the definition of $2l$ meta-signals.

The signals for all subformulae are sent along parallel trajectories and form a *beam*. They are stacked in the diagram in order of nesting, inner-most subformulae first. This order is important for the process of percolation that will take place at the end.

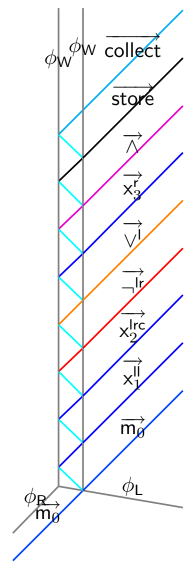
The process can be initiated by just 3 signals as shown in Fig. 4(c). The delay between the two signals from the left, $\overrightarrow{\mathsf{m}_0}$ and ϕ_{R}, controls the width of the beam. Since space is continuous, this width can be made as small as desired.



(a) Labeled tree

Meta-Signal	Speed
$\overleftarrow{\wedge}$, $\overleftarrow{\vee^{\mathsf{l}}}$, $\overleftarrow{\neg^{\mathsf{lr}}}$, $\overleftarrow{\mathsf{x}_1^{\mathsf{ll}}}$, $\overleftarrow{\mathsf{x}_2^{\mathsf{lrc}}}$, $\overleftarrow{\mathsf{x}_3^{\mathsf{r}}}$	-1
$\overrightarrow{\wedge}$, $\overrightarrow{\vee^{\mathsf{l}}}$, $\overrightarrow{\neg^{\mathsf{lr}}}$, $\overrightarrow{\mathsf{x}_1^{\mathsf{ll}}}$, $\overrightarrow{\mathsf{x}_2^{\mathsf{lrc}}}$, $\overrightarrow{\mathsf{x}_3^{\mathsf{r}}}$	1

(b) Generated signals



(c) Initial displaying

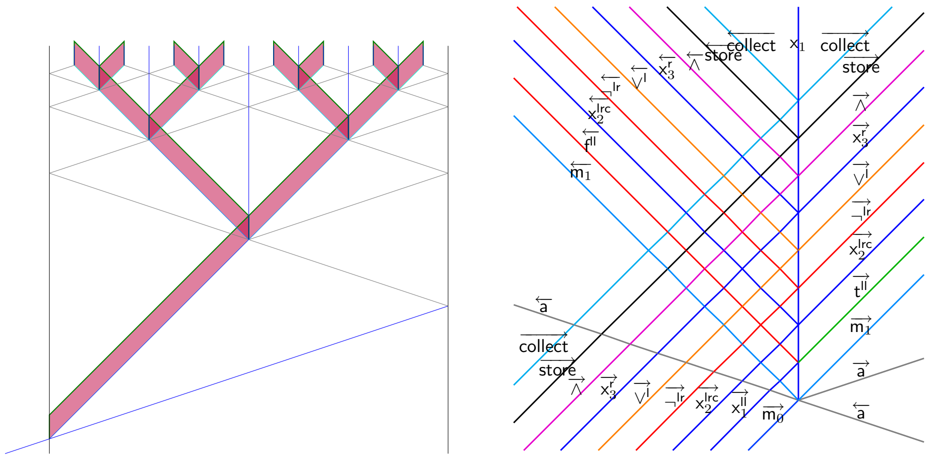
Fig. 4. Compiling the formula

5 Propagating the beam

The formula's beam is now propagated down the decision tree. For each decision point, the beam is duplicated: one part goes through, the other is reflected. Thus, by construction, every branch of the beam tree encounters a decision point for every variable at least once. If the beam is sufficiently narrow, the guarantee becomes "exactly once," as shown in Fig. 5(a). Although we lack space for a detailed explanation, it can easily be verified that emitting ϕ_{L} from the origin with a speed of $1 - 7/(3k \cdot 2^{n+2})$ is more than sufficient, where k is the number of signals in the beam and n is the number of variables in the formula. This rational number can be computed in time at most quadratic in the size of the formula.

