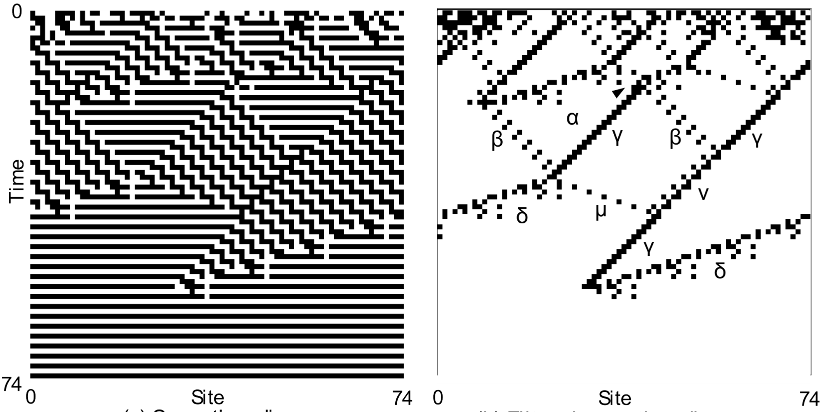
Computing with signals: a generic and modular signal machine for satisfiability problems

Denys Duchier, Jérôme Durand-Lose, Maxime Senot

Laboratoire d'Informatique Fondamentale d'Orléans,
University of Orléans, Orléans, FRANCE

2^{nd} international workshop *NWC '11*
LIFO, Orléans — 24 May 2011

Analyzing CA with signals



(a) Space-time diagram. (b) Filtered space-time diagram.

[Das, Crutchfield, Mitchell 95]

Designing CA with signals

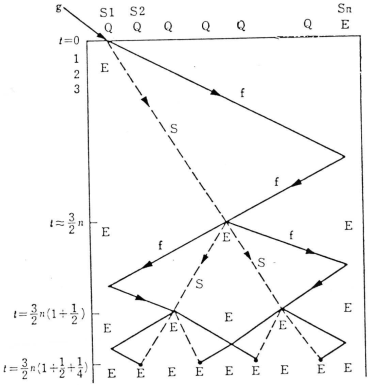


図 3·5 一斉射撃の問題（連続近似）

G	s_1	s_2	s_3	s_4	s_5	s_6
g	Q	Q	Q	Q	Q	E
$t=0$	f's'Efs	Q	Q	Q	Q	E
1	E	Q2f	Q	Q	Q	E
2	E	Q1	Qf	Q	Q	E
3	E	Q&	Q	Qf	Q	E
4	E	Q	Q2	Q	Qf	E
5	E	Q	Q1	Q	Q	f'Ef
6	E	Q	QS	Q	f'Q	E
7	E	Q	Q	a'Q*	Q	E
8	E	Q	f'S'ESf	f's'Esf	Q	E
9	E	f'2Q	E	E	Q2f	E
10	f'Ef	1Q	E	E	Q1	f'Ef
11	E	f'S'ESf	E	E	f's'Esf	E
12	a'Ea	E	a'Ea	a'Ea	E	a'Ea
13	F	F	F	F	F	F

図 3·6 一斉射撃解 (n=6)

Goto's solution to the Firing Squad Synchronization Problem [Goto66]

Designing CA with signals

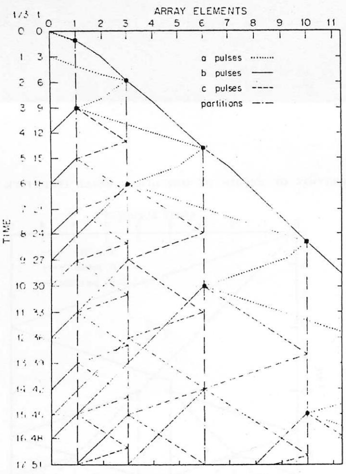
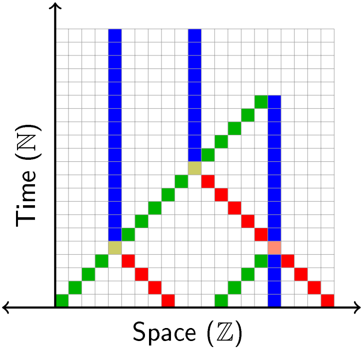
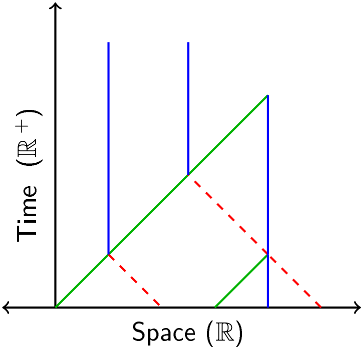


Fig. 2. Solution to the prime problem

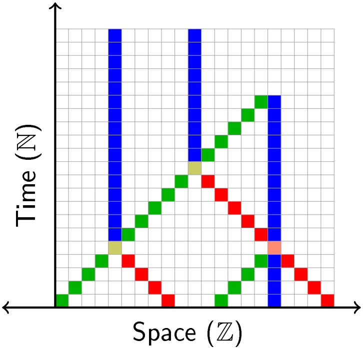
Generating primes [Fischer, 1965, Fig. 2]

