Jérôme Durand-Lose and Aurélien Emmanuel. Abstract geometrical computation 11: Slanted firing squad synchronisation on signal machines. *Theoret Comp Sci*, 894:103–120, 2021. doi: 10.1016/j.tcs.2021.06.009. arXiv 2106.11176.

# Abstract Geometrical Computation 11: Slanted Firing Squad Synchronisation on Signal Machines

Jérôme Durand-Lose[a,*], Aurélien Emmanuel[a]

[a] *Université d'Orléans, INSA Centre Val de Loire, LIFO EA 4022, FR-45067 Orléans, France*

**Abstract**

Firing Squad Synchronisation on Cellular Automata is the dynamical synchronisation of finitely many cells without any prior knowledge of their range. This can be conceived as a signal with an infinite speed. Most of the proposed constructions naturally translate to the continuous setting of signal machines and generate fractal figures with an accumulation on a horizontal line, i.e. synchronously, in the space-time diagram. Signal machines are studied in a series of articles named Abstract Geometrical Computation.

In the present article, we design a signal machine that is able to synchronise/accumulate on any non-infinite slope. The slope is encoded in the initial configuration. This is done by constructing an infinite tree such that each node computes the way the tree expands.

The interest of Abstract Geometrical computation is to do away with the constraint of discrete space, while tackling new difficulties from continuous space. The interest of this paper in particular is to provide basic tools for further study of computable accumulation lines in the signal machine model.

*Keywords:* Abstract Geometrical Computation ; Cellular Automata ; Divide and Conquer ; Firing Squad Synchronisation ; Fractal ; Signal Machines.

## 1. Introduction

The Firing Squad Synchronisation Problem (FSSP) is as follows: a general wants a whole line of riflemen to fire synchronously. However, communication is only between very close individuals and the number of riflemen is unknown. There is no global time, but everyone receives ticks simultaneously and repeatedly. Moreover, it is supposed that the number of states of each rifleman is finite. Finite state, discrete space and time, synchrony and uniformity correspond to the Cellular Automata (CA) model. In this context, the goal is, starting from a single activated cell, to reach a configuration in which all cells of a given region are in a special firing state simultaneously and for the first time.

This problem have been studied from the 1960's [9, 17, 20], with a wide range of construction [3, 15, 16] and is still active nowadays [10, 11, 12, 13, 19, 21]. In [15], a six-state minimal time solution is given. Solutions to this problem also often serve as a tool to solve other problems. In [3] a variation with two generals is used to solve a related complexity problem. In [18], the election leader problem, which can be seen as a reversed FSSP is solved efficiently for any finite dimension. In [14], FSSP is used on demand to synchronise computation steps.

The most common solutions involve signals sent at different speeds bouncing onto the edges of the line and sprouting new signals whenever crossing each other in such a way that signals eventually and simultaneously evenly fill the line which triggers the firing. A tree structure often appears in the construction. These trees

*Corresponding author

*Email addresses:* jerome.durand-lose@univ-orleans.fr (Jérôme Durand-Lose), aurelien.emmanuel@univ-orleans.fr (Aurélien Emmanuel)

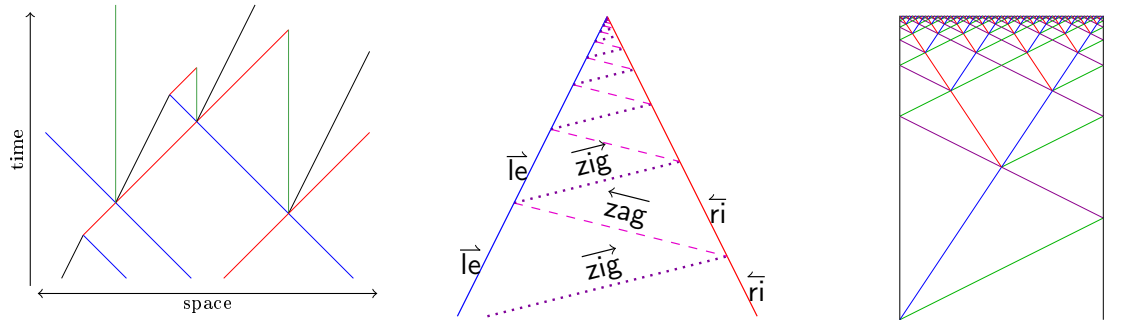
*URL:* http://www.univ-orleans.fr/lifo/Members/Jerome.Durand-Lose (Jérôme Durand-Lose)

are rooted in the general and the leaves reach the firing riflemen. Very commonly, the solutions are presented in a continuous setting, then adapted to the discrete space and time of CA; reaching the granularity of space (i.e. cells) indicates when to fire.

It appears natural to consider this problem in the continuous setting of Abstract geometrical computation: the study of computing in an euclidean space with geometrical tools. It is done using signal machines, which compute by drawing coloured lines which interact in specific ways upon collision. It lead to a series of articles, for example, isolated accumulations on single point are characterised in [7].

Signal machines are an abstraction of 1D CA: signals carrying a given label travel at a given speed through a continuous space; any collision between two or more signals results in the vanishing of incoming signals, and the emergence of new ones, according to their labels and to some predefined rules. Figure 1a provides some space-time diagram of an example signal machine (time elapses upward). Signal machines are defined by meta-signals that define signals, and collision rules, that define their interactions; for example, the space-time diagram in Fig. 1b was obtained with a machine with meta-signals $\overrightarrow{\mathsf{zig}}$, $\overleftarrow{\mathsf{zag}}$, $\overrightarrow{\mathsf{le}}$ and $\overleftarrow{\mathsf{ri}}$, as well as the collision rules $\{\overrightarrow{\mathsf{zig}},\overleftarrow{\mathsf{ri}}\} \to \{\overleftarrow{\mathsf{zag}},\overleftarrow{\mathsf{ri}}\}$ and $\{\overleftarrow{\mathsf{zag}},\overrightarrow{\mathsf{le}}\} \to \{\overrightarrow{\mathsf{zig}},\overrightarrow{\mathsf{le}}\}$.

Compared to CA, there is no underlying grid and only signal dynamics is addressed. The continuous time and space allow phenomena alien to CA: accumulation as illustrated in Fig. 1b: there are infinitely many collisions leading to the top of the triangle. It is already known that forecasting of any accumulation on rational signal machines is highly unpredictable [5] and the possible localisation of isolated accumulation of such machines have been characterised [8].




(a) space-time diagram (b) isolated accumulation (c) segment accumulation

Figure 1: Space-time diagrams and basic accumulations.

Most of the schemes used for FSSP in CA can be defined directly as signals machines (but going from there to a CA is no triviality [1]). In CA, the construction stops when the granularity of space is reached. In the continuous setting of signal machines, there is no such a thing so that the divide and conquer process never stops and generates a fractal. The result, a horizontal firing line in a space-time diagram, consists of an accumulation line as in Fig. 1c (each colour corresponds to one meta-signal, speed magnitudes are 0, 1 or 3, such that red and blue signals form an infinite binary tree). That is a line of points nearby which an infinity of signal collisions happen (the continuous equivalent of filling a discrete line with signals). Such a construction with a finite yet unbounded binary tree has been used in [4] to provide unbounded branching to solve PSPACE-complete problems with polynomial depth.

The accumulation line in the space-time diagram needs not be horizontal although there are not much research on the matter since, in the more usual discrete settings, synchronisation is the goal and slope is less meaningful. Achieving a given slope would correspond to riflemen firing in succession at a predefined rate. Much like the regular FSS problem is about synchrony, our Slanted FSS problem is about coordination. Its solution could be used to as a tool for problems requiring tasks to be done in a quick succession—too quick for the end of a task to trigger the beginning of the next one; for example sending self driving cars through an intersection with shorter security distances than possible using uncoordinated human drivers.

In the present paper, we prove that not only it is possible to generate any segment of any non-infinite slope as an accumulation set but moreover that they can all be generated by the same signal machine.

Without loss of generality, we aim at segment with extremities at abscissa $-1$ and $+1$. The target segment is encoded in the initial configuration: the ordinates (the times) of the extremities are encoded as distance between signals.

The idea is to draw an infinite unary-binary tree in a finite amount of space, halving the size of edges after each node with two children. The structure of the tree dictates the shape of the accumulation set. As in [4], the tree structure conveys some information used to extend the tree: with sign test, addition and subtraction (on real numbers) primitives, each node computes whether to delay or split and updates and broadcasts the information. These primitives are also part of the linear version of the Blum, Shub and Smale model of computation [2] witch is related to Signal Machines [6].

Unary-binary tree is one of the many possible constructions that can fulfil our goal, we chose it for its simplicity, with complete disregard for time minimality or optimisation of the number of meta-signals or collision rules - corresponding notions to states and rule of a cellular automaton.

The algorithm is first implemented on an *augmented signal machine* to allow the signals in the tree to carry an unlimited quantity of information. This allows to concentrate on the algorithm without dealing with technical details. Then these augmented signals are encoded as a ray of signals on an ordinary signal machine and needed arithmetic operations are implanted. This is much more involving because it has to deal with updating the information now encoded with only finitely many possible signals.

The paper is organised as follows. The relevant definitions are provided in Sect. 2. The algorithm is explained in Sect. 3 and implemented on augmented signal machines in Sect. 4. This is then implemented on ordinary signal machine in Sect. 5. Section 6 concludes the paper.

## 2. Definitions

### *2.1. Signal machines*

A signal machine regroups the definitions of its meta-signals and their dynamics: constant speed outside of collisions and rewriting rules at collisions.

A *signal machine* is a triplet $(M, S, R)$ such that:

- $M$ is a finite set, whose elements are called *meta-signals*;
- $S : M \to \mathbb{R}$ is the *speed function* (each meta-signal has a constant speed);
- $R$ is a finite set of *collision rules*; a collision rule $\rho$ can be written $\rho^- \to \rho^+$, and consists in an *input set* $\rho^-$ and an *output set* $\rho^+$ of meta-signals of distinct speeds, with $\rho^-$ containing at least two meta-signals. $R$ is deterministic: $\rho \neq \rho'$ implies that $\rho^- \neq \rho'^-$.

In the example in Fig. 1b, there are four meta-signals: $\overleftarrow{\mathsf{zag}}$, $\overleftarrow{\mathsf{ri}}$, $\overrightarrow{\mathsf{le}}$, and $\overrightarrow{\mathsf{zig}}$ of speeds $-1$, $-1/2$, $1/2$, and $1$ respectively. The collision rules are $\{\overrightarrow{\mathsf{zig}},\overleftarrow{\mathsf{ri}}\} \to \{\overleftarrow{\mathsf{zag}},\overleftarrow{\mathsf{ri}}\}$ and $\{\overrightarrow{\mathsf{le}},\overleftarrow{\mathsf{zag}}\} \to \{\overrightarrow{\mathsf{le}},\overrightarrow{\mathsf{zig}}\}$.

A *configuration*, $c$, is a mapping from the points of the real line to either a meta-signal, a rule or the value $\oslash$ (indicating that there is nothing there). There are finitely many non-$\oslash$ locations in any *initial configuration*. There are 3 signals in the initial configuration in Fig. 1b, from left to right: $\overrightarrow{\mathsf{le}}$, $\overrightarrow{\mathsf{zig}}$, and $\overleftarrow{\mathsf{ri}}$.

A *space-time diagram* is the collection of configurations as time elapses. It forms a two dimensional picture (time is elapsing upwards in the figures). It is a function from $\mathbb{R} \times \mathbb{R}^+ \mapsto M \cup R \cup \{\oslash\}$.

A *signal* of a space-time diagram is a maximal segment or half-line mapped by the space-time diagram to a meta-signal $\mu$ and of inverse slope $S(\mu)$ (signals cannot be horizontal). Meta-signals can be thought of as type, and signals as instances of them. The meta-signal of a signal is also called its *type* for the sake of concision.

A space-time diagram follows the rules:

- any point of the space-time diagram that maps to a meta-signal $\mu$ belongs to a signal which verifies:
  - its starting location is either a rule $\rho$ with $\mu \in \rho^+$ or a point of the initial configuration mapping to $\mu$ and
  - its end, if any, is a rule $\rho$ with $\mu \in \rho^-$;
- any point of the space-time diagram with a collision of rule $\rho$ is at:
  - the starting end of one signal of type $\mu$ for every $\mu$ in $\rho^+$ and
  - the ending end of one signal of type $\mu$ for every $\mu$ in $\rho^-$ (if not in the starting configuration);

– there is no infinite time-backward continuous path/sequence of signals each of whose start is the end of the next.

This ensures that signals propagate with uniform speed, collision rules are properly applied and no signal nor collision appears out of the blue. In particular we do not want anything appearing from accumulation points as defined below.

### *2.2. Accumulation points*

In the example in Fig. 1b, there is a special point on top such that there are infinitely many signals and collisions leading to it. The signal machine does not provide any definition for what happens there so that the space-time diagram cannot be defined for all times and locations. We extend the definition as follows: a space-time diagram is a partial function from $\mathbb{R}\times\mathbb{R}^+$ to $M\cup R\cup\{\oslash,✺\}$ where ✺ is a new symbol introduced to designate accumulation points (and only such points). Outside of accumulation points, the constrains are as before but signals may also end in accumulations. Any accumulation point must be the forward limit of an infinite sequence of collision points.

The example in Fig. 1b is the simplest example of accumulation. There is only one point valued ✺; the rest of the space-time diagram is $\oslash$. In the example of Fig. 1c, a connected accumulation set (a line segment) is generated.

### *2.3. Augmented signal machines*

*Augmented Signal Machines (ASM)* are natural extensions of signal machine where signals are allowed to carry analog information.