When the beam encounters a decision point (a stationary signal for a variable x_i), then a split occurs producing two branches. Except for the sign of their velocity, most signals remain identical in both branches; most, except those corresponding to occurrences of x_i: those become `false` in the left branch and `true` in the right branch. Fig. 5(b) shows the beam intersecting the decision signal for variable x_1. Note how the incident signal $\overrightarrow{\mathsf{x}_1^{\mathsf{ll}}}$ becomes $\overleftarrow{\mathsf{f}^{\mathsf{ll}}}$ on the left and $\overrightarrow{\mathsf{t}^{\mathsf{ll}}}$ on the right; the path decoration is preserved since, as we shall see, it is essential later for the percolation process. This is achieved by the collision rule:

$$\{\overrightarrow{\mathsf{x}_1^{\mathsf{ll}}}, \mathsf{x}_1\} \to \{\overleftarrow{\mathsf{f}^{\mathsf{ll}}}, \mathsf{x}_1, \overrightarrow{\mathsf{t}^{\mathsf{ll}}}\}$$



(a) Corridors (b) Split

Fig. 5. Propagating the formula's beam

Since a decision point is encountered exactly once for each variable on each branch of the beam tree, at the bottom of the tree, all signals corresponding to occurrences of variables have been assigned a boolean value.

6 Evaluating the formula

Remember how, at the very bottom of the decision tree, we added an extra division using signals $\mathsf{b_l}$ or $\mathsf{b_r}$: their purpose is to initiate the percolation process. $\mathsf{b_l}$ is for starting the percolation process of a left branch, while $\mathsf{b_r}$ is for a right branch. Fig. 6 zooms on one case of our example: The invariant is that all signals that reach $\mathsf{b_r}$ have determined boolean values. When $\overrightarrow{\mathsf{t}^{\mathsf{ll}}}$ reaches $\mathsf{b_r}$, it gets reflected as $\overleftarrow{\mathsf{T}^{\mathsf{ll}}}$. The change from lowercase to uppercase indicates that the subformula's signal is now able to interact with the signal of its parent connective. The stacking order ensures that reflected signals of subformulae will interact with the incoming signal of their parent connective before the latter reaches $\mathsf{b_r}$. This enforces the invariant.

A connective is evaluated by colliding with the (uppercased) boolean signals of its arguments. For example, the disjunction collides with its first argument. Depending on its value, it becomes the one-argument function identity or the constant `true`. This is the way the rules of Tab. 2 should be understood.

Note how the path decorations are essential to ensure that the right subformulae interact with the right occurrences of connectives. Conjunctions and negations can be handled similarly. Finally, $\overrightarrow{\mathsf{store}}$ projects the truth value of the formula's root on $\mathsf{b_r}$ where it is temporarily stored until $\overrightarrow{\mathsf{collect}}$ starts the aggregation of the results.

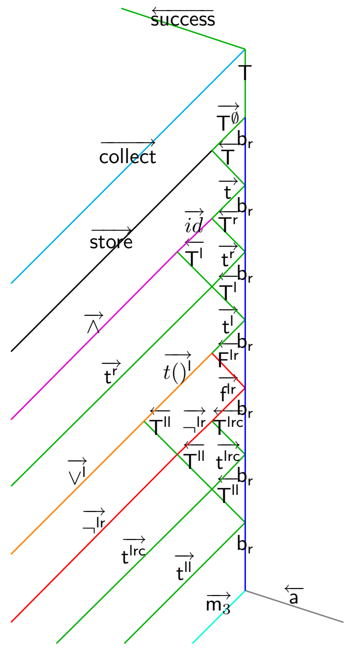


Fig. 6. Evaluation at the bottom of the comb.

$\{\ \overrightarrow{\vee^{\mathsf{l}}},\ \overleftarrow{\mathsf{T}^{\mathsf{ll}}}\ \} \to \{\ \overrightarrow{t()^{\mathsf{l}}}\ \}$	$\{\ \overrightarrow{t()^{\mathsf{l}}},\ \overleftarrow{\mathsf{T}^{\mathsf{lr}}}\ \} \to \{\ \overrightarrow{\mathsf{t}^{\mathsf{l}}}\ \}$	$\{\ \overrightarrow{id^{\mathsf{l}}},\ \overleftarrow{\mathsf{T}^{\mathsf{lr}}}\ \} \to \{\ \overrightarrow{\mathsf{t}^{\mathsf{l}}}\ \}$
$\{\ \overrightarrow{\vee^{\mathsf{l}}},\ \overleftarrow{\mathsf{F}^{\mathsf{ll}}}\ \} \to \{\ \overrightarrow{id^{\mathsf{l}}}\ \}$	$\{\ \overrightarrow{t()^{\mathsf{l}}},\ \overleftarrow{\mathsf{F}^{\mathsf{lr}}}\ \} \to \{\ \overrightarrow{\mathsf{t}^{\mathsf{l}}}\ \}$	$\{\ \overrightarrow{id^{\mathsf{l}}},\ \overleftarrow{\mathsf{F}^{\mathsf{lr}}}\ \} \to \{\ \overrightarrow{\mathsf{f}^{\mathsf{l}}}\ \}$

Table 2. Collision rules to evaluate the disjunction $\vee^l$.