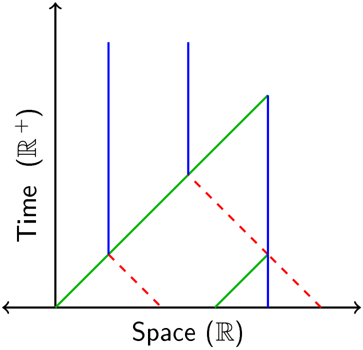
From cellular automata to signal machines





From cellular automata to signal machines





⇒ *From a discrete to a continuous* space-time

Computing the middle

Meta-signals

M (0)

Collision rules

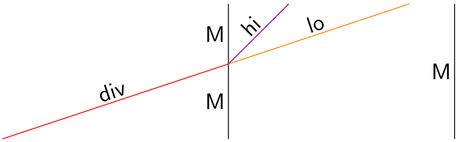
Computing the middle

Meta-signals

M (0)
div (3)

Collision rules

Computing the middle



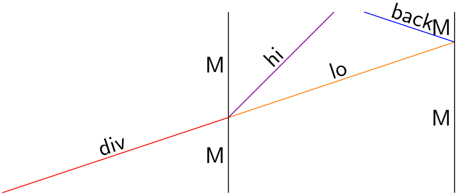
Meta-signals

M (0)
div (3)
hi (1)
lo (3)

Collision rules

{ div,M } → { M,hi,lo }

Computing the middle



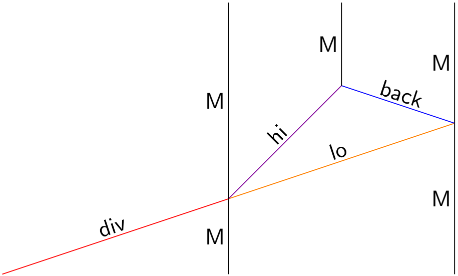
Meta-signals

M (0)
div (3)
hi (1)
lo (3)
back (-3)

Collision rules

{ div,M } → { M,hi,lo }
{ lo,M } → { back,M}

Computing the middle



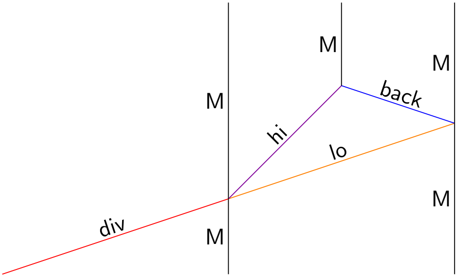
Meta-signals

M (0)
div (3)
hi (1)
lo (3)
back (-3)

Collision rules

{ div,M } $\longrightarrow$ { M,hi,lo }
{ lo,M } $\longrightarrow$ { back,M}
{ hi,back } $\longrightarrow$ { M }

Computing the middle



Meta-signals

M (0)
div (3)
hi (1)
lo (3)
back (-3)

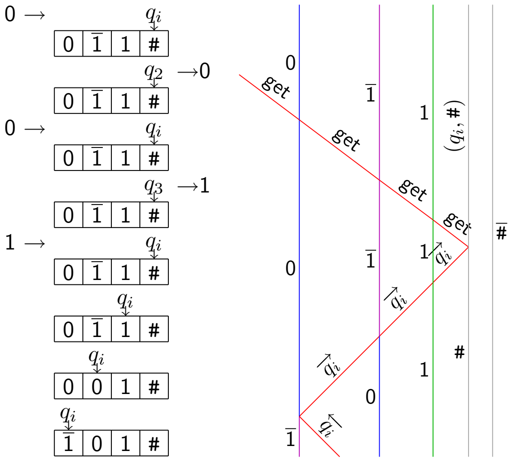
Signal Machine

1. Meta-signal (speed)
2. Collision rules

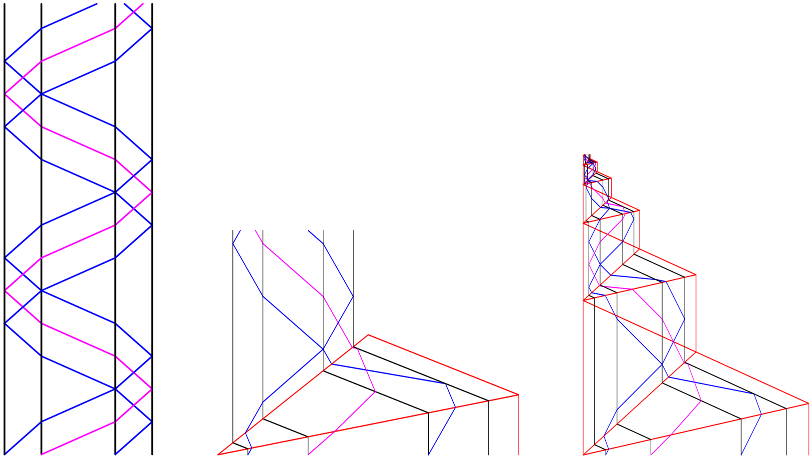
Collision rules

{ div,M } $\rightarrow$ { M,hi,lo }
{ lo,M } $\rightarrow$ { back,M}
{ hi,back } $\rightarrow$ { M }