An *augmented signal machine* is a triplet $(M, S, R)$ such that:

– $M$ is a finite set of *meta-signals*;
– $S : M \to \mathbb{R}$ is the *speed function*;
– each $\mu$ in $M$ has an associated set, called *domain* and noted $D_\mu$: signals may carry information depending on their type;
– the set of *augmented meta-signals* is $M' = \{(\mu, x)|\mu \in M, x \in D_\mu\}$. The speed of an augmented meta-signal is simply the speed of its corresponding meta-signal;
– $R$ is a set of *collision rules*; a collision rule $\rho$ can be written $\rho^- \to \rho^+$, and consists in an *input set* $\rho^-$ and an *output set* $\rho^+$ of augmented meta-signals of distinct speeds, with $\rho^-$ containing at least two augmented meta-signals. $R$ is deterministic.

Configurations, diagrams, augmented signals and accumulation points are defined the same way as for regular signal machines, replacing meta-signals by augmented meta-signals.

A signal machine is a special case of augmented signal machine where every $D_\mu$ is a singleton (or a finite set). In this paper, domains are either singletons or subsets of $\mathbb{R}^2$. We thus note augmented meta-signals $\mu$ when $D_\mu$ is a singleton, and otherwise we note $\mu_l^r$ instead of $(\mu, (l, r))$.

Since there are potentially infinitely many rules for an augmented signal machines, we define several of them at a time through parameterized patterns: $X \xrightarrow{C} Y$ where $X$ is a left member of a rule containing free variables, $C$ is a condition on them and $Y$ is a right member which might also depend on them. Concrete examples of rule pattern are given in Sect. 4.

### *2.4. Slanted Firing Squad Synchronization Problem*

The goal of the *Firing Squad Synchronization Problem* (FSSP) on signal machines is to design a machine and an initial configuration which create an horizontal line as the set of accumulation points of the corresponding space-time diagram.

Formally, a solution of the FSSP is a signal machine together with an initial configuration composed of two border signals, a sequence of signals —called the general— along with the prescribed distance between them near the left border. If the width of the general is small enough relative to distance between border signals, there exists a time $t_l$ such that, in the resulting space-time diagram, the set of accumulation point is the horizontal segment of equation $t = t_l$ —within the boundaries.

The goal of the *Slanted Firing Squad Synchronization Problem* (SFSSP) for signal machines is to design a machine and an initial configuration which creates a slanted line as the set of accumulation points of the corresponding space-time diagram. In other words, looking at the firing line over time, we want the firing to happen in succession, as an apparent dot moving at a constant, specified speed. Since space-time diagram is a central piece of signal machines, we prefer the image of and the vocabulary related to a slanted line in a space-time diagram over those of a virtual moving dot.

Formally, a *solution* of the Slanted FSSP of slope $\alpha$ is again a signal machine and an initial configuration, such that

- the initial configuration is composed of 2 static bounding signals (to delimit the firing line), one on the left border of the general, one far beyond the right border;
- for all big enough ratio of width between the line and the general, there exists a time $t_l$ such that, in the resulting space-time diagram, the set of accumulation points is the slanted line segment of equation $t = t_l + \alpha \times x$ within the boundaries.

Conversely, instead of aiming at a big enough line, we can shrink the general. This is our approach in the rest of the paper and we fix the boundaries at $-1$ and 1.

Additionally and informally, a *universal solution* to the SFSSP is a single signal machine and a way to encode any given slope $\alpha$ into a general so as to solve the corresponding SFSSP.

In this paper, instead of aiming at a slope $\alpha$, we aim at a segment $[(-1, l), (1, r)]$ with $l, r \geq 1$. This allows to solve for any slope, and gives perspective about drawing arbitrary segment or curve in the space-time diagram, while foregoing time-optimality.

## 3. Algorithm and ASM implementation

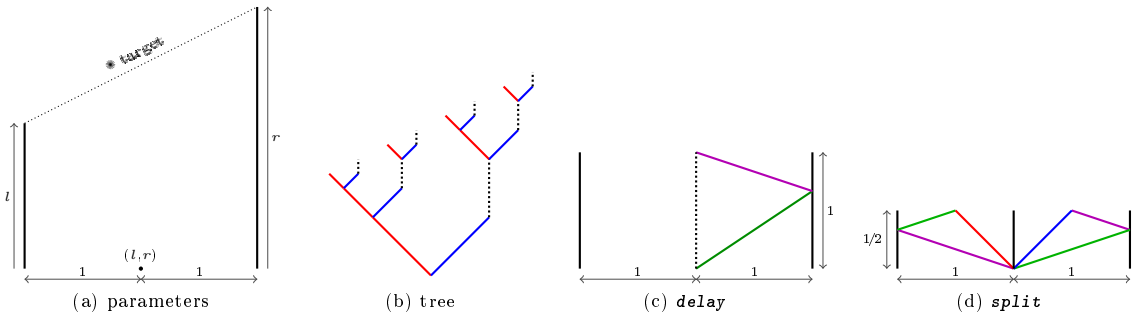
### *3.1. General scheme and algorithm*

The target segment and the parameters are depicted in Fig. 2a : parameters $l$ and $r$ are the ordinates of the extremities of the target segment. To be valid, the parameters have to be both at least 1.

The algorithm relies on a very simple divide and conquer strategy: to *draw* —accumulate on— the segment $[(-1, l), (1, r)]$, it suffices be able, depending on the situation, to either:

- draw the segment $[(-1, l-1), (1, r-1)]$ translated one unite of distance up, or
- draw the segments $[(-1, l-1/2), (0, (l+r)/2 - 1/2)]$ and $[(0, (l+r)/2 - 1/2), (1, r-1/2)]$ translated to have their roots at $(-1/2, 1/2)$ and $(1/2, 1/2)$ respectively. It is scaled by $1/2$ yielding the formula in Algo. 1.

Figure 2b illustrates this infinite recursion, with vertical segments drawn as dotted and black, and diagonal segments of slope 1 and $-1$ being drawn as red or blue, depending whether they go to the left or right (ascending) respectively. This colour code applies to section 4 as well. In the one entwined in Fig. 1c (looking only at red and blue signals), each node always have two descendants. The unary steps are used to provide delays and thus slope. Each (sub-)tree is bounded by two motionless signals as in figures 2c and 2d. When an update is carried out, the collision happens exactly at the middle of the bounds which are considered 2 units of space apart.




(a) parameters (b) tree (c) `delay` (d) `split`

Figure 2: Parameters, unary-binary tree, and elementary steps.

The following primitives are used to direct the dynamics in the algorithm:

`delay` $(l, r)$ : the current node is of degree 1. Its single child is 1 unit up and the algorithm is prompted with parameters $(l, r)$. In Fig. 2b a `delay` corresponds to a dotted connections and

`split` $((l, r),(l', r'))$ : the current node is of degree 2. The left child is at $(-1/2, 1/2)$ relatively to the current node, the right child at $(1/2, 1/2)$. In Fig. 2b, the links between the node and its children are red and blue. At each child, the algorithm is prompted with the scale reduce by a factor 2 and with the parameters $(l, r)$ for the left one, and $(l', r')$ for the right one.

Figure 2c outlines how a delay can be implemented by an augmented signal machine, with two auxiliary signals bouncing off the boundary at the correct speeds (3/2 for green and $-3$ for purple) to wait the correct amount of time. Figure 2d outlines how a split can be implemented, with auxiliary green signals of speed 3, and purple $-3$. Altogether, it yields the algorithm 1 where the tree is initialised with the "top level" $l$ and $r$ for the targeted final accumulation segment. The strategy is to delay whenever possible, i.e. when both extremities are at least 2 units above.

```
if 2 <= l and 2 <= r :                  # time += 1
    delay ( l - 1 , r - 1 )                     #          remove 1
else :                                  # time += 1/2
    split ( ( l + l - 1 , r + l - 1 ) ,         # add l then remove 1
            ( l + r - 1 , r + r - 1 ) )         # add r then remove 1
```

Algorithm 1: Code for augmented collision rules.

Figure 3 illustrates the update formula after a split: the new target heights are shorter by 1/2, and are multiplied by 2 to account for the new scale.

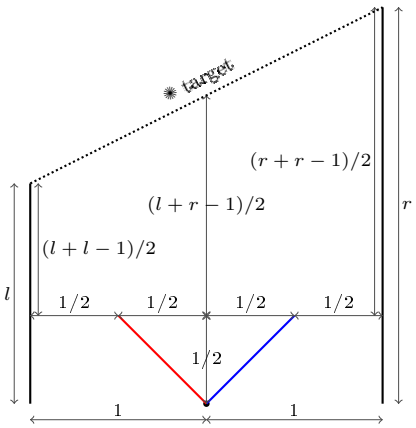



Figure 3: *Split* update formula illustrated.

Nodes are called `split` or `delay` depending on the used primitive. There is no end to the recursion: the constructed unary-binary tree is infinite to produce the whole accumulations segment.

### *3.2. Correctness*

The `split` nodes play a key role in the demonstration of the correctness of the algorithm. The *depth* of a collision is the number of its ancestors of degree 2 (i.e. `split` nodes). From now on, let $d$ denotes the depth of a given collision and $\alpha$ denotes the slope of the targeted segment, that is $(r_0 - l_0)/2$ where $(r_0, l_0)$ denotes the parameter at the root of the tree.

**Lemma 1** (Invariants)**.** *Algorithm 1 satisfies the following invariants:*
1. $1 \leq l$ *and* $1 \leq r$ *(the parameters remain valid);*
2. $(r - l)/2 = \alpha$ *(slope is preserved);*

3. *the boundary points of the current targeted segment are on the initially targeted segment: assume the initial parameters are $l_0$ and $r_0$, and consider a collision happening at $(x,t)$, with parameters $l$, $r$. The points $(x,t)+2^{-d}(-1,l)$ and $(x,t)+2^{-d}(+1,r)$ are on the initial targeted segment. That is the segment with extremities $(-1,l_0)$ and $(1,r_0)$*

*Proof.* The first two invariants are straightforward to prove from Algo. 1. Let us consider a node at $(x,t)$ with parameters $l,r$ (both at least 1). Let us prove the last invariant by induction. It is true for the root node.

After a `delay` node: $(x-2^{-d},(t+2^{-d})+2^{-d}(l-1)) = (x-2^{-d},t+2^{-d}l)$ and similarly for $r$: the targeted segment extremities are unchanged.

After a `split` node: $(x-2^{-(d+1)}-2^{-(d+1)},(t+2^{-(d+1)})+2^{-(d+1)}.2.(l-1/2)) = (x-2^{-d},t+2^{-d}l)$: the left extremity of the left new branch matches the left extremity of the formerly targeted segment. Since $(x-2^{-(d+1)}+2^{-(d+1)},(t+2^{-(d+1)})+2^{-(d+1)}.2.((l+r)/2-1/2) = (x,t+2^{-d}(l+r)/2)$, the right extremity of the left new branch matches the middle of the formerly targeted segment. Proof is the same for the right branch. □

**Lemma 2.** *A `split` collision at $(x,t)$ happens before the point on the targeted segment at the same spatial location, but not sooner than $2^{-d}(2+|\alpha|)$ before. In other words:*

$$t+2^{-d}(l+r)/2-2^{-d}(2+|\alpha|) \;\leq\; t \;\leq\; t+2^{-d}(l+r)/2 \;.$$

*Proof.* The second inequality is obvious. For the first one, we can observe that: $(l+r)/2 = \min\{l,r\}+|\alpha|$. Indeed, we have:

$$\begin{aligned} 2\min\{l,r\}+|r-l| &= 2\min\{l,r\}+\max\{l,r\}-\min\{l,r\} \\ &= \min\{l,r\}+\max\{l,r\} = l+r \;. \end{aligned}$$

We can then write:

$$t \;=\; t+2^{-d}(l+r)/2-2^{-d}(\min\{l,r\}+|\alpha|) \;.$$

Finally, since it is a `split` node, $\min\{l,r\}\leq 2$, yielding the desired inequality. □

Let us note that any infinite branch of the tree contains infinitely many `split`. There are always infinitely many branching on both sides.

**Theorem 3** (Correctness of the algorithm)**.** *Given inputs $l_0$ and $r_0$ no lesser than 1, the algorithm draws a tree whose closure is the segment $[(-1,l_0),(1,r_0)]$.*

*Proof.* Let $x$ be in $[-1,1]$. Our goal is to prove that there is an accumulation at $(x,(l_0+r_0)/2+\alpha x)$.

We remark that the set of abscissae of `split` nodes is exactly the set of dyadic rationals comprised strictly between $-1$ and 1, which is dense in $[-1,1]$. Indeed, these are the number written $\Sigma_1^d a_i 2^{-i}$ with the $a_i$s in $\{+1,-1\}$.

Thus, there is a sequence of nodes $(x_d,t_d)_{d\in\mathbb{N}}$ such that:

1. $(x_d,t_d)$ is a `split` collision point of depth $d$.
2. $(x_d)$ converges, with limit $x$. We can even take $|x_d-x|<2^{-d}$, and we do, for convenience.

$$\begin{aligned} |t_d-((l_0+r_0)/2+\alpha x)| &\leq |t_d-((l_0+r_0)/2+\alpha x_d)| \\ &\quad + |((l_0+r_0)/2+\alpha x_d)-((l_0+r_0)/2+\alpha x)| \\ &\leq |t_d-((l_0+r_0)/2+\alpha x_d)|+|\alpha(x_d-x)| \\ &\leq |t_d-((l_0+r_0)/2+\alpha x_d)|+|\alpha|\times 2^{-d} \end{aligned}$$

then, because of lemmas 2 and 1,

$$-2^{-d}(2+|\alpha|)\leq t_d-((l_0+r_0)/2+\alpha x_d)\leq 0$$

which implies

$$|t_d-((l_0+r_0)/2+\alpha x_d)|\leq 2^{-d}(2+|\alpha|) \;.$$

Putting things together, we have:

$$|t_d-(l_0+r_0)/2+\alpha x|\leq 2^{-d}(2+|\alpha|+|\alpha|)x \xrightarrow[d\to+\infty]{} 0 \;.$$

Proving that $(t_d)$ converges, with limit $(l_0{+}r_0)/2{+}\alpha x$. That proves $(x, (l_0{+}r_0)/2{+}\alpha x)$ is an accumulation point.