7 Collecting the results

At the end of the propagation phase, the results of evaluating the formula for all possible assignments have been stored as stationary signals replacing the $\mathsf{b_l}$ and $\mathsf{b_r}$ signals. We must now compute the disjunction of all these results. This is the collection phase and it is initiated and carried out by signals $\overleftarrow{\mathsf{collect}}$ and $\overrightarrow{\mathsf{collect}}$ as illustrated in Fig. 7. The required collision rules are summarized in Tab. 3.

$\{\ B,\ \overrightarrow{\mathsf{collect}}\ \} \to \{\ \overleftarrow{B}\ \}$	$\{\ \overrightarrow{\mathsf{collect}},\ \mathsf{x}_i\ \} \to \{\ \overleftarrow{\mathsf{collect}},\ \mathsf{L},\ \overrightarrow{\mathsf{collect}}\ \}$
$\{\ B,\ \overleftarrow{\mathsf{collect}}\ \} \to \{\ \overrightarrow{B}\ \}$	$\{\ \mathsf{x}_i,\ \overleftarrow{\mathsf{collect}}\ \} \to \{\ \overleftarrow{\mathsf{collect}},\ \mathsf{R},\ \overrightarrow{\mathsf{collect}}\ \}$
$\{\ \overrightarrow{B_1},\ \mathsf{R},\ \overleftarrow{B_2}\ \} \to \{\ \overrightarrow{B_3}\ \}$	for $B, B_1, B_2, B_3 \in \{\mathsf{T}, \mathsf{F}\}$
$\{\ \overrightarrow{B_1},\ \mathsf{L},\ \overleftarrow{B_2}\ \} \to \{\ \overleftarrow{B_3}\ \}$	and $B_3 = B_1 \vee B_2$

Table 3. Collection rules

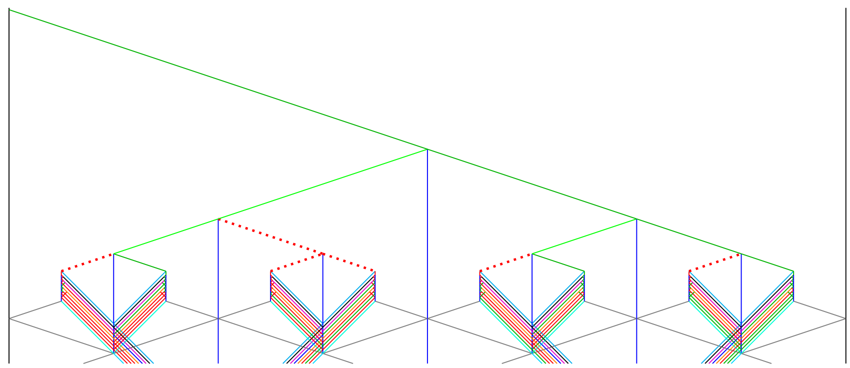


Fig. 7. Collecting the result.

Putting it all together, we get the space-time diagram of Fig. 8. Although, it cannot be seen on the picture, four signals are emitted on the first collision (bottom left). Two have very close speeds so that when the signals for the formula are generated, the resulting beam is sufficiently narrow (see Fig. 4(c) for a zoom in on this part).

8 Complexities

We now turn to a crucial question: what is the complexity of our construction as a function of the size of the formula? What is a meaningful way to measure this complexity?

The width of the construction measures the space requirement: it is independent of the formula and can be fixed to any value we like. The height measures the time requirement: it is also independent of the formula because of the fractal construction and the continuity of space-time. If more variables are involved, the comb gains extra levels, but its height remains bounded by the fractal.

As a consequence, while width (space) and height (time) are the natural continuous extensions of traditional complexity measures used in the discrete universe of cellular automata, in the context of abstract geometrical computations, they loose all pertinence.

Instead we should regard our construction as a computational device transforming inputs into outputs. The inputs are given by the initial state of the signal machine at the bottom of the diagram. The output is the computed result that comes out at the top. The transformation is performed in parallel by many threads: a thread here is an ascending path through the diagram from an input to the output. The operations that are "performed" by the thread are all the collisions found along the path.