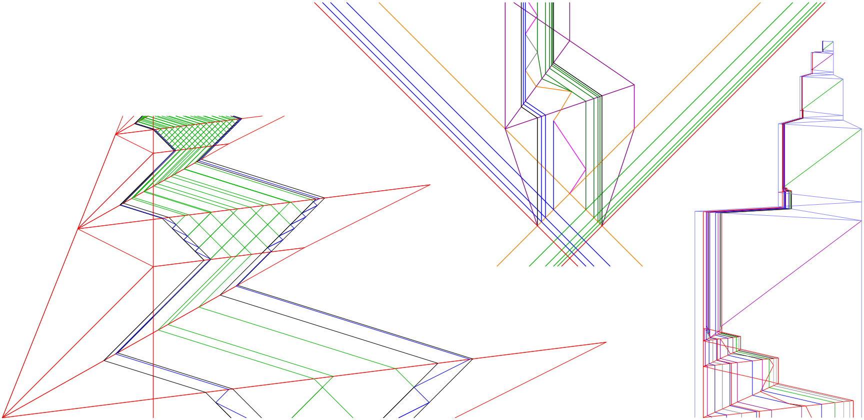
Church-Turing computing

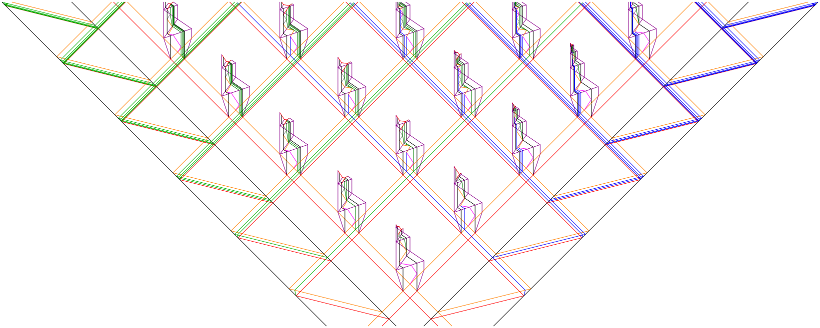


Scaling down and bounding the duration

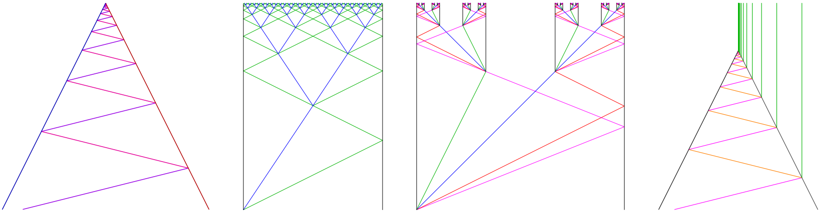


Other examples





Examples of *Accumulations* and *Zeno's paradox*



Decision problem **Q-SAT**

Input : a quantified boolean formula ϕ.
Question : Is ϕ true or false?

Decision problem Q-SAT

Input : a quantified boolean formula ϕ.
Question : Is ϕ true or false?

Example: the formula $\phi = \exists x_1 \forall x_2 \forall x_3 \ \ (x_1 \wedge \neg x_2) \vee x_3$ is *false*.

Decision problem **Q-SAT**

Input : a quantified boolean formula ϕ.
Question : Is ϕ true or false?

Example: the formula $\phi = \exists x_1 \forall x_2 \forall x_3 \ \ (x_1 \wedge \neg x_2) \vee x_3$ is *false*.

Theorem [Stockmeyer,1973]

Q-SAT is **PSPACE**-complete.

Decision problem **Q-SAT**

Input : a quantified boolean formula ϕ.
Question : Is ϕ true or false?

Example: the formula $\phi = \exists x_1 \forall x_2 \forall x_3 \ \ (x_1 \wedge \neg x_2) \vee x_3$ is *false*.

Theorem [Stockmeyer,1973]

Q-SAT is **PSPACE**-complete.

On our classical model of computation at usual cost.

Brute-force solution to Q-SAT

Let *qsat* be the recursive algorithm defined by:

- $qsat(\exists x\ \psi) = qsat(\psi[x \leftarrow \text{false}]) \vee qsat(\psi[x \leftarrow \text{true}])$

Brute-force solution to Q-SAT

Let *qsat* be the recursive algorithm defined by:

- $qsat(\exists x\ \psi) = qsat(\psi[x \leftarrow \text{false}]) \vee qsat(\psi[x \leftarrow \text{true}])$
- $qsat(\forall x\ \psi) = qsat(\psi[x \leftarrow \text{false}]) \wedge qsat(\psi[x \leftarrow \text{true}])$

Brute-force solution to Q-SAT

Let *qsat* be the recursive algorithm defined by:

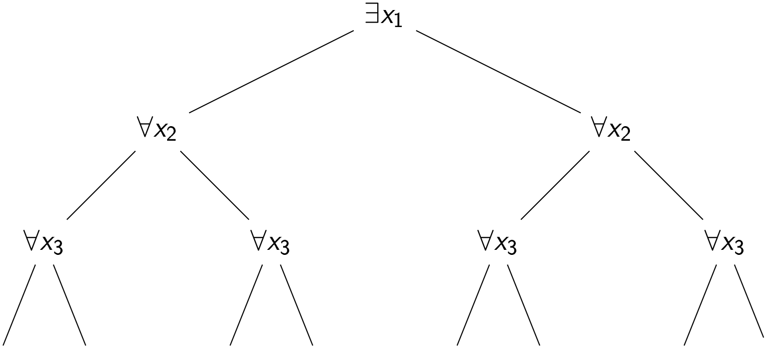
- $qsat(\exists x\ \psi) = qsat(\psi[x \leftarrow \text{false}]) \vee qsat(\psi[x \leftarrow \text{true}])$
- $qsat(\forall x\ \psi) = qsat(\psi[x \leftarrow \text{false}]) \wedge qsat(\psi[x \leftarrow \text{true}])$
- $qsat(\beta) = eval(\beta)$ if β is a ground boolean formula.