Points above the targeted segment are not accumulation points as no node is above it, as shows Lem. 2 (`delay` nodes have to have a `split` heir above them, eventually). For a point below the accumulated segment, say by a amount $\Delta t$, then, according to Lem. 2, there's a depth $d$ big enough beyond which nodes are no lower than $\Delta t/2$ below the accumulation segment.

All in all, the accumulation set is exactly the targeted segment. □

## 4. ASM implementation

Each augmented meta-signal that will translate into a ray of regular signals has its name written slanted, whereas 'regular' ones that translate directly into the SM version in next section, like border, are upright.

The initial configuration starts with borders at $-1$ and $+1$ as shown in Fig. 2a. As mentioned, each sub-tree is bounded by a pair of border signals. Each time, we have to consider three cases depending on how the node was generated (unary, left or right binary).

Figure 4 shows how a `delay` is implemented. It is initiated by a collision between a main (red, dotted black, or blue) which carries the values of $l$ and $r$ and an auxiliary signal (green or purple). At this collision, when both $l$ and $r$ are greater than 2, one augmented signal goes forth (green $\overrightarrow{bounce_{sl}}$, *sl* stands for slow) and back (purple $\overleftarrow{bounce}$) to the border on the right. The amount of time waited is directly proportional to the width of the bounds, and with proper speeds the delay can be made half the width as desired. These speeds are given in the list of meta-signals in Fig. 6.

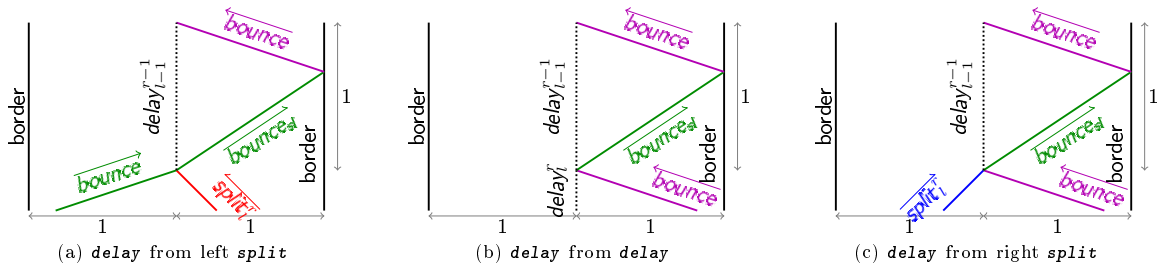



Figure 4: Nodes when $2 \leq l$ and $2 \leq r$.

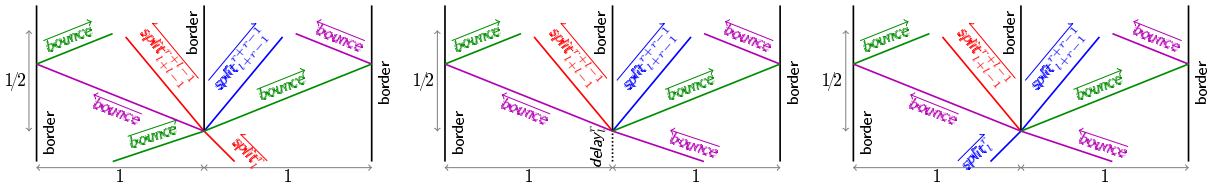
Auxiliary signals ($\overrightarrow{bounce}$ and the likes) bouncing off the borders is described by the collision rules in Fig. 7a. The last two rules deal with the case where two such signals bounce at the same point of the border from both sides.

The three rule patterns of Fig. 7b correspond to central collision in the three cases in Fig. 4. The left part of the rules corresponds to collisions happening at nodes (with respect to the simulated algorithm) prompted by a previous node accordingly -whether prompted by a `delay`, or either side of a `split`. The right parts of these rule patterns are the same: it is the signals necessary to perform a delay, i.e. to wait a unit of time.

Figure 5 shows how a `split` is carried out: once again with green and purple signals bouncing off the walls, this time intercepting split signals half a unit of time later and in the middle of the new boundaries. The border created by a `split` ensures both sides of the computation start in the middle of their own borders and are kept separated. It also effectively scales these subsequent computations down by a factor of two. The three rule patterns of Fig. 7c correspond to central collision in the three cases in Fig. 5.

## 5. SM implementation

It is first presented how bouncing signals are handled and how augmented signals are encoded as *macro-signals*: coherent packs of parallel signals (also seen as *rays*) that can store the analogue information. The update and routing is then presented in three stages. The first stage tests whether the (next) step is `delay`




(a) `split` from left `split` (b) `split` from `delay` (c) `split` from right `split`

Figure 5: Nodes when $l < 2$ or $r < 2$.

$s_{\text{bounce}} = 3$

$s^{\text{slow}}_{\text{bounce}} = 3/2$

| Meta-signal | Speed |
|---|---|
| border | 0 |
| $\overrightarrow{bounce_{sl}}$ | $s^{\text{slow}}_{\text{bounce}}$ |
| $\overleftarrow{bounce}$ | $-s_{\text{bounce}}$ |
| $\overrightarrow{bounce}$ | $s_{\text{bounce}}$ |

| Augmented Meta-signal | | Speed |
|---|---|---|
| $l, r \in [1, +\infty)$ | $delay^r_l$ | 0 |
| $l, r \in [1, +\infty)$ | $\overrightarrow{split^r_l}$ | 1 |
| $l, r \in [1, +\infty)$ | $\overleftarrow{split^r_l}$ | -1 |

Figure 6: Regular and augmented meta-signals.

$$\{\overrightarrow{bounce_{sl}}, \text{border}\} \to \{\overleftarrow{bounce}, \text{border}\}$$
$$\{\overrightarrow{bounce}, \text{border}\} \to \{\overleftarrow{bounce}, \text{border}\}$$
$$\{\text{border}, \overleftarrow{bounce}\} \to \{\text{border}, \overrightarrow{bounce}\}$$
$$\{\overrightarrow{bounce_{sl}}, \text{border}, \overleftarrow{bounce}\} \to \{\overleftarrow{bounce}, \text{border}, \overrightarrow{bounce}\}$$
$$\{\overrightarrow{bounce}, \text{border}, \overleftarrow{bounce}\} \to \{\overleftarrow{bounce}, \text{border}, \overrightarrow{bounce}\}$$

(a) bouncing

$$\{delay^r_l, \overleftarrow{bounce}\} \xrightarrow{2\le l \wedge 2 \le r} \{delay^{r-1}_{l-1}, \overrightarrow{bounce_{sl}}\}$$
$$\{\overrightarrow{bounce}, \overleftarrow{split^r_l}\} \xrightarrow{2\le l \wedge 2 \le r} \{delay^{r-1}_{l-1}, \overrightarrow{bounce_{sl}}\}$$
$$\{\overrightarrow{split^r_l}, \overleftarrow{bounce}\} \xrightarrow{2\le l \wedge 2 \le r} \{delay^{r-1}_{l-1}, \overrightarrow{bounce_{sl}}\}$$

(b) `delay`

$$\{delay^r_l, \overleftarrow{bounce}\} \xrightarrow{\neg(2\le l \wedge 2 \le r)} \{\overleftarrow{bounce}, \overleftarrow{split^{r+l-1}_{l+l-1}}, \text{border}, \overrightarrow{split^{r+r-1}_{l+r-1}}, \overrightarrow{bounce}\}$$
$$\{\overrightarrow{bounce}, \overleftarrow{split^r_l}\} \xrightarrow{\neg(2\le l \wedge 2 \le r)} \{\overleftarrow{bounce}, \overleftarrow{split^{r+l-1}_{l+l-1}}, \text{border}, \overrightarrow{split^{r+r-1}_{l+r-1}}, \overrightarrow{bounce}\}$$
$$\{\overrightarrow{split^r_l}, \overleftarrow{bounce}\} \xrightarrow{\neg(2\le l \wedge 2 \le r)} \{\overleftarrow{bounce}, \overleftarrow{split^{r+l-1}_{l+l-1}}, \text{border}, \overrightarrow{split^{r+r-1}_{l+r-1}}, \overrightarrow{bounce}\}$$

(c) `split`

Figure 7: Collision rules.

or `split`. The second stage is the routing in a *macro-collision* (coherent pack of collisions issued from a collision with one or more macro-signals, implement an augmented collision). The third stage is the updating of the parameters.

Please note that the updating of parameters is started at the end of the routing macro-collision and that the test for next step is done right after. This ensures that the values are updated and the routing scheme is known before the next step starts.

For the sake of readability, signals that are redundant or irrelevant to the construction are often removed from schematic pictures. Figures without annotation (signal names) are generated through a Java SM simulator and are complete and thorough.

The colour scheme of these pictures is as follows:

- border signals are black;
- tree signals are blue—tree signals are those at the exact same position as the augmented signal in the augmented signal machine solution of Sect. 4;
- tree macro-signals are filled with yellow;
- bouncing signals are green, slow bouncing signals are green (and appear almost black in the generated pictures);
- signals helping with the geometry of macro-collisions are orange;
- purple, green and light green are used for computation within a macro-signal (in preparation for collision), with purple more often associated to testing and green to changing value;
- other colours are used for more particular signals and explained in due time;
- signals foreshadowing or helping to build a `split` are dashed; and
- signals foreshadowing or helping to build a `delay` are dotted.

Although special care has been taken so that pictures remain readable in black and white.

*5.1. ASM simulation structure*

As in Sect. 4, the tree structure is built with the help of fast signals bouncing on borders. The initial configuration thus has two border signals, defining the unit of space.

The way the ASM is simulated by a SM is depicted in Fig. 8. The filled area corresponds to macro-signals and macro-collisions of the tree. This area is also used to carry out updating. The rest are regular signals used for constructing the `delay` and `split` steps as previously, but some are doubled to take the width of the macro-signals into account.

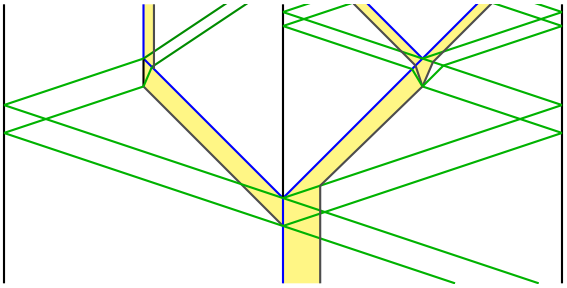


Figure 8: Ray scheme example: after a `delay`, a `split` then a `delay` on the left and a `split` on the right.

As can be seen in Fig. 8, some meta-signals of the augmented signal machine implementation have to be doubled in order to deal with the width of the macro-signals. This is the case for $\overrightarrow{bounce_{sl}}$, $\overrightarrow{bounce}$ and $\overleftarrow{bounce}$. For example, $\overrightarrow{bounce}$ is replaced by a pair $(\overrightarrow{\mathsf{bounce}^{\mathsf{top}}}, \overrightarrow{\mathsf{bounce}^{\mathsf{bot}}})$ where $\overrightarrow{\mathsf{bounce}^{\mathsf{top}}}$ should be "above" $\overrightarrow{\mathsf{bounce}^{\mathsf{bot}}}$.

These fast signals as well as borders as well as relevant collision rules are defined in Fig. 9. These definitions are quite straightforward from the ASM definition and are not exemplified. Their actions can be seen in figures 8 and 27.

| Meta-signal | Speed |
|---|---|
| border | 0 |
| $\overrightarrow{\mathsf{bounce}^{\mathsf{bot}}_{\mathsf{sl}}}$, $\overrightarrow{\mathsf{bounce}^{\mathsf{top}}_{\mathsf{sl}}}$ | $s^{\mathrm{slow}}_{\mathrm{bounce}}$ |
| $\overrightarrow{\mathsf{bounce}^{\mathsf{bot}}}$, $\overrightarrow{\mathsf{bounce}^{\mathsf{top}}}$ | $s_{\mathrm{bounce}}$ |
| $\overleftarrow{\mathsf{bounce}^{\mathsf{bot}}}$, $\overleftarrow{\mathsf{bounce}^{\mathsf{top}}}$ | $-s_{\mathrm{bounce}}$ |

$$\forall i \in \{\mathrm{bot}, \mathrm{top}\}\ \{\ \overrightarrow{\mathsf{bounce}^{i}_{\mathsf{sl}}},\ \mathsf{border}\ \} \to \{\ \overleftarrow{\mathsf{bounce}^{i}},\ \mathsf{border}\ \}$$
$$\forall i \in \{\mathrm{bot}, \mathrm{top}\}\ \{\ \overrightarrow{\mathsf{bounce}^{i}},\ \mathsf{border}\ \} \to \{\ \overleftarrow{\mathsf{bounce}^{i}},\ \mathsf{border}\ \}$$
$$\forall i \in \{\mathrm{bot}, \mathrm{top}\}\ \{\ \overleftarrow{\mathsf{bounce}^{i}},\ \mathsf{border}\ \} \to \{\ \overrightarrow{\mathsf{bounce}^{i}},\ \mathsf{border}\ \}$$

$$\forall i, j \in \{\mathrm{bot}, \mathrm{top}\}\ \{\ \overrightarrow{\mathsf{bounce}^{i}_{\mathsf{sl}}},\ \mathsf{border},\ \overleftarrow{\mathsf{bounce}^{j}}\ \} \to \{\ \overleftarrow{\mathsf{bounce}^{i}},\ \mathsf{border},\ \overrightarrow{\mathsf{bounce}^{j}}\ \}$$
$$\forall i, j \in \{\mathrm{bot}, \mathrm{top}\}\ \{\ \overrightarrow{\mathsf{bounce}^{i}},\ \mathsf{border},\ \overleftarrow{\mathsf{bounce}^{j}}\ \} \to \{\ \overleftarrow{\mathsf{bounce}^{i}},\ \mathsf{border},\ \overrightarrow{\mathsf{bounce}^{j}}\ \}$$

Figure 9: Definitions for bouncing signals.