Thus, if we view the diagram as an acyclic graph of collisions (vertices) and signals (arcs), the time complexity can then be defined as the maximal length

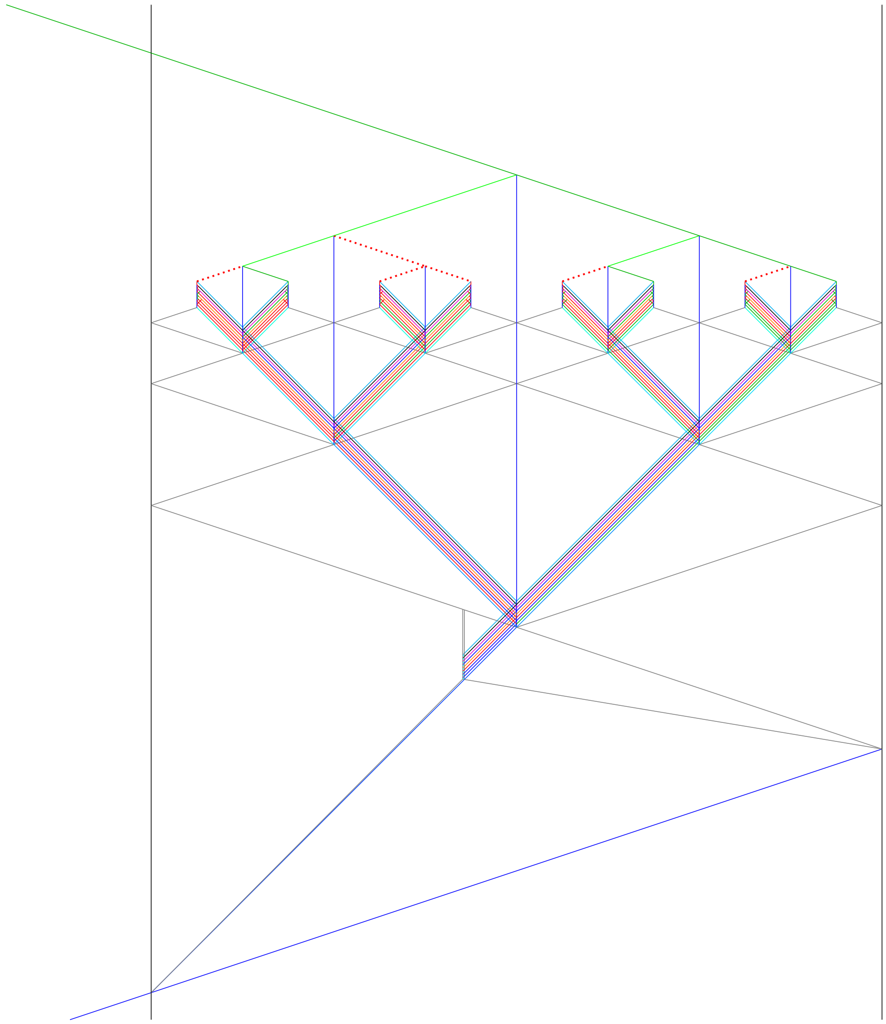


Fig. 8. The whole diagram.

of a chain and the space complexity can be defined as the maximal length of an anti-chain.

Let t be the size of the formula and n the number of variables. At the bottom level of the comb, there is an anti-chain of length approximately $t2^n$. The space complexity is exponential.

Generation of the comb, initiation, propagation, evaluation and aggregation contribute along any path a number of collisions at most linear in the size of the formula. However, intersections of incident and reflected branches at every level add $O(nt)$ because there are $O(n)$ levels and the beam consists of $O(t)$ parallel signals. Thus the time complexity is $O(nt)$.

It should also be pointed out that the signal machine depends on the formula but the compilation of the formula into a rational signal machine is done in polynomial time, presicely in quadratic time. The size of the generated signal machine is as follows. The number of meta-signals is linear in n for the comb and in t for the formula. The number of non blank collision rules is proportional to nt (each node of the formula is split on each variable). Counting the blank collision rules, it sums up to t^2. There are only seven distinct speeds: -6, -3, -1, 0, 1, 3 and one special rational value for the initiation.

9 Conclusion

In this article, we have shown how to achieve massive parallelism with signal machines, by means of a fractal pattern. We call this *fractal parallelism* and it is a novel contribution in the field of abstract geometrical computation.