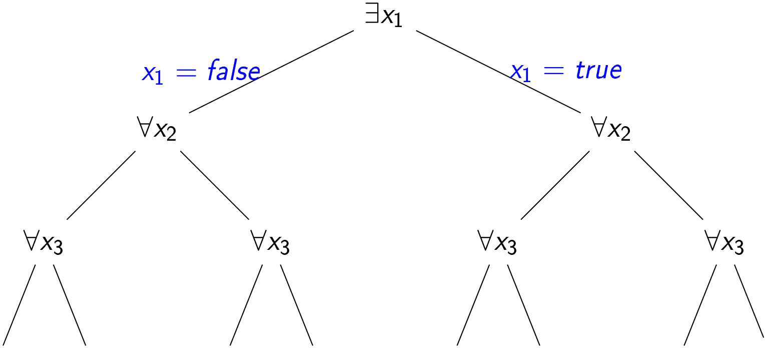
Brute-force solution to Q-SAT

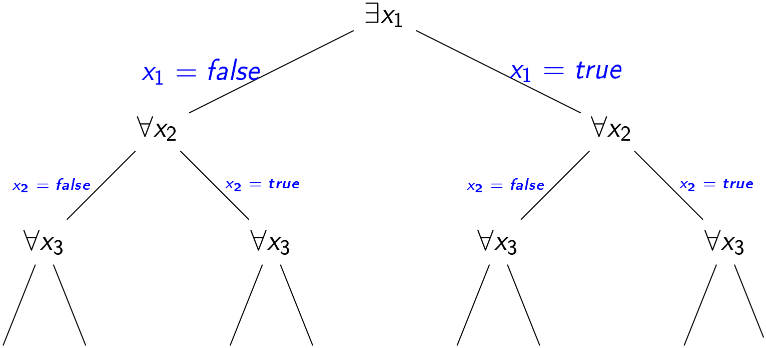
Let *qsat* be the recursive algorithm defined by:

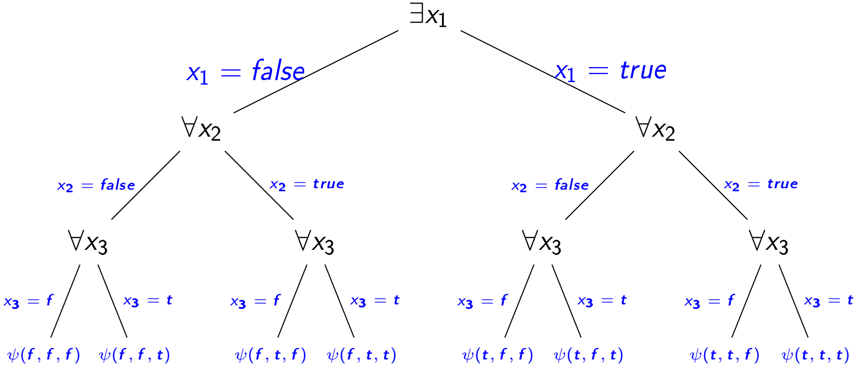
- $qsat(\exists x\ \psi) = qsat(\psi[x \leftarrow \text{false}]) \vee qsat(\psi[x \leftarrow \text{true}])$
- $qsat(\forall x\ \psi) = qsat(\psi[x \leftarrow \text{false}]) \wedge qsat(\psi[x \leftarrow \text{true}])$
- $qsat(\beta) = eval(\beta)$ if β is a ground boolean formula.

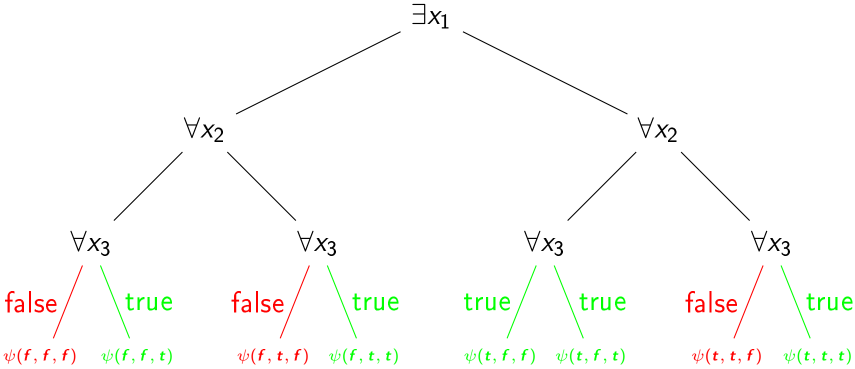
Then *qsat* solves the problem Q-SAT with exponential time and polynomial space.