The height of the bouncing part, $h_b$, and the width of the tree macro-signals, $w_t$, verify: $h_b = w_t$ on both output of `split` and $h_b = 4/3 w_t$ after `delay`. To keep coherence, the following updates are made through steps:

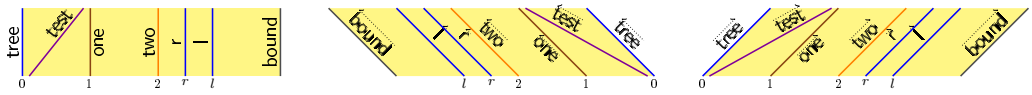
- through a `split` from a `split`, $h_{b'} = 1/2 h_b$ and $w_{t'} = 1/2 w_t$,
- through a `split` from a `delay`, $h_{b'} = 1/2 h_b$ and $w_{t'} = 3/8 w_t$,
- through a `delay` from a `split`, $h_b = 4/3 w_t$, and
- through a `delay` from a `delay`, unchanged.

These remain coherent as long as the initial values are. Although the computations are not detailed here, these relations are satisfied by all the following constructions and can be checked from the speeds of the signals involved.

*5.2. Encoding of macro-signals*

The augmented signals percolating through the tree, $\mathit{delay}^{r}_{l}$, $\overrightarrow{\mathit{split}^{r}_{l}}$ and $\overleftarrow{\mathit{split}^{r}_{l}}$, carry analogue information. Thus they have to be encoded as by macro-signals with multiple (regular) signals. The way they are encoded after an updating is depicted in Fig. 10. The sequence of parallel signals is to be understood as follows.

The tree (and variants) signals are at the exact locations of the augmented signals in the infinite tree. Then come signal one (brown) and signal two (orange) which provide the local scale. Values are encoded by the distance of the signal to tree (so 0 is at tree). Signal bound marks the other end of the macro-signal. These four signals are structural and not affected by the updating of the parameters.




(a) `delay` (b) left `split` (c) right `split`

Figure 10: Encoding of augmented signals at leaving a macro-collision.

The parameters $l$ and $r$ are encoded by l and r blue signals between one (included) and bound (excluded). Thanks to Lem. 2, all the values of the parameters are bounded by $2+\alpha$ from the start so that the scale can be set to ensure that l and r always remain between one and bound. If the value $l$ (or $r$ or both) is equal to 1 or 2, then a special signal is used that amount for the superposition. These special cases are straightforward and are not addressed anymore.

An extra signal with a different speed, test, initiates the test for the next step inside the macro-signal as presented later on. Other signals might be present inside the macro-signals to carry out the parameter updating. The scale is small enough to ensure that all involved signals remain between tree and bound during parameter updates and that any computation finishes before the next macro-collision starts. Throughout the updating, signals and positions changed, but the encoding remains similar.

Each macro-signal has the same speed as the augmented signal it encodes: 0 for $\mathit{delay}_l^r$ as in Fig. 10a, 1 for $\overrightarrow{\mathit{split}_l^r}$ as in Fig. 10c and -1 for $\overleftarrow{\mathit{split}_l^r}$ as in Fig. 10b. The sequence of signals is displayed left to right except for $\overleftarrow{\mathit{split}_l^r}$ as in Fig. 10b.

In the rest of this paper, meta-signals exclusively used to form macro-signal $\mathit{delay}_l^r$, $\overrightarrow{\mathit{split}_l^r}$ and $\overleftarrow{\mathit{split}_l^r}$ respectively carry no arrow, a right dotted arrow and a left dotted arrow, irrespective of their speeds and directions. For example one, $\dot{\text{one}}$ and $\dot{\text{one}}$ each encodes the position of scale 1, but in different macro-signals. Involved meta-signals (except test signals listed in the next section) are listed in Figure 11.

| Meta-signal | Speed |
|---|---|
| $\dot{\text{tree}}$, bound, $\dot{\text{one}}$, $\dot{\text{two}}$, $\dot{\text{l}}$, $\dot{\text{r}}$ | 1 |
| tree, bound, one, two, l, r | 0 |
| $\dot{\text{tree}}$, bound, $\dot{\text{one}}$, $\dot{\text{two}}$, $\dot{\text{l}}$, $\dot{\text{r}}$ | −1 |

Figure 11: Meta-signals for encoding the augmented meta-signal information.

### *5.3. Test*

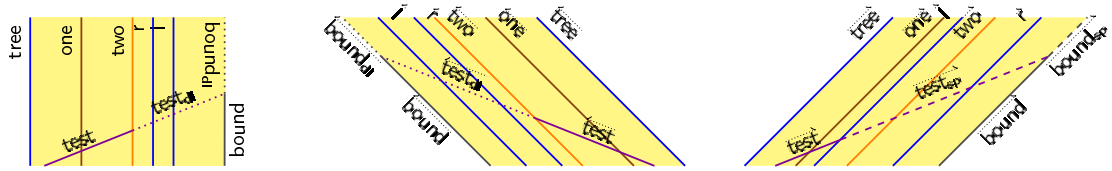
The first stage is to test whether the next step is `delay` or `split`, i.e. whether $l$ and $r$ are both greater than or equal to 2. In order to do so with the encoding, starting from tree, the test is true if two is met before any l or r. So, as shown in Fig. 12, a purple test signal starts from tree and turns to some signal recording whether the next node ought to be `delay` ($\text{test}_{\text{dl}}$, dotted) or `split` ($\text{test}_{\text{sp}}$, dashed). The information is then *stored* on the last signal bound, which becomes $\text{bound}_{\text{dl}}$ (dotted) or $\text{bound}_{\text{sp}}$ (dashed) accordingly. All signals in the macro-signals are then parallel and nothing happens until the next macro-collision.

Figures 12a and 12b provide an example where the test lead to `delay` while Fig. 12c provides one leading to `split`.

The meta-signals and collision rules at play are listed in Figure 13.

### *5.4. Rerouting*

The second stage is the macro-collision started by either $\overleftarrow{\text{bounce}^{\text{bot}}}$ or $\overrightarrow{\text{bounce}^{\text{bot}}}$ colliding with the macro-signal. It amounts for the next *tick* of the clock: the time to start generating the next node of the tree.




(a) from *delay* (b) *split* from left *split* (c) *delay* from right *split*

Figure 12: Testing for the next step.

| Meta-signal | Speed |
|---|---|
| $\overbrace{\mathsf{test}}$, $\overbrace{\mathsf{test}_{\mathsf{dl}}}$, $\overbrace{\mathsf{test}_{\mathsf{sp}}}$ | $s_{\mathrm{test}}$ |
| test, $\overbrace{\mathsf{test}_{\mathsf{dl}}}$, $\overbrace{\mathsf{test}_{\mathsf{sp}}}$ | $s_{\mathrm{test}}$ |
| $\overbrace{\mathsf{test}}$, $\overbrace{\mathsf{test}_{\mathsf{dl}}}$, $\overbrace{\mathsf{test}_{\mathsf{sp}}}$ | $-s_{\mathrm{test}}$ |

$s_{\mathrm{test}} = 2$

| Meta-signal | Speed |
|---|---|
| $\overbrace{\mathsf{bound}_{\mathsf{sp}}}$, $\overbrace{\mathsf{bound}_{\mathsf{dl}}}$ | 1 |
| $\overbrace{\mathsf{bound}_{\mathsf{sp}}}$, $\overbrace{\mathsf{bound}_{\mathsf{dl}}}$ | 0 |
| $\overbrace{\mathsf{bound}_{\mathsf{sp}}}$, $\overbrace{\mathsf{bound}_{\mathsf{dl}}}$ | $-1$ |

$\{\mathsf{test}, \mathsf{two}\} \to \{\mathsf{two}, \mathsf{test}_{\mathsf{dl}}\}$
$\{\mathsf{test}_{\mathsf{dl}}, \mathsf{bound}\} \to \{\mathsf{bound}_{\mathsf{dl}}\}$

$\{\mathsf{test}, \mathsf{l}\} \to \{\mathsf{two}, \mathsf{test}_{\mathsf{sp}}\}$
$\{\mathsf{test}, \mathsf{r}\} \to \{\mathsf{two}, \mathsf{test}_{\mathsf{sp}}\}$
$\{\mathsf{test}_{\mathsf{sp}}, \mathsf{bound}\} \to \{\mathsf{bound}_{\mathsf{sp}}\}$

Collision rules with arrows are the same as without arrows.

Figure 13: Definitions for the test.

Duration of steps are handled by bouncing signals so that only the macro-collision is addressed. The cases of *delay* and *split* are considered one after the other. In each case, the previous node has to be considered.

Rerouting is done using common signal machine tools. In particular, a *reflection* is when each signal of a macro-signal bounce off a common signal at a common speed. A *refraction* is when each signal of a macro-signal takes a new common speed upon crossing a common signal. In both cases, parallelism ensures that the proportion between distances between signals remains unchanged.

Since signals one, two, l and r are only rerouted and each in the same fashion, in the schematic pictures, for the sake of clarity, only signal one is shown.

### *5.4.1. delay rerouting*

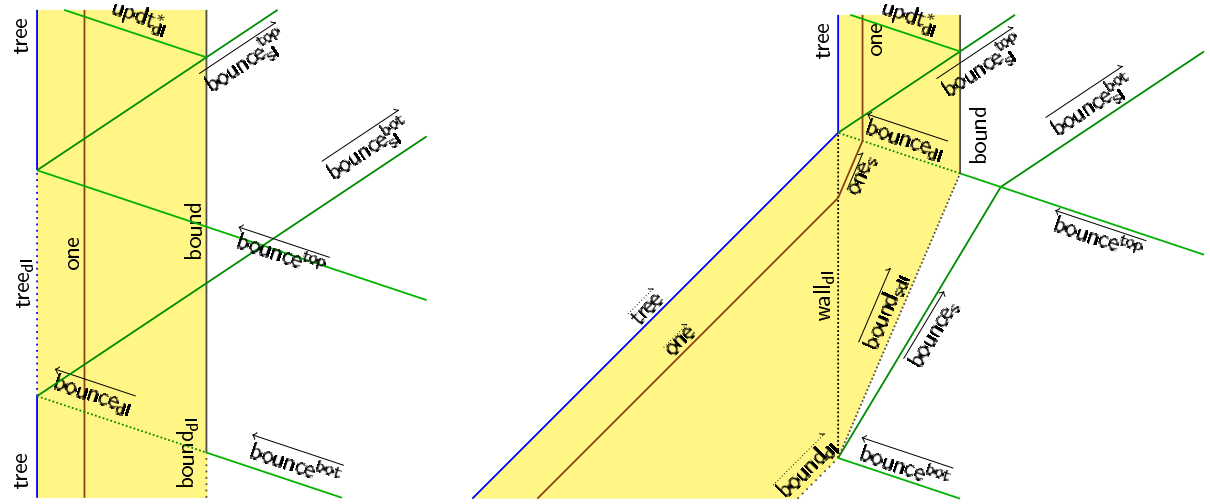
At a *delay* step, the macro-signal is only straightened if it is not already vertical, and bouncing signals are sent.

*delay rerouting from a delay.* Since the macro-signal is already vertical, bouncing signals have to be sent back and the updating initiated. This is done as in Fig. 14a: upon crossing $\mathsf{bound}_{\mathsf{dl}}$, the bottom bouncing signal $\overleftarrow{\mathsf{bounce}^{\mathsf{bot}}}$ takes note that the next intersection is a *delay*, becoming $\overleftarrow{\mathsf{bounce}_{\mathsf{dl}}}$. It then simply bounces off tree, becoming the slower $\overrightarrow{\mathsf{bounce}^{\mathsf{bot}}_{\mathsf{sl}}}$. While bouncing, it also turns tree into $\mathsf{tree}_{\mathsf{dl}}$, indicating the top bouncing signal to just bounce back. When leaving, the signal $\overrightarrow{\mathsf{bounce}^{\mathsf{top}}}$ sprouts a signal $\mathsf{updt}^{*}_{\mathsf{dl}}$ which will start the updating (parameter update then test) within the macro-signals on meeting tree.

Signals like one are unaffected and just pass through the macro-collision.

*delay rerouting from a right branch of a split.* If the macro-signal comes from the left, it goes first through a *refraction* on $\mathsf{wall}_{\mathsf{dl}}$ and then a *refraction* on $\overleftarrow{\mathsf{bounce}_{\mathsf{dl}}}$, as in Fig. 14b to ensure the correct scaling of the macro-signal (the two *refraction* are more visible on Fig. 23b). The intermediate speed is such that the width of the macro-signal is halved.

Again, information about the next step is carried on the bound side of the macro-signal, by $\overbrace{\mathsf{bound}_{\mathsf{dl}}}$. The output of the starting collision (between $\overleftarrow{\mathsf{bounce}^{\mathsf{bot}}}$ and $\overbrace{\mathsf{tree}}$) contains three signals: A vertical signal $\mathsf{tree}_{\mathsf{dl}}$ that refracts signal in the band once. A signal $\overrightarrow{\mathsf{bound}_{\mathsf{sdl}}}$ that is the diffraction of the bound signal, and still carries the *delay* information; that way, $\overleftarrow{\mathsf{bounce}^{\mathsf{top}}}$ can become $\overleftarrow{\mathsf{bounce}_{\mathsf{dl}}}$ and handles the second *refraction*. A signal $\overrightarrow{\mathsf{bounce}_{\mathsf{s}}}$ that, upon intersecting with $\overleftarrow{\mathsf{bounce}^{\mathsf{top}}}$, can start $\overrightarrow{\mathsf{bounce}^{\mathsf{bot}}_{\mathsf{sl}}}$ at the correct place.