Our approach is able to solve SAT, and thus any NP-problem, in bounded space and time by a methodic use of continuity. It does so while respecting the principle that everywhere the density of information is finite and its speed is bounded; a principle typically not considered by other abstract models of computation.

The complexity is not hidden inside the compilation of the machine nor in the initial configuration. Admittedly, the "magic" rational velocity used to control the narrowness of the beam constitutes an infelicity of presentation as it is the only one that depends on the formula. It can be eliminated using a slightly more involved beam-narrowing technique, but that extension is beyond the scope of the present article.

Since, clearly, time and space are no longer appropriate measures of complexity, we have also proposed to replace them respectively by the maximum length of a chain and an anti-chain in the space-time diagram regarded as a directed acyclic graph. According to these new definitions, our construction has exponential space complexity and quadratic time complexity. The compilation of formulae into signal machines can be done uniformly in quadratic time by a single classical machine.

We are currently furthering this research along two axes. First, we are considering how to tackle other complexity classes such as PSPACE, #P or EXPTIME using abstract geometrical computation. Second, we would like to design a generic signal machine for SAT, *i.e.* a single machine solving any instance of SAT, where the formula is merely compiled into an initial configuration.

Bibliography

Eugene Asarin and Oded Maler. Achilles and the Tortoise climbing up the arithmetical hierarchy. In *FSTTCS '95*, number 1026 in LNCS, pages 471–483, 1995.

Olivier Bournez. Some bounds on the computational power of piecewise constant derivative systems (extended abstract). In *ICALP '97*, number 1256 in LNCS, pages 143–153, 1997.

Todd A. Brun. Computers with closed timelike curves can solve hard problems efficiently. *Foundations of Physics Letters*, 16:245–253, 2003.

Stephen A. Cook. The complexity of theorem proving procedures. In *3rd Symposium on Theory of Computing (STOC 71)*, pages 151–158. ACM, 1971.

Jérôme Durand-Lose. Abstract geometrical computation: Turing computing ability and undecidability. In B.S. Cooper, B. Löwe, and L. Torenvliet, editors, *New Computational Paradigms, 1st Conf. Computability in Europe (CiE '05)*, number 3526 in LNCS, pages 106–116. Springer, 2005.

Jérôme Durand-Lose. Abstract geometrical computation with accumulations: Beyond the Blum, Shub and Smale model. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *Logic and Theory of Algorithms, 4th Conf. Computability in Europe (CiE '08) (abstracts and extended abstracts of unpublished papers)*, pages 107–116. University of Athens, 2008.

Jérôme Durand-Lose. Abstract geometrical computation 3: Black holes for classical and analog computing. *Nat. Comput.*, 8(3):455–572, 2009a.

Jérôme Durand-Lose. Abstract geometrical computation and computable analysis. In J.F. Costa and N. Dershowitz, editors, *International Conference on Unconventional Computation 2009 (UC '09)*, number 5715 in LNCS, pages 158–167. Springer, 2009b.

Gábor Etesi and Istvá Németi. Non-turing computations via Malament-Hogarth space-time. *International Journal of Theoret. Physics*, 41:341–370, 2002.

Ulrich Huckenbeck. Euclidian geometry in terms of automata theory. *Theoret. Comp. Sci.*, 68(1):71–87, 1989.

Giuseppe Jacopini and Giovanna Sontacchi. Reversible parallel computation: an evolving space-model. *Theoret. Comp. Sci.*, 73(1):1–46, 1990.

Leonid Levin. Universal search problems. In *Problems of Information Transmission.*, pages 265–266, 1973.

Maurice Margenstern and Kenichi Morita. NP problems are tractable in the space of cellular automata in the hyperbolic plane. *Theor. Comp. Sci.*, 259 (1-2):99–128, 2001.

Thomas J. Naughton and Damien Woods. On the computational power of a continuous-space optical model of computation. In M. Margenstern, editor, *Machines, Computations, and Universality (MCU '01)*, number 2055 in LNCS, pages 288–299, 2001.

Gheorghe Păun. P systems with active membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.

Petr Sosík. Solving a PSPACE-Complete problem by P-systems with active membranes. In M. Cavaliere, C. Martín-Vide, and G. Păun, editors, *Brainstorming Week on Membrane Computing*, pages 305–312. Tarragona: Universidad Rovira i Virgili, 2003.