$\exists x_1$
$\forall x_2$
$\forall x_2$
$\forall x_3$
$\forall x_3$
$\forall x_3$
$\forall x_3$

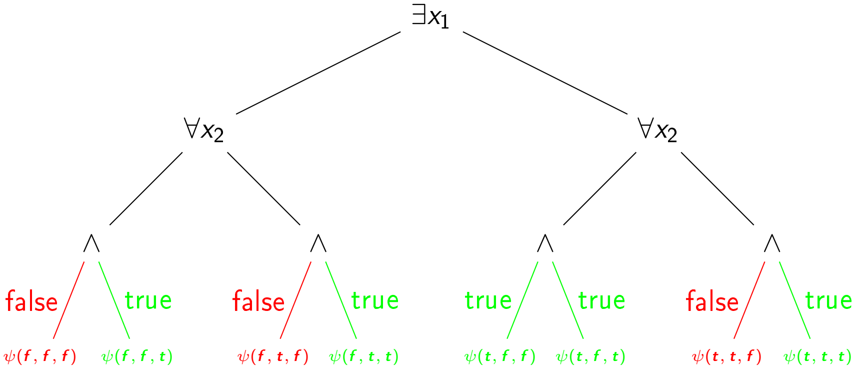

$\exists x_1$
$x_1 = false$
$x_1 = true$
$\forall x_2$
$\forall x_2$
$\forall x_3$
$\forall x_3$
$\forall x_3$
$\forall x_3$


$\exists x_1$
$x_1 = false$
$x_1 = true$
$\forall x_2$
$\forall x_2$
$x_2 = false$
$x_2 = true$
$x_2 = false$
$x_2 = true$
$\forall x_3$
$\forall x_3$
$\forall x_3$
$\forall x_3$

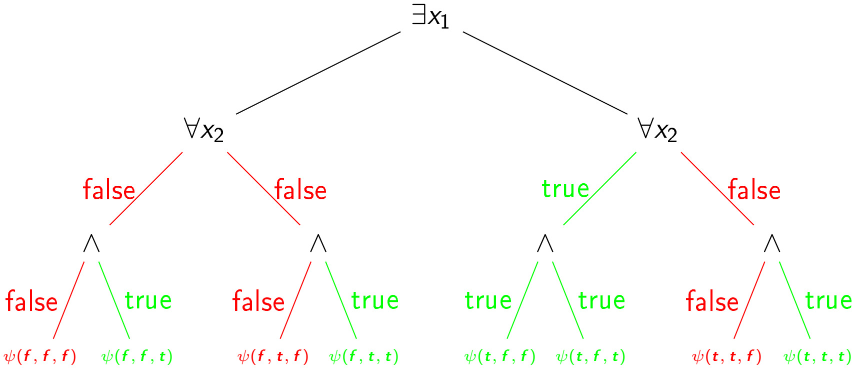

∃x1
x1 = false
x1 = true
∀x2
∀x2
x2 = false
x2 = true
x2 = false
x2 = true
∀x3
∀x3
∀x3
∀x3
x3 = f
x3 = t
x3 = f
x3 = t
x3 = f
x3 = t
x3 = f
x3 = t
ψ(f, f, f)
ψ(f, f, t)
ψ(f, t, f)
ψ(f, t, t)
ψ(t, f, f)
ψ(t, f, t)
ψ(t, t, f)
ψ(t, t, t)



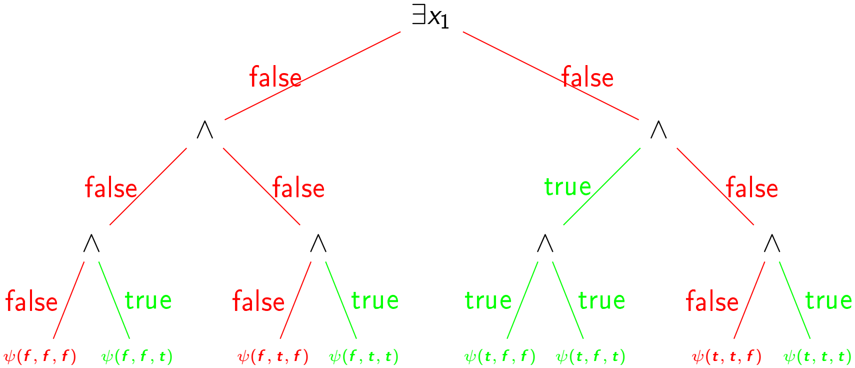
$\phi = \exists x_1 \forall x_2 \forall x_3 \;\; \psi(x_1, x_2, x_3)$ where $\psi = (x_1 \wedge \neg x_2) \vee x_3$



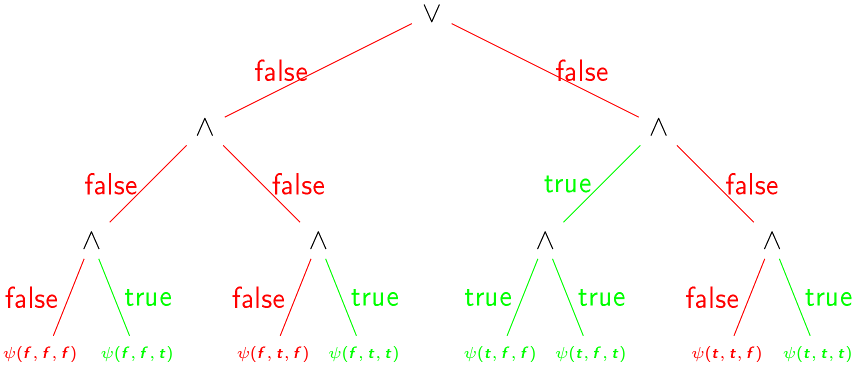
$\phi = \exists x_1 \forall x_2 \forall x_3 \;\; \psi(x_1, x_2, x_3)$ where $\psi = (x_1 \wedge \neg x_2) \vee x_3$