(a) after a `delay`.

(b) after a right `split`

Figure 14: `delay` after a `delay` or a right `split`.

`delay` *rerouting from a left branch of a* `split`. The macro-signal comes from the right, it first undergoes a `reflection` on $\mathsf{wall}_{\mathsf{dl}}$, then a `refraction` on $\overleftarrow{\mathsf{bounce}_{\mathsf{dl}}}$, as shown in Fig. 15 (it is more visible on Fig. 23c). Once again, the intermediate speed is such that the width of the macro-signal is halved. So after the first `reflection` everything behaves the same way than after the first `refraction` when coming from the left.

The reason for using a `reflection` is the following: information on the next step is carried on the bound side of the macro-signal ($\overleftarrow{\mathsf{bound}_{\mathsf{dl}}}$) and has to be below (before) $\overleftarrow{\mathsf{tree}}$; this is why a leftward macro-signal is a mirror image of a rightward macro-signal, rather than a mere slanting of a vertical macro-signal—in other words the bound is on the left.

Without a $\overleftarrow{\mathsf{bounce}^{\mathsf{top}}}$ to handle the `refraction`, we need another signal, $\overrightarrow{\mathsf{dsep}_2}$ (orange) coming from somewhere else. A $\overleftarrow{\mathsf{dsep}_1}$ (orange as well) signal is thus launched at the bottom intersection with a default fast speed of $-3$. It intersects the $\overrightarrow{\mathsf{bounce}^{\mathsf{top}}}$ signal and sprout $\overrightarrow{\mathsf{dsep}_2}$. The $\overrightarrow{\mathsf{dsep}_2}$ signal in turn has a finely tuned speed so as to intersect with $\overrightarrow{\mathsf{bounce}_{\mathsf{s}}}$ at the right time and place. There, a $\overleftarrow{\mathsf{bounce}^{\mathsf{top}}}$ is sprouted as if it came from the right and handles the `refraction`.

The new meta-signals and collision rules involved are listed in Figure 16.

### *5.4.2. Split Rerouting*

At a `split` step we need to fork the macro-signal, and go half a unit of space both ways during half a unit of time (again using bouncing signals). Forking is done by sending a copy of each side (with corresponding orders of signals inside the new macro-signals) and adding a motionless $\mathsf{border}$ signal, so as to set border and scale for the new branches.

`split` *rerouting from a* `split`. Routing from each branch is done symmetrically (as can be seen in Figs. 26b and 26c) so only one case is presented. Forking the macro-signal from the right branch of a `split` is done by raising a vertical signal $\mathsf{wall}_{\mathsf{sp}}$ on which the macro-signal gets duplicated as can be seen in Fig. 17. The duplication can be viewed as a simultaneously operating a `reflection` and a `refraction` on the macro-signal. Both branches are then routed and their widths are halved in the process.

The meta-signals and collision rules at play are listed in Fig. 18.

`split` *rerouting from a* `delay`. If the previous step is a `delay`, the construction, shown in Fig. 19, is more involving. Again this is due to the difference of orientation between the signals in a leftward band and a straight one. The right part of the collision output is obtained by a simple `refraction` on signal $\overleftarrow{\mathsf{bounce}^{1}_{\mathsf{sp}}}$.

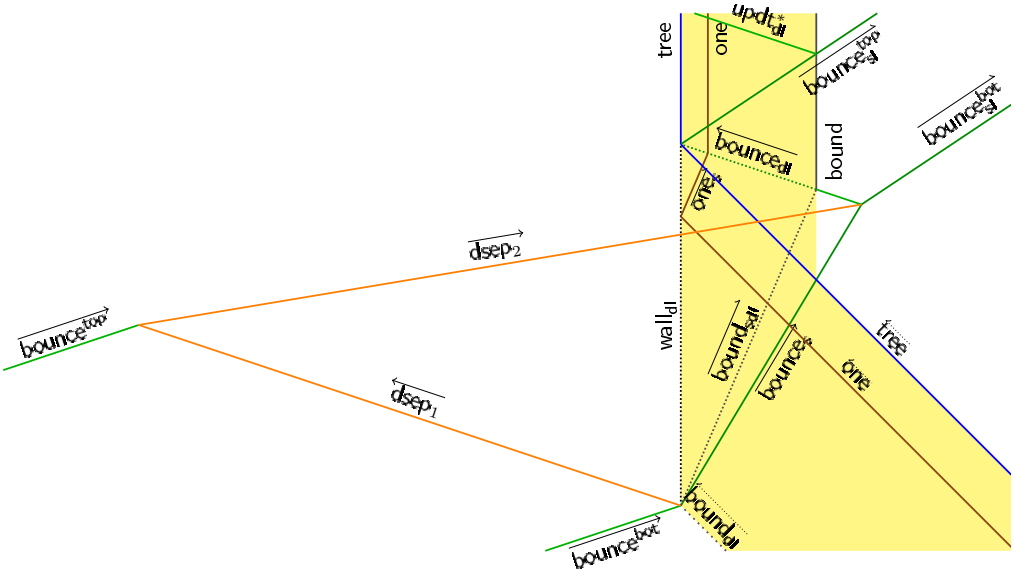



Figure 15: `delay` after a left `split`.

$s_{\text{shrink}} = 3/7$
$s^{\text{fast}}_{\text{shrink}} = 3/5$

| Meta-signal | Speed | Meta-signal | Speed |
|---|---|---|---|
| $\text{tree}_{\text{dl}}$, $\text{wall}_{\text{dl}}$ | $0$ | $\overleftarrow{\text{bounce}_{\text{dl}}}$, $\text{updt}^*_{\text{dl}}$ | $-s_{\text{bounce}}$ |
| $\overleftarrow{\text{dsep}_1}$ | $-s_{\text{bounce}}$ | $\overrightarrow{\text{bounce}_{\text{s}}}$ | $s^{\text{fast}}_{\text{shrink}}$ |
| $\overrightarrow{\text{dsep}_2}$ | $2s_{\text{bounce}}$ | $\overrightarrow{\text{bound}_{\text{s dl}}}$, $\overrightarrow{\text{one}_{\text{s}}}$, $\overrightarrow{\text{two}_{\text{s}}}$, $\overrightarrow{\text{r}_{\text{s}}}$, $\overrightarrow{\text{l}_{\text{s}}}$ | $s_{\text{shrink}}$ |

$$
\begin{aligned}
\{\, \text{bound}_{\text{dl}},\ \overleftarrow{\text{bounce}^{\text{bot}}} \,\} &\to \{\, \overleftarrow{\text{bounce}_{\text{dl}}},\ \text{bound} \,\} \\
\{\, \overleftarrow{\text{bounce}_{\text{dl}}},\ \text{tree} \,\} &\to \{\, \text{tree}_{\text{dl}},\ \overrightarrow{\text{bounce}^{\text{bot}}_{\text{sl}}} \,\} \\
\{\, \text{tree}_{\text{dl}},\ \overleftarrow{\text{bounce}^{\text{top}}} \,\} &\to \{\, \text{tree},\ \overleftarrow{\text{bounce}^{\text{top}}_{\text{sl}}} \,\} \\
\{\, \overleftarrow{\text{bound}_{\text{dl}}},\ \overleftarrow{\text{bounce}^{\text{bot}}} \,\} &\to \{\, \text{wall}_{\text{dl}},\ \overrightarrow{\text{bound}_{\text{s dl}}},\ \overrightarrow{\text{bounce}_{\text{s}}} \,\} \\
\{\, \overrightarrow{\text{bounce}_{\text{s}}},\ \overleftarrow{\text{bounce}^{\text{top}}} \,\} &\to \{\, \overleftarrow{\text{bounce}^{\text{top}}},\ \overrightarrow{\text{bounce}^{\text{bot}}_{\text{sl}}} \,\} \\
\{\, \overleftarrow{\text{tree}},\ \text{wall}_{\text{dl}},\ \overleftarrow{\text{bounce}_{\text{dl}}} \,\} &\to \{\, \text{tree},\ \overleftarrow{\text{bounce}^{\text{top}}_{\text{sl}}} \,\} \\
\{\, \overrightarrow{\text{bounce}^{\text{bot}}},\ \overleftarrow{\text{bound}_{\text{dl}}} \,\} &\to \{\, \overleftarrow{\text{dsep}_1},\ \text{wall}_{\text{dl}},\ \overrightarrow{\text{bound}_{\text{s dl}}},\ \overrightarrow{\text{bounce}_{\text{s}}} \,\} \\
\{\, \overrightarrow{\text{bounce}^{\text{top}}},\ \overleftarrow{\text{dsep}_1} \,\} &\to \{\, \overrightarrow{\text{dsep}_2} \,\} \\
\{\, \overrightarrow{\text{bounce}_{\text{s}}},\ \overrightarrow{\text{dsep}_2} \,\} &\to \{\, \overleftarrow{\text{bounce}^{\text{top}}},\ \overrightarrow{\text{bounce}^{\text{bot}}_{\text{sl}}} \,\} \\
\{\, \text{wall}_{\text{dl}},\ \overleftarrow{\text{tree}},\ \overleftarrow{\text{bounce}_{\text{dl}}} \,\} &\to \{\, \text{tree},\ \overrightarrow{\text{bounce}^{\text{top}}_{\text{sl}}} \,\} \\
\{\, \overrightarrow{\text{bounce}^{\text{top}}_{\text{sl}}},\ \text{bound} \,\} &\to \{\, \text{updt}^*_{\text{dl}},\ \text{bound},\ \overrightarrow{\text{bounce}^{\text{top}}_{\text{sl}}} \,\} \\
\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\, \overrightarrow{\mu},\ \text{wall}_{\text{dl}} \,\} &\to \{\, \text{wall}_{\text{dl}},\ \overrightarrow{\mu_{\text{s}}} \,\} \\
\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\, \overleftarrow{\mu},\ \text{wall}_{\text{dl}} \,\} &\to \{\, \text{wall}_{\text{dl}},\ \overleftarrow{\mu_{\text{s}}} \,\} \\
\{\, \overrightarrow{\text{bound}_{\text{s dl}}},\ \overleftarrow{\text{bounce}^{\text{top}}} \,\} &\to \{\, \overleftarrow{\text{bounce}_{\text{dl}}},\ \text{bound} \,\} \\
\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\, \overrightarrow{\mu_{\text{s}}},\ \overleftarrow{\text{bounce}_{\text{dl}}} \,\} &\to \{\, \overleftarrow{\text{bounce}_{\text{dl}}},\ \mu \,\}
\end{aligned}
$$

Figure 16: Definitions for rerouting at a `delay` node.

The left part is obtained through a `refraction` by $\overrightarrow{\text{sep}_1}$, a `reflection` on $\overleftarrow{\text{bound}_1}$ and a `refraction` on $\overrightarrow{\text{sep}_3}$.

The new meta-signals and collision rules at play are listed in Fig. 20.

### *5.5. Updating the parameters*

The updating of parameters is done according to Algo. 1. It occurs right after a macro-collision and depends on its nature.

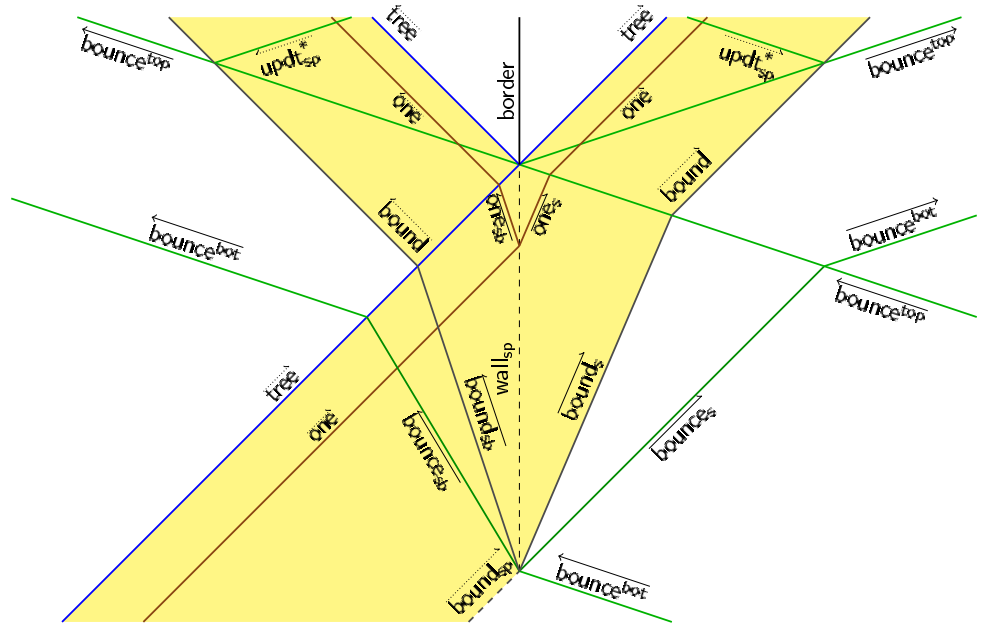



Figure 17: `split` after a right `split` (routing).