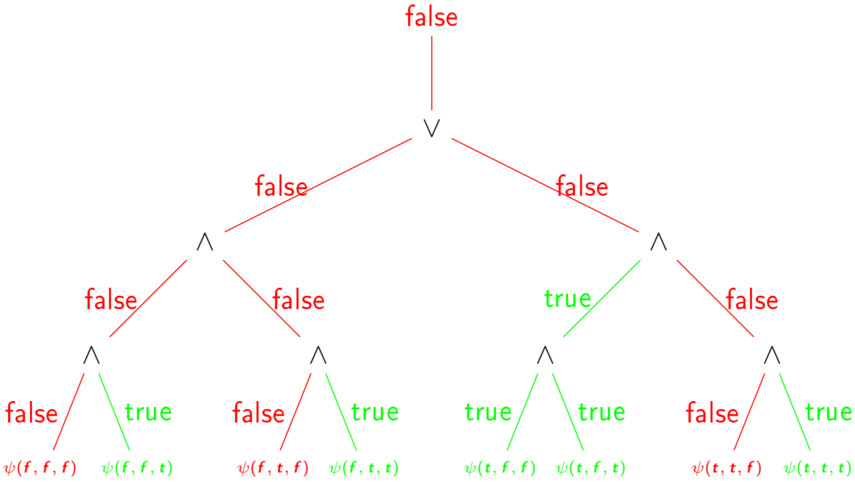
$\phi = \exists x_1 \forall x_2 \forall x_3 \ \ \psi(x_1, x_2, x_3)$ where $\psi = (x_1 \wedge \neg x_2) \vee x_3$



$\phi = \exists x_1 \forall x_2 \forall x_3 \quad \psi(x_1, x_2, x_3)$ where $\psi = (x_1 \wedge \neg x_2) \vee x_3$

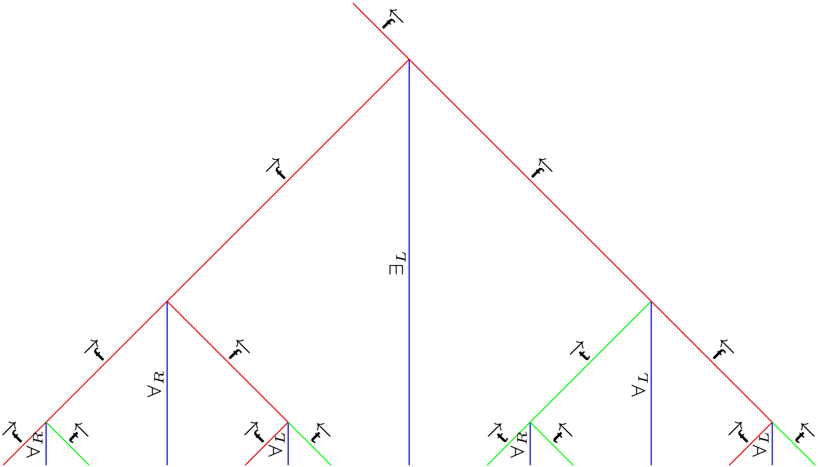


$\phi = \exists x_1 \forall x_2 \forall x_3 \;\; \psi(x_1, x_2, x_3)$ where $\psi = (x_1 \wedge \neg x_2) \vee x_3$

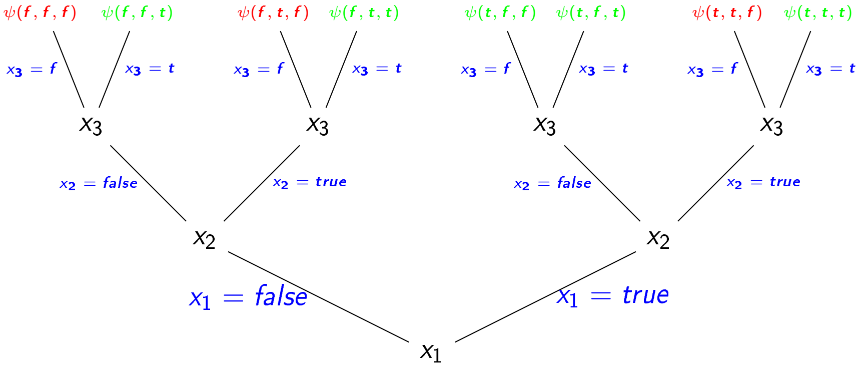


$\phi = \exists x_1 \forall x_2 \forall x_3 \;\; \psi(x_1, x_2, x_3)$ where $\psi = (x_1 \wedge \neg x_2) \vee x_3$