$s^{\text{back}}_{\text{shrink}} = 1/3$
$s^{\text{bounce}}_{\text{shrink}} = 1$
$s^{\text{bounceBack}}_{\text{shrink}} = 3/5$

| Meta-signal | Speed |
|---|---|
| $\text{wall}_{\text{sp}}$ | $0$ |
| $\overrightarrow{\text{bounce}_{\text{s}}}$ | $s^{\text{bounce}}_{\text{shrink}}$ |
| $\overleftarrow{\text{bounce}_{\text{s}}}$ | $-s^{\text{bounce}}_{\text{shrink}}$ |
| $\overrightarrow{\text{bounce}_{\text{sb}}}$ | $s^{\text{bounceBack}}_{\text{shrink}}$ |
| $\overleftarrow{\text{bounce}_{\text{sb}}}$ | $-s^{\text{bounceBack}}_{\text{shrink}}$ |

| Meta-signal | Speed |
|---|---|
| $\overrightarrow{\text{bound}_{\text{s}}}$ | $s_{\text{shrink}}$ |
| $\overleftarrow{\text{one}_{\text{s}}}, \overleftarrow{\text{two}_{\text{s}}}, \overleftarrow{\text{r}_{\text{s}}}, \overleftarrow{\text{l}_{\text{s}}}, \overleftarrow{\text{bound}_{\text{s}}}$ | $-s_{\text{shrink}}$ |
| $\overrightarrow{\text{one}_{\text{sb}}}, \overrightarrow{\text{two}_{\text{sb}}}, \overrightarrow{\text{r}_{\text{sb}}}, \overrightarrow{\text{l}_{\text{sb}}}, \overrightarrow{\text{bound}_{\text{sb}}}$ | $s^{\text{back}}_{\text{shrink}}$ |
| $\overleftarrow{\text{one}_{\text{sb}}}, \overleftarrow{\text{two}_{\text{sb}}}, \overleftarrow{\text{r}_{\text{sb}}}, \overleftarrow{\text{l}_{\text{sb}}}, \overleftarrow{\text{bound}_{\text{sb}}}$ | $-s^{\text{back}}_{\text{shrink}}$ |
| $\text{updt}^{*}_{\text{sp}}$ | $-s_{\text{test}}$ |
| $\text{updt}^{*}_{\text{sp}}$ | $s_{\text{test}}$ |

$$
\begin{aligned}
\{\, \overleftarrow{\text{bounce}^{\text{bot}}},\ \overrightarrow{\text{bound}_{\text{sp}}} \,\} &\to \{\, \overrightarrow{\text{bounce}_{\text{sb}}},\ \overrightarrow{\text{bound}_{\text{sb}}},\ \text{wall}_{\text{sp}},\ \overrightarrow{\text{bound}_{\text{s}}},\ \overrightarrow{\text{bounce}_{\text{s}}} \,\} \\
\{\, \overrightarrow{\text{bounce}^{\text{bot}}},\ \overleftarrow{\text{bound}_{\text{sp}}} \,\} &\to \{\, \overrightarrow{\text{bounce}_{\text{sb}}},\ \overrightarrow{\text{bound}_{\text{sb}}},\ \text{wall}_{\text{sp}},\ \overleftarrow{\text{bound}_{\text{s}}},\ \overleftarrow{\text{bounce}_{\text{s}}} \,\} \\
\{\, \overleftarrow{\text{tree}},\ \text{wall}_{\text{sp}},\ \overleftarrow{\text{bounce}^{\text{top}}} \,\} &\to \{\, \overleftarrow{\text{bounce}^{\text{top}}},\ \text{updt}^{\text{lo}}_{\text{sp}},\ \text{updt}^{\text{hi}}_{\text{sp}},\ \overleftarrow{\text{tree}},\ \text{border},\ \overrightarrow{\text{tree}},\ \overrightarrow{\text{bounce}^{\text{top}}} \,\} \\
\{\, \overrightarrow{\text{bound}_{\text{s}}},\ \overleftarrow{\text{bounce}^{\text{top}}} \,\} &\to \{\, \overleftarrow{\text{bounce}^{\text{top}}},\ \overleftarrow{\text{bound}} \,\} \\
\{\, \overleftarrow{\text{tree}},\ \overleftarrow{\text{bound}_{\text{sb}}} \,\} &\to \{\, \overleftarrow{\text{bound}},\ \overleftarrow{\text{tree}} \,\} \\
\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\, \text{wall}_{\text{sp}},\ \overrightarrow{\mu} \,\} &\to \{\, \overrightarrow{\mu_{\text{sb}}},\ \text{wall}_{\text{sp}},\ \overrightarrow{\mu_{\text{s}}} \,\} \\
\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\, \overrightarrow{\mu_{\text{s}}},\ \overleftarrow{\text{bounce}^{\text{top}}} \,\} &\to \{\, \overleftarrow{\text{bounce}^{\text{top}}},\ \overrightarrow{\mu} \,\} \\
\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\, \overleftarrow{\text{tree}},\ \overleftarrow{\mu_{\text{sb}}} \,\} &\to \{\, \overleftarrow{\mu},\ \overleftarrow{\text{tree}} \,\} \\
\{\, \overleftarrow{\text{tree}},\ \overleftarrow{\text{bounce}_{\text{sb}}} \,\} &\to \{\, \overleftarrow{\text{bounce}^{\text{bot}}},\ \overleftarrow{\text{tree}} \,\} \\
\{\, \overrightarrow{\text{bounce}^{\text{top}}},\ \overleftarrow{\text{bound}} \,\} &\to \{\, \text{updt}^{*}_{\text{sp}},\ \overrightarrow{\text{bounce}^{\text{top}}},\ \overleftarrow{\text{bound}} \,\} \\
\\
\{\, \overleftarrow{\text{bound}_{\text{s}}},\ \overrightarrow{\text{bounce}^{\text{top}}} \,\} &\to \{\, \overrightarrow{\text{bounce}^{\text{top}}},\ \overleftarrow{\text{bound}} \,\} \\
\{\, \overrightarrow{\text{tree}},\ \overrightarrow{\text{bound}_{\text{sb}}} \,\} &\to \{\, \overrightarrow{\text{bound}},\ \overrightarrow{\text{tree}} \,\} \\
\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\, \text{wall}_{\text{sp}},\ \overleftarrow{\mu} \,\} &\to \{\, \overleftarrow{\mu_{\text{s}}},\ \text{wall}_{\text{sp}},\ \overrightarrow{\mu_{\text{sb}}} \,\} \\
\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\, \overleftarrow{\mu_{\text{s}}},\ \overrightarrow{\text{bounce}^{\text{top}}} \,\} &\to \{\, \overrightarrow{\text{bounce}^{\text{top}}},\ \overleftarrow{\mu} \,\} \\
\forall \mu \in \{\text{one}, \text{two}, \text{r}, \text{l}\} \quad \{\, \overrightarrow{\text{tree}},\ \overrightarrow{\mu_{\text{sb}}} \,\} &\to \{\, \overrightarrow{\mu},\ \overrightarrow{\text{tree}} \,\} \\
\{\, \overrightarrow{\text{tree}},\ \overrightarrow{\text{bounce}_{\text{sb}}} \,\} &\to \{\, \overrightarrow{\text{bounce}^{\text{bot}}},\ \overrightarrow{\text{tree}} \,\} \\
\{\, \overleftarrow{\text{bounce}^{\text{top}}},\ \overleftarrow{\text{bound}} \,\} &\to \{\, \text{updt}^{*}_{\text{sp}},\ \overleftarrow{\text{bounce}^{\text{top}}},\ \overleftarrow{\text{bound}} \,\}
\end{aligned}
$$

Figure 18: Definitions for routing at a `split` from right or left.

### *5.5.1.* `delay` *parameter update*

The updating is done by removing 1 from both $l$ and $r$ as depicted in Fig. 21. This corresponds to a shift of l and r by the distance from tree to one. For l, this is done by constructing a parallelogram with one side going from one to tree and the opposite side from l to its updated position. Parallel lines simply correspond to signals with the same speed ($\text{one}_{\text{back}}$ and $\text{r}_{\text{back}}$, and $\text{updt}^2_{\text{dl}}$ and $\text{minus}^2$). Signal r is shifted similarly. After

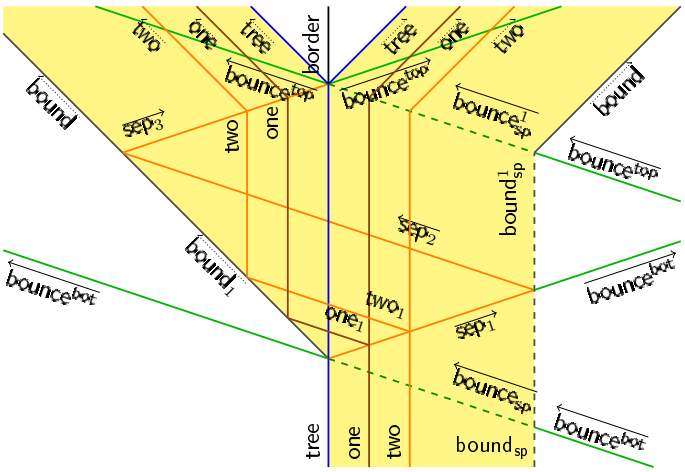



Figure 19: Routing for *split* after a *delay*.

| Meta-signal | Speed |
|---|---|
| $\overleftarrow{\mathsf{bound}_1}$ | $-1$ |
| $\overrightarrow{\mathsf{sep}_1}$, $\overrightarrow{\mathsf{sep}_3}$ | $s_{\mathrm{bounce}}$ |
| $\overleftarrow{\mathsf{sep}_2}$ | $-s_{\mathrm{bounce}}$ |
| $\overleftarrow{\mathsf{bounce}_{\mathsf{sp}}}$, $\overleftarrow{\mathsf{bounce}^1_{\mathsf{sp}}}$ | $-s_{\mathrm{test}}$ |

$$\{\, \mathsf{bound}_{\mathsf{sp}},\ \overleftarrow{\mathsf{bounce}^{\mathsf{bot}}} \,\} \to \{\, \overleftarrow{\mathsf{bounce}_{\mathsf{sp}}},\ \mathsf{bound}^1_{\mathsf{sp}} \,\}$$
$$\{\, \mathsf{bound}^1_{\mathsf{sp}},\ \overleftarrow{\mathsf{bounce}^{\mathsf{top}}} \,\} \to \{\, \overleftarrow{\mathsf{bounce}^1_{\mathsf{sp}}},\ \overleftarrow{\mathsf{bound}} \,\}$$
$$\{\, \mathsf{tree}, \overleftarrow{\mathsf{bounce}_{\mathsf{sp}}} \,\} \to \{\, \overleftarrow{\mathsf{bounce}^{\mathsf{bot}}}, \overleftarrow{\mathsf{bound}_1}, \mathsf{tree}, \overrightarrow{\mathsf{sep}_1} \,\}$$
$$\forall \mu \in \{\mathsf{one}, \mathsf{two}, \mathsf{r}, \mathsf{l}\} \quad \{\, \overrightarrow{\mathsf{sep}_1},\ \mu \,\} \to \{\, \mu_1,\ \mu,\ \overrightarrow{\mathsf{sep}_1} \,\}$$
$$\{\, \overrightarrow{\mathsf{sep}_1},\ \mathsf{bound}^1_{\mathsf{sp}} \,\} \to \{\, \overleftarrow{\mathsf{sep}_2},\ \mathsf{bound}^1_{\mathsf{sp}},\ \overrightarrow{\mathsf{bounce}^{\mathsf{bot}}} \,\}$$
$$\forall \mu \in \{\mathsf{one}, \mathsf{two}, \mathsf{r}, \mathsf{l}\} \quad \{\, \overleftarrow{\mathsf{bound}_1},\ \mu_1 \,\} \to \{\, \overleftarrow{\mathsf{bound}_1},\ \mu \,\}$$
$$\{\, \overleftarrow{\mathsf{bound}_1},\ \overleftarrow{\mathsf{sep}_2} \,\} \to \{\, \overleftarrow{\mathsf{bound}},\ \overrightarrow{\mathsf{sep}_3} \,\}$$
$$\forall \mu \in \{\mathsf{one}, \mathsf{two}, \mathsf{r}, \mathsf{l}\} \quad \{\, \mu,\ \overleftarrow{\mathsf{bounce}^1_{\mathsf{sp}}} \,\} \to \{\, \overleftarrow{\mathsf{bounce}^1_{\mathsf{sp}}},\ \overrightarrow{\mu} \,\}$$
$$\forall \mu \in \{\mathsf{one}, \mathsf{two}, \mathsf{r}, \mathsf{l}\} \quad \{\, \overrightarrow{\mathsf{sep}_3},\ \mu \,\} \to \{\, \overrightarrow{\mathsf{sep}_3},\ \overleftarrow{\mu} \,\}$$
$$\{\, \overrightarrow{\mathsf{sep}_3},\ \mathsf{tree},\ \overleftarrow{\mathsf{bounce}^1_{\mathsf{sp}}} \,\} \to \{\, \overleftarrow{\mathsf{bounce}^{\mathsf{top}}},\ \overleftarrow{\mathsf{tree}},\ \mathsf{border},\ \overrightarrow{\mathsf{tree}},\ \overrightarrow{\mathsf{bounce}^{\mathsf{top}}} \,\}$$

Figure 20: Definitions for *split* routing after a *delay*.

that, the signal $\mathsf{test}_{\mathsf{start}}$ is generated and sent to collide tree to generate test to start the test sequence as seen previously.

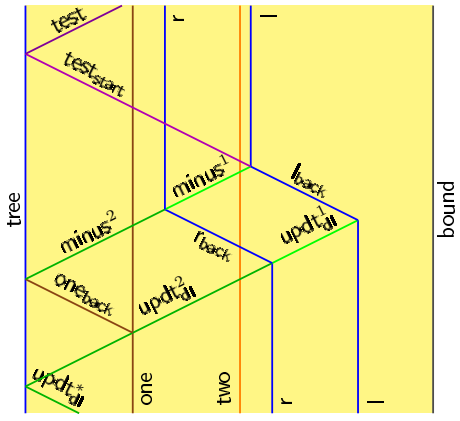



Figure 21: Updating $l$ and $r$ after a *delay*.