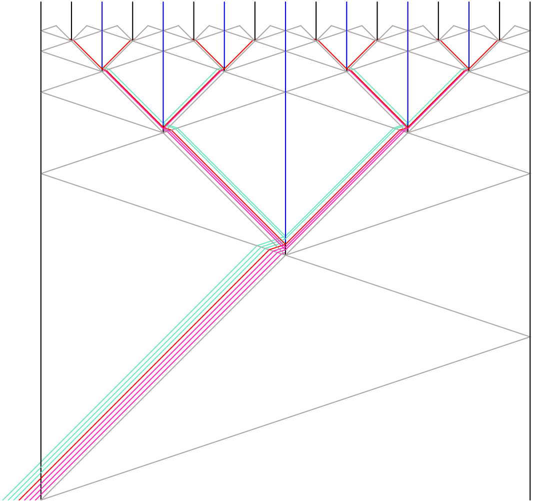
Collecting the results with signals



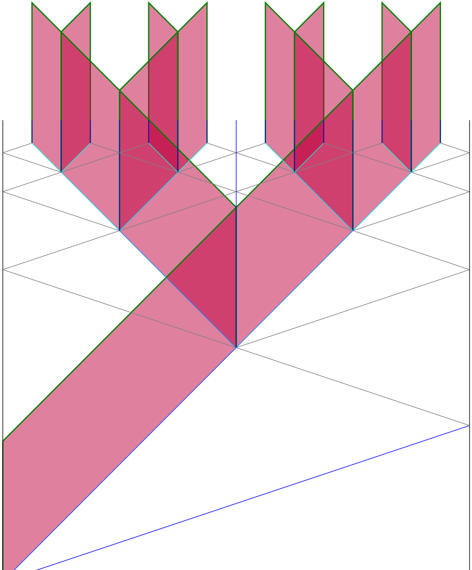
Trying all possible cases



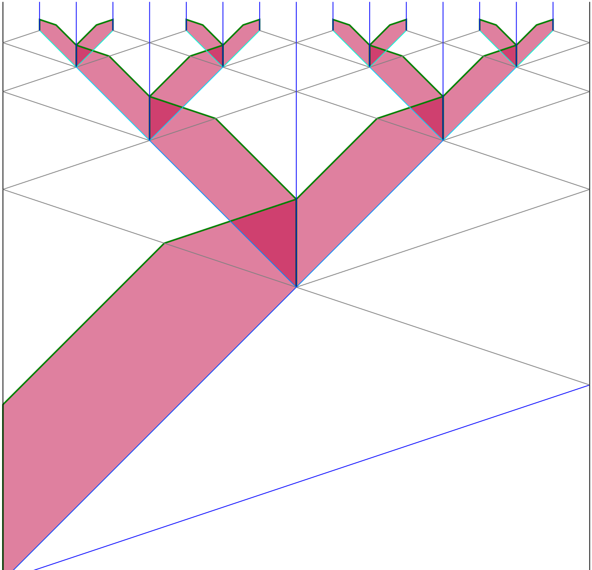
Building the tree / combinatorial comb



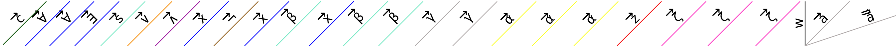
Propagation lanes without scaling



The lens device

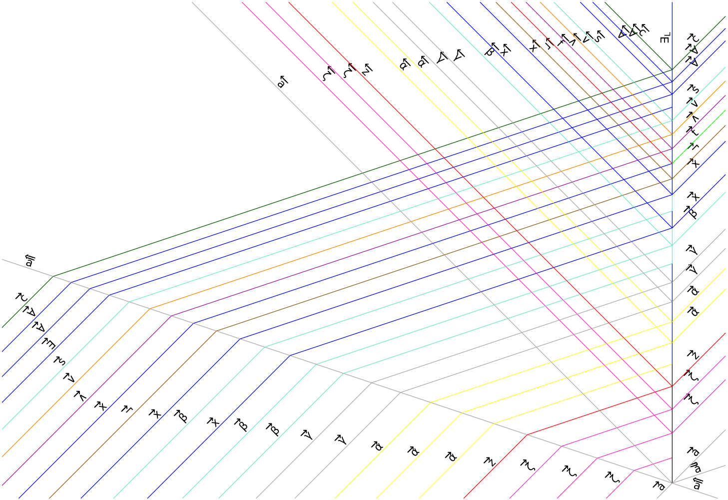


Initial configuration by *modules*



[red($Q_i x_i$)] [map(ψ)] [decide(n)] [until(n)][start]