The new meta-signals and collision rules used in 21 are listed in Fig. 22. The signals $\mathsf{updt}^2_{\mathsf{dl}}$ and $\mathsf{updt}^1_{\mathsf{dl}}$ (as well as $\mathsf{minus}^2$ and $\mathsf{minus}^1$) are used to count down before disappearing after meeting both l and r (in whatever order).

| Meta-signal | Speed |
|---|---|
| $\mathsf{updt}^2_{\mathsf{dl}}$, $\mathsf{updt}^1_{\mathsf{dl}}$, $\mathsf{minus}^2$, $\mathsf{minus}^1$, $\grave{\mathsf{updt}}^2_{\mathsf{dl}}$, $\grave{\mathsf{updt}}^1_{\mathsf{dl}}$, $\grave{\mathsf{minus}}^2$, $\grave{\mathsf{minus}}^1$ | $s_{\text{test}}$ |
| $\mathsf{one}_{\mathsf{back}}$, $\mathsf{l}_{\mathsf{back}}$, $\mathsf{r}_{\mathsf{back}}$, $\mathsf{test}_{\mathsf{start}}$, $\grave{\mathsf{one}}_{\mathsf{back}}$, $\grave{\mathsf{l}}_{\mathsf{back}}$, $\grave{\mathsf{r}}_{\mathsf{back}}$, $\grave{\mathsf{test}}_{\mathsf{start}}$, $\grave{\mathsf{updt}}^*_{\mathsf{dl}}$ | $-s_{\text{test}}$ |
| $\mathsf{updt}^2_{\mathsf{dl}}$, $\mathsf{updt}^1_{\mathsf{dl}}$, $\mathsf{minus}^2$, $\mathsf{minus}^1$ | $-s_{\text{test}}$ |
| $\hat{\mathsf{one}}_{\mathsf{back}}$, $\hat{\mathsf{l}}_{\mathsf{back}}$, $\hat{\mathsf{r}}_{\mathsf{back}}$, $\hat{\mathsf{test}}_{\mathsf{start}}$, $\hat{\mathsf{updt}}^*_{\mathsf{dl}}$ | $s_{\text{test}}$ |

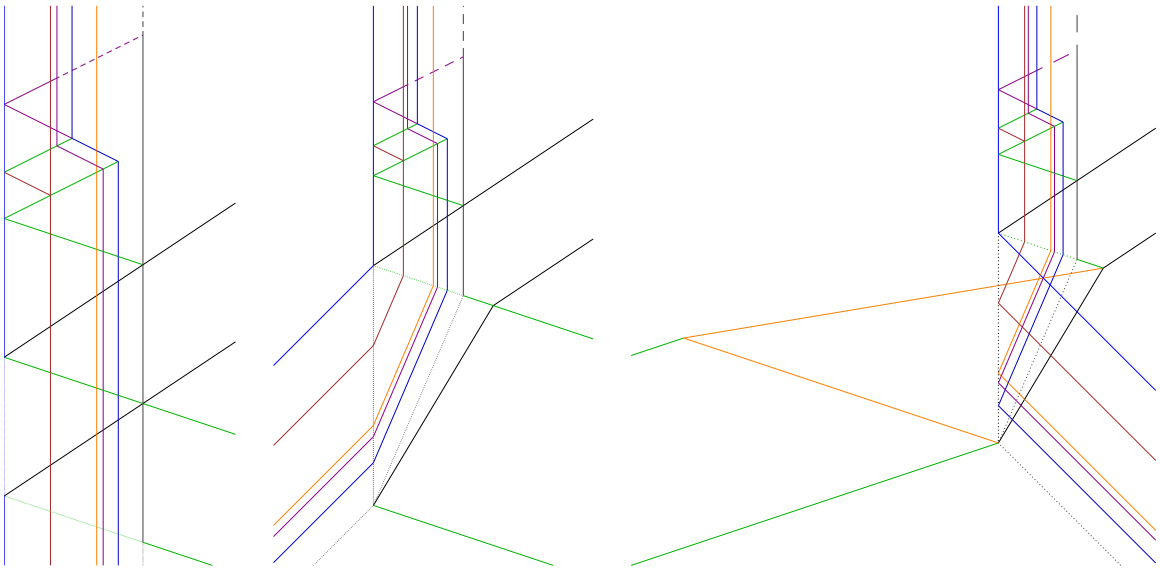
$\{\mathsf{tree}, \mathsf{updt}^*_{\mathsf{dl}}\} \to \{\mathsf{tree}, \mathsf{updt}^2_{\mathsf{dl}}\}$
$\{\mathsf{updt}^2_{\mathsf{dl}}, \mathsf{one}\} \to \{\mathsf{one}_{\mathsf{back}}, \mathsf{one}, \mathsf{updt}^2_{\mathsf{dl}}\}$
$\{\mathsf{tree}, \mathsf{one}_{\mathsf{back}}\} \to \{\mathsf{tree}, \mathsf{minus}^2\}$
$\{\mathsf{tree}, \mathsf{test}_{\mathsf{start}}\} \to \{\mathsf{tree}, \mathsf{test}\}$

$\forall \mu \in \{\mathsf{r}, \mathsf{l}\}$ $\{\mathsf{updt}^2_{\mathsf{dl}}, \mu\} \to \{\mu_{\text{ back}}, \mathsf{updt}^1_{\mathsf{dl}}\}$
$\forall \mu \in \{\mathsf{r}, \mathsf{l}\}$ $\{\mathsf{updt}^1_{\mathsf{dl}}, \mu\} \to \{\mu_{\text{ back}}\}$
$\forall \mu \in \{\mathsf{r}, \mathsf{l}\}\{\mathsf{minus}^2, \mu_{\text{ back}}\} \to \{\mu, \mathsf{minus}^1\}$
$\forall \mu \in \{\mathsf{r}, \mathsf{l}\}\{\mathsf{minus}^1, \mu_{\text{ back}}\} \to \{\mu, \mathsf{test}_{\mathsf{start}}\}$

Collision rules with arrows are the same as without arrows.

Figure 22: Definitions for updating parameters after a ***delay***.

A full ***delay*** step, that is rerouting and parameter update, is performed as shown in Fig. 23.



(a) ***delay*** from ***delay*** (b) ***delay*** from right ***split*** (c) ***delay*** from left ***split***

Figure 23: ***delay*** implementations.

*5.5.2.* ***split*** *parameter update*

The updating on the right (resp. left) branch of a split is illustrated by Fig. 24 and done as follows: $r$ (resp. $l$) is added to both $l$ and $r$; then 1 is removed from both $l$ and $r$. What happens on the right and on the left branch is symmetrical with the roles of $l$ and $r$ swapped; so that only what happens on a right branch is presented.

Adding $r$ to both $l$ and $r$ is done by moving $\mathsf{l}$ and $\mathsf{r}$ by the distance from $\mathsf{tree}$ to $\mathsf{r}$ (again using a parallelogram). The computing signals bounce back to $\grave{\mathsf{tree}}$ after measuring that distance and before adding it so that the construction does not depend on the order of $\mathsf{l}$ and $\mathsf{r}$. The new meta-signals and collision rules at play are listed in Fig. 25.

The computation is finished by removing 1 and starting the test stage. This is done exactly as in the parameter update occurring after a ***delay*** step. The (dotted arrow) meta-signals and collision rules are already defined in Fig. 22.

A full ***split*** step, that is routing and parameter update is shown in Fig. 26, with Fig. 26c zooming on the routing part.

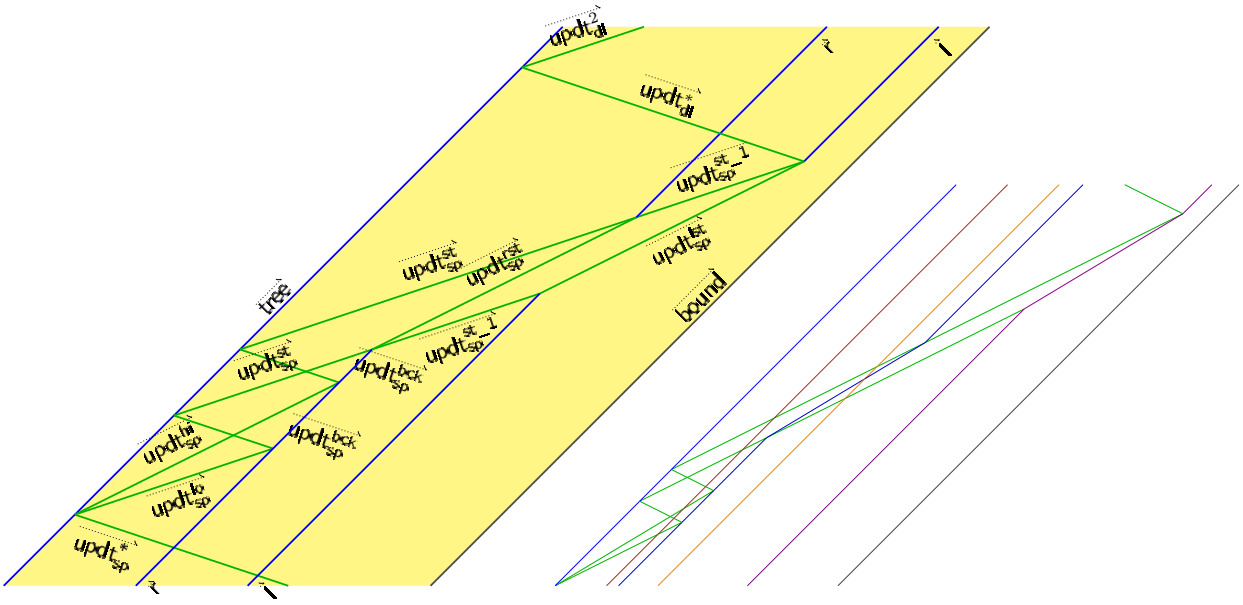



Figure 24: Updating $l$ and $r$ after a ***split***.

$s^{\text{up}}_{\text{split}} = 5/3$

| Meta-signal | Speed | | |
|---|---|---|---|
| $\overleftarrow{\text{updt}^{\text{lo}}_{\text{sp}}}$, $\overleftarrow{\text{updt}^{\text{st}}_{\text{sp}}}$, $\overleftarrow{\text{updt}^{\text{st}\_\text{i}}_{\text{sp}}}$, $\overleftarrow{\text{updt}^{\text{bck}}_{\text{sp}}}$, $\overleftarrow{\text{updt}^{*}_{\text{dl}}}$ | $s_{\text{test}}$ | $\forall i \in \{\text{lo}, \text{hi}\}$ | $\{\overrightarrow{\text{updt}^{i}_{\text{sp}}}, \overleftarrow{\text{i}}\} \to \{\overleftarrow{\text{updt}^{\text{bck}}_{\text{sp}}}, \overleftarrow{\text{i}}\}$ |
| $\overrightarrow{\text{updt}^{\text{lo}}_{\text{sp}}}$, $\overrightarrow{\text{updt}^{\text{st}}_{\text{sp}}}$, $\overrightarrow{\text{updt}^{\text{st}\_\text{i}}_{\text{sp}}}$, $\overrightarrow{\text{updt}^{\text{bck}}_{\text{sp}}}$, $\overrightarrow{\text{updt}^{*}_{\text{dl}}}$ | $-s_{\text{test}}$ | $\forall i \in \{\text{lo}, \text{hi}\}$ | $\{\overleftarrow{\text{updt}^{i}_{\text{sp}}}, \overrightarrow{\text{i}}\} \to \{\overrightarrow{\text{updt}^{\text{bck}}_{\text{sp}}}, \overrightarrow{\text{i}}\}$ |
| $\overrightarrow{\text{updt}^{\text{hi}}_{\text{sp}}}$, $\overrightarrow{\text{updt}^{\text{lst}}_{\text{sp}}}$, $\overrightarrow{\text{updt}^{\text{rst}}_{\text{sp}}}$ | $-s^{\text{up}}_{\text{split}}$ | † $\forall i \in \{\text{r}, \text{l}\}$ | $\{\overrightarrow{\text{updt}^{\text{st}}_{\text{sp}}}, \overrightarrow{\text{i}}\} \to \{\overrightarrow{\text{updt}^{\text{st}\_\text{i}}_{\text{sp}}}, \overrightarrow{\text{updt}^{\text{ist}}_{\text{sp}}}\}$ |
| $\overleftarrow{\text{updt}^{\text{hi}}_{\text{sp}}}$, $\overleftarrow{\text{updt}^{\text{lst}}_{\text{sp}}}$, $\overleftarrow{\text{updt}^{\text{rst}}_{\text{sp}}}$ | $s^{\text{up}}_{\text{split}}$ | † $\forall i \in \{\text{r}, \text{l}\}$ | $\{\overleftarrow{\text{updt}^{\text{st}\_\text{i}}_{\text{sp}}}, \overleftarrow{\text{i}}\} \to \{\overleftarrow{\text{updt}^{\text{ist}}_{\text{sp}}}\}$ |
| | | † $\forall i \in \{\text{r}, \text{l}\}$ | $\{\overleftarrow{\text{updt}^{\text{st}\_\text{i}}_{\text{sp}}}, \overrightarrow{\text{updt}^{\text{ist}}_{\text{sp}}}\} \to \{\overrightarrow{\text{i}}, \overrightarrow{\text{updt}^{*}_{\text{dl}}}\}$ |
| | | † | $\{\overrightarrow{\text{updt}^{*}_{\text{sp}}}, \overrightarrow{\text{tree}}\} \to \{\overrightarrow{\text{tree}}, \overrightarrow{\text{updt}^{\text{hi}}_{\text{sp}}}, \overrightarrow{\text{updt}^{\text{lo}}_{\text{sp}}}\}$ |

† Each line starting with this symbol defines two rules, one with all right over arrows, one with all left.

Figure 25: Definitions for updating parameters after a ***split***.