Propagating the beam

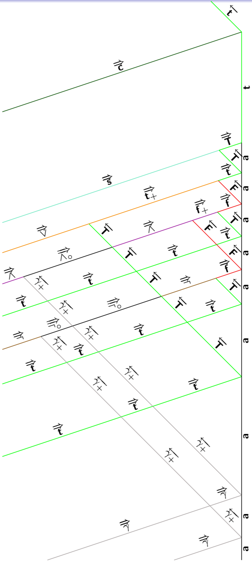


Formula evaluation

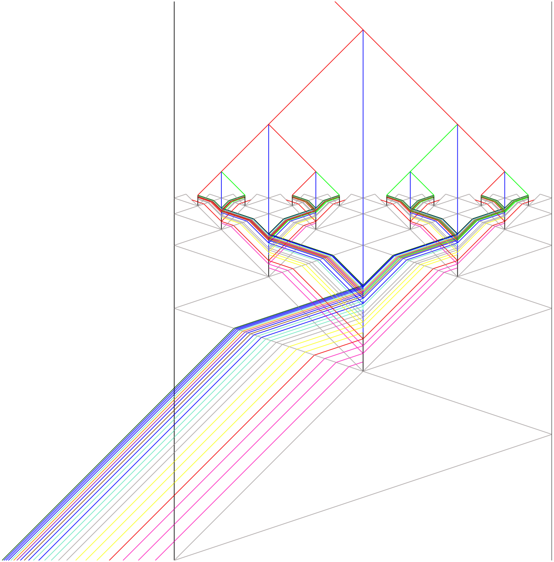
$$\phi = \exists x_1 \forall x_2 \forall x_3 \quad (x_1 \wedge \neg x_2) \vee x_3$$

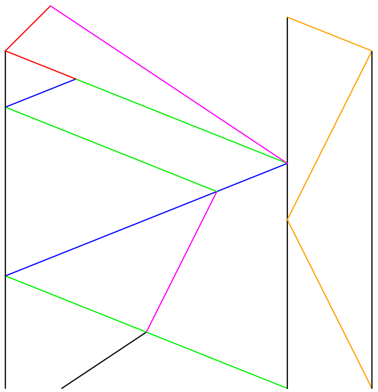
Case here

true ∧ (¬true ∨ true)



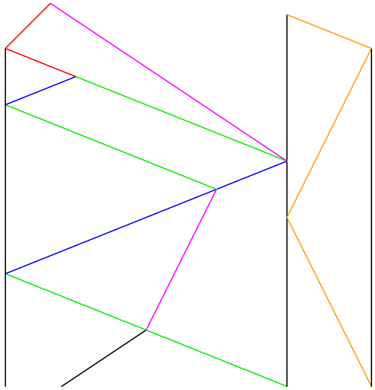
The whole diagram





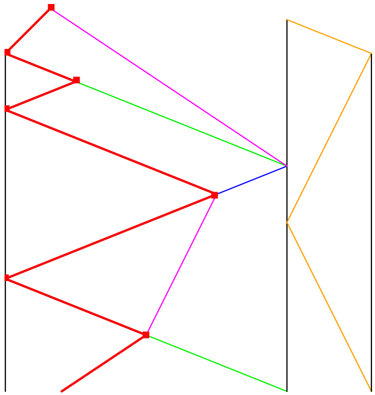
Time complexity

Length of the maximal chain.



Time complexity

Length of the maximal chain.
We speak of *collision depth*.

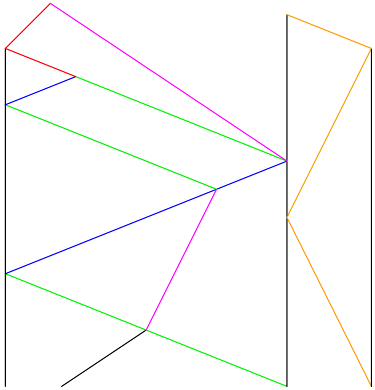


Time complexity

Length of the maximal chain.
We speak of *collision depth*.

Space complexity

Maximal number of signals existing simultaneously.

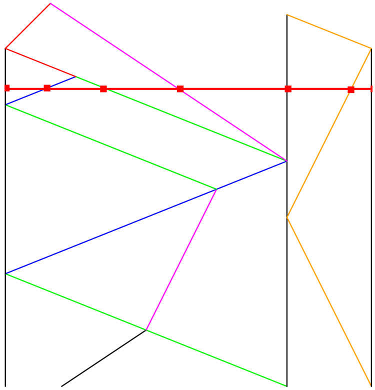


Time complexity

Length of the maximal chain.
We speak of *collision depth*.

Space complexity

Maximal number of signals existing simultaneously.



Conclusion

Results

Q-SAT can be solved in cubic depth by a single signal machine.

Conclusion

Results

Q-SAT can be solved in cubic depth by a single signal machine.
With the modular approach, we can also provide in cubic collision depth (and bounded space and time) signal machines for:

- **SAT** (special instance of **Q-SAT**)
- **#SAT**
- **MAX-SAT**
- **ENUM-SAT** (enumerating all solutions of **SAT**)

Future work

Future work and Perspectives

- Looking for other complexity classes (**EXPTIME**,. . .) or other hard problems?

Future work

Future work and Perspectives

- Looking for other complexity classes (**EXPTIME**,. . .) or other hard problems?
- Generating and using automatically other fractal structures? e.g. what computations can be inserted in Cantor's triadic?

Future work

Future work and Perspectives

- Looking for other complexity classes (**EXPTIME**,. . .) or other hard problems?
- Generating and using automatically other fractal structures? e.g. what computations can be inserted in Cantor's triadic?
- Defining formally the notion of *geometrical programmation by modules*?

Future work

Future work and Perspectives

- Looking for other complexity classes (**EXPTIME**,...) or other hard problems?
- Generating and using automatically other fractal structures? e.g. what computations can be inserted in Cantor's triadic?
- Defining formally the notion of *geometrical programmation by modules*?
- Defining complexity classes for signal machines?

Thanks for listening