### *5.6. Initial Configuration*

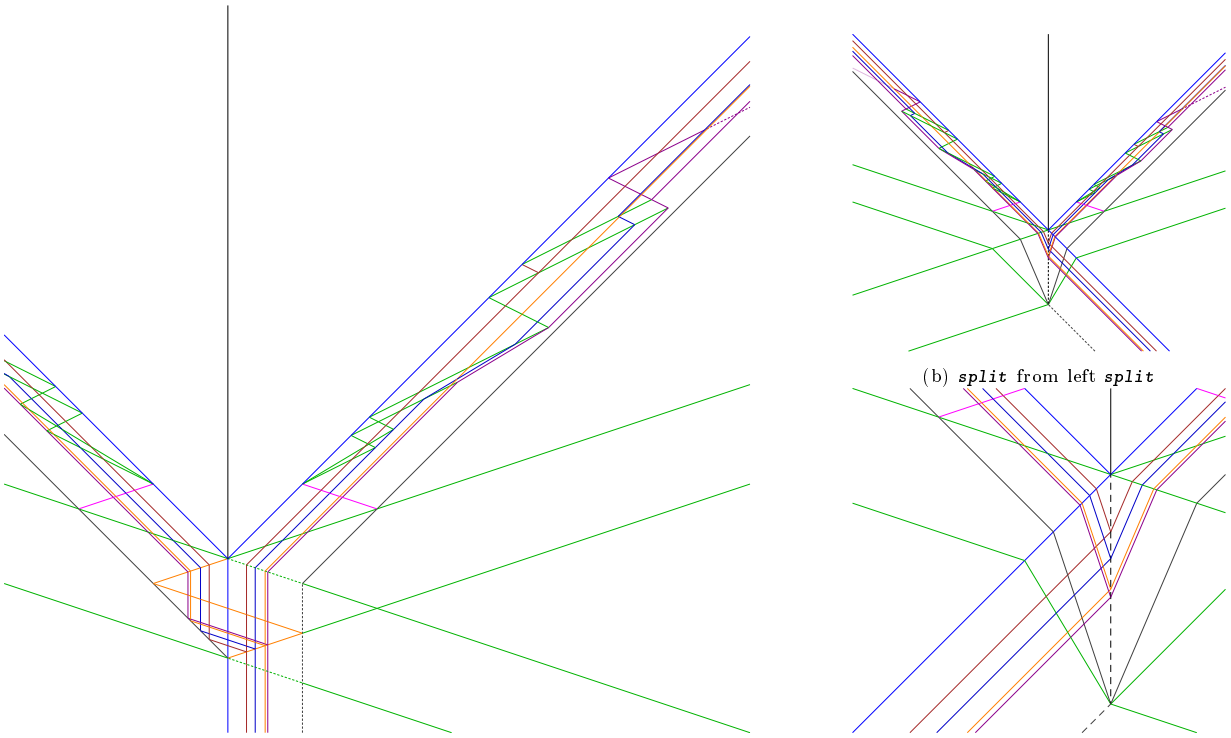
The initial configuration consists in a rightward tree macro-signal, built as per subsection 5.2, as well as two bouncing signals (non slow and going right as well), separated by three times the width of the macro-signal. A $\overrightarrow{\text{bounce}^{\text{top}}}$ signal is still inside the macro-signal, so as to prompt updating and testing. The parameters inside the macro-signal at time 0 are chosen so as to become the targeted $l$ and $r$ after update.

## 6. Conclusion

We provide a recursive geometrical algorithm to draw an infinite tree accumulating on a parameterised slope together with its proof and its implementation as a signal machine. An extended run on a signal machine can be seen in Fig. 27, with $l = 150/113$ and $r = 200/101$.

By comparison with cellular automata, the active signals (the general) start on an arbitrarily small area. It corresponds to the single active cell at the start in cellular automata.

It takes several signals to code any slope with finitely many meta-signals. It is possible to shrink the general to a single point and do similar construction to ours (for example by building bands of signals such as our). There would, however, be only one attainable slope per general (assuming its position is fixed), and finitely many per signal machine (since point generals are one of finitely many meta-signals).



(a) *split* from *delay*

(b) *split* from left *split*

(c) *split* from right *split*

Figure 26: *split* implementations.

An important model that appears in the current construction is the augmented signal machine. It allows to construct and prove at some level, leaving only "technicalities" to simulate with a usual signal machine (using macro-signals and macro-collisions). These technicalities can be involving by themselves.

Our construction exhibits rational robustness: if the signal speed and initial positions are rational, so will be the targeted slope. Further more the tree drawn is fractal (can be recursively defined) if and only if $l$ and $r$ are rational numbers. Indeed, if say, $l$ is irrational, then the path from the general to $(-1, l)$ is made of a non-periodic sequence of *delay* and *split*. If $l$ and $r$ are rational, then one can show the set of reachable parameters is finite by observing the invariant: there is some positive integer $m$ (the least common multiple of the denominators of initial $l$ and $r$) such that both $m.l$ and $m.r$ always belongs to $\mathbb{N}$, as well as the fact that $l$ and $r$ are bounded.

This construction is a first step toward a study of possible accumulation sets of signal machines. The next step is to look for curves, i.e. non piece-wise rectilinear. We can already adapt the initial configuration (without modifying the machine!) to accumulate on a continuous piece-wise linear function, as shown in Fig. 28.

## References


[1] Besson, T., 2018. Automatic discretization of signal machines into cellular automata. Thèse de doctorat. Université d'Orléans. URL: https://tel.archives-ouvertes.fr/tel-01975875.

[2] Blum, L., Shub, M., Smale, S., 1989. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bulletin of the American Mathematical Society 21, 1–46.

[3] Čulik II, K., 1989. Variations of the firing squad problem and applications. Information Processing Letters 30, 153–157. doi:10.1016/0020-0190(89)90134-8.

[4] Duchier, D., Durand-Lose, J., Senot, M., 2012. Computing in the fractal cloud: modular generic solvers for SAT and Q-SAT variants, in: Agrawal, M., Cooper, B.S., Li, A. (Eds.), Theory and Applications of Models of Computations (TAMC '12), Springer. pp. 435–447. URL: http://arxiv.org/abs/1105.3454, doi:10.1007/978-3-642-29952-0_42.

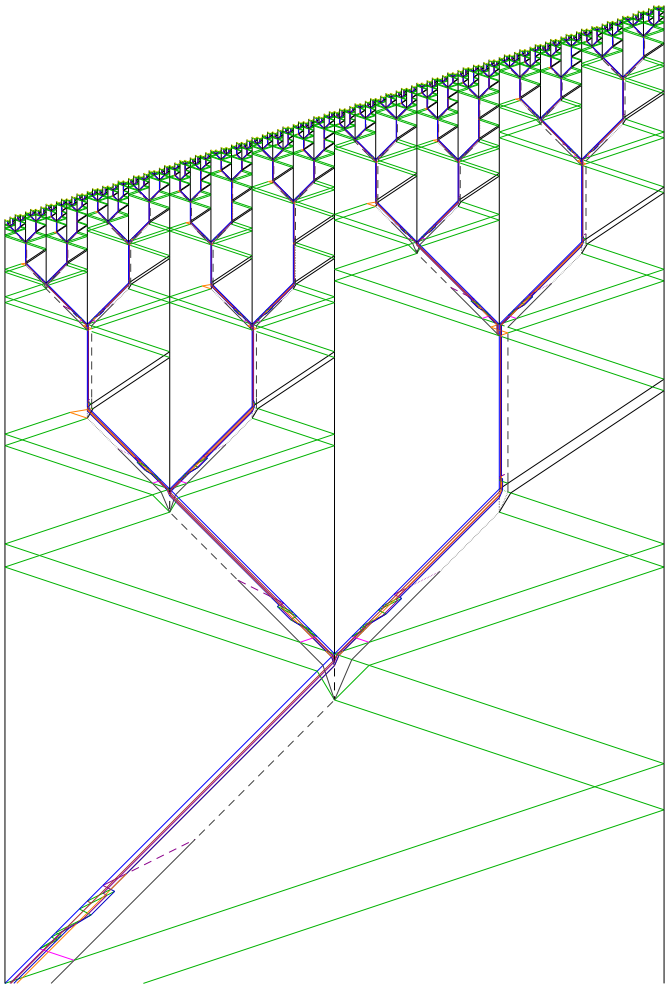


Figure 27: The whole signal machine in action involving more than 74.000 collisions.


[5] Durand-Lose, J., 2006. Forcasting black holes in abstract geometrical computation is highly unpredictable, in: Cai, J.Y., Cooper, B.S., Li, A. (Eds.), Theory and Applications of Models of Computations (TAMC '06), Springer. pp. 644–653. doi:10.1007/11750321_61.

[6] Durand-Lose, J., 2007. Abstract geometrical computation and the linear Blum, Shub and Smale model, in: Cooper, B.S., Löwe, B., Sorbi, A. (Eds.), Computation and Logic in the Real World, 3rd Conf. Computability in Europe (CiE 2007), Springer. pp. 238–247. doi:10.1007/978-3-540-73001-9_25.

[7] Durand-Lose, J., 2011a. Abstract geometrical computation 5: embedding computable analysis. Natural Computing 10, 1261–1273. doi:10.1007/s11047-010-9229-6. special issue on Unconv. Comp. '09.

[8] Durand-Lose, J., 2011b. Geometrical accumulations and computably enumerable real numbers (extended abstract), in: Calude, C.S., Kari, J., Petre, I., Rozenberg, G. (Eds.), Int. Conf. Unconventional Computation 2011 (UC '11), Springer. pp. 101–112. doi:10.1007/978-3-642-21341-0_15.

[9] Goto, E., 1962. A minimum time solution of the firing squad synchronization problem, in: Courses Notes for Applied Mathematics, Harvard University.

[10] Maignan, L., Yunès, J., 2012. A spatio-temporal algorithmic point of view on firing squad synchronisation problem, in: Sirakoulis, G.C., Bandini, S. (Eds.), 10th Int. Conf. on Cellular Automata for Research and Industry (ACRI), Santorini Island, Greece, Springer. pp. 101–110. doi:10.1007/978-3-642-33350-7_11.

[11] Maignan, L., Yunès, J., 2016a. A field based solution of mazoyer's FSSP schema, in: Yacoubi, S.E., Was, J., Bandini, S. (Eds.), 12th Int. Conf. on Cellular Automata for Research and Industry (ACRI), Morocco, Springer. pp. 134–143. doi:10.1007/978-3-319-44365-2_13.

[12] Maignan, L., Yunès, J., 2016b. Finitization of infinite field-based multi-general FSSP solution. J. Cellular Automata 12, 121–139.

[13] Maignan, L., Yunès, J., 2018. Generalized FSSP on two triangular tilings, in: Sixth International Symposium on Computing

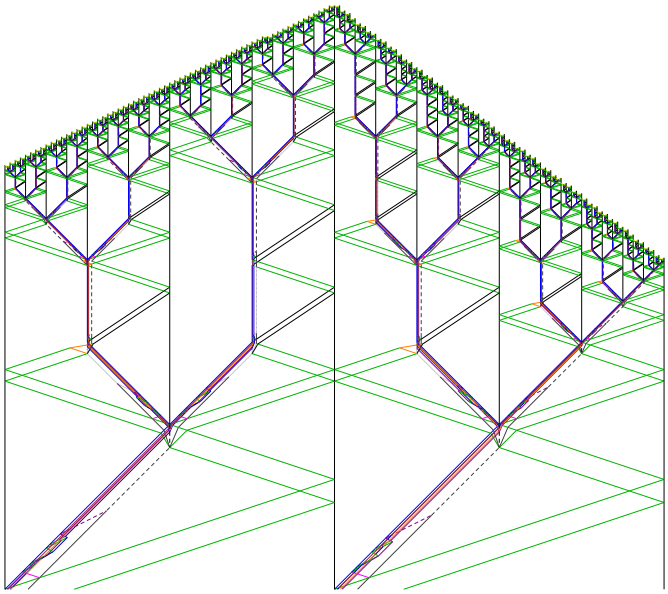


Figure 28: Two different slopes.

and Networking, CANDAR Workshops, Takayama, Japan, IEEE Computer Society. pp. 27–31. doi:10.1109/CANDARW.2018.00013.

[14] Martin, B., 1994. A universal cellular automaton in quasi-linear time and its S-n-m form. Theoretical Computer Science 123, 199–237.

[15] Mazoyer, J., 1987. A 6-states minimal-time solution to the Firing squad synchronisation problem. Theoretical Computer Science 50, 183–237.

[16] Mazoyer, J., 1996. On optimal solutions to the Firing squad synchronization problem. Theoretical Computer Science 168, 367–404. doi:10.1016/S0304-3975(96)00084-9.

[17] Moore, E.F., 1964. Sequential machines, Selected papers. Addison-Wesley.

[18] Nichitiu, C., Mazoyer, J., Rémila, E., 2001. Algorithms for leader election by cellular automata. Journal of Algorithms 41, 302 – 329. URL: http://www.sciencedirect.com/science/article/pii/S0196677401911757, doi:https://doi.org/10.1006/jagm.2001.1175.

[19] Umeo, H., 2017. Cellular automata, firing squad synchronization problem in, in: Meyers, R.A. (Ed.), Encyclopedia of Complexity and Systems Science. Springer Berlin Heidelberg, pp. 1–60. doi:10.1007/978-3-642-27737-5_211-4.

[20] Waksman, A., 1966. An optimum solution to the firing squad synchronization problem. Information and Control 9, 66–78.

[21] Yunès, J.B., 2007. Simple new algorithms which solve the firing squad synchronization problem: a 7-states 4n-steps solution, in: Durand-Lose, J., Margenstern, M. (Eds.), Machine, Computations and Universality (MCU 2007), Springer. pp. 316–324.

bibtex entry

```
@article{durand-lose+emmanuel21tcs,
  doi = {10.1016/j.tcs.2021.06.009},
  arxiv = {https://arxiv.org/abs/2106.11176},
  note = {arXiv 2106.11176},
  author = {Durand-Lose, J{\'e}r{\^o}me and
                Emmanuel, Aur{\'e}lien},
  title = {Abstract Geometrical Computation 11: Slanted Firing
                Squad Synchronisation on Signal Machines},
  journal = {Theoretical Computer Science},
  year = {2021},
  volume = {894},
  pages = {103--120},
  language = {english}
}
```