GP-SAT: a Framework for Parallel SAT Solving

Ludovic Le Frioux [1,2] Souheib Baarir [1,2,3] Julien Sopena [2]
Fabrice Kordon [2]

LRDE, EPITA, Kremlin-Bicêtre, France

Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, Paris, France
CNRS, UMR 7606, LIP6, Paris, France

Université Paris Nanterre, France

MeFoSyLoMa – March 31, 2017

Context

How does it work?

- **Input** : a boolean logic formula
- **Output** : SAT (+ solution) or UNSAT

Context

How does it work?

- **Input** : a boolean logic formula
- **Output** : SAT (+ solution) or UNSAT

What is it used for?

- Planning decision
- Theorem proving
- Computational biology
- Artificial intelligence
- **Model checking**
- etc.

Context

How does it work?

- **Input** : a boolean logic formula
- **Output** : SAT (+ solution) or UNSAT

What is it used for?

- Planning decision
- Theorem proving
- Computational biology
- Artificial intelligence
- **Model checking**
- etc.

SAT is **NP-complete** [Coo71]

CPU greedy + low memory consumption $\rightarrow$ parallelization.

My objective is to improve the parallelization of SAT solving.

Boolean SATisfiability Problem

φ is a formula in **conjunctive normal from** (CNF).

$\varphi = \omega_1 \wedge \ldots \wedge \omega_n$, where ω_i are **clauses**.

Boolean SATisfiability Problem

φ is a formula in **conjunctive normal from** (CNF).

$\varphi = \omega_1 \wedge \ldots \wedge \omega_n$, where ω_i are **clauses**.
$\omega = \ell_1 \vee \ldots \vee \ell_k$, where ℓ_i are **literals**.

Boolean SATisfiability Problem

φ is a formula in **conjunctive normal from** (CNF).

$\varphi = \omega_1 \wedge \ldots \wedge \omega_n$, where ω_i are **clauses**.
$\omega = \ell_1 \vee \ldots \vee \ell_k$, where ℓ_i are **literals**.
$\ell = x$ or $\ell = \neg x$, where x is a **boolean variable**.

Boolean SATisfiability Problem

φ is a formula in **conjunctive normal from** (CNF).

$\varphi = \omega_1 \wedge \ldots \wedge \omega_n$, where ω_i are **clauses**.
$\omega = \ell_1 \vee \ldots \vee \ell_k$, where ℓ_i are **literals**.
$\ell = x$ or $\ell = \neg x$, where x is a **boolean variable**.

We can see formulas and clauses as sets.

Boolean SATisfiability Problem

φ is a formula in **conjunctive normal from** (CNF).

$\varphi = \omega_1 \wedge \ldots \wedge \omega_n$, where ω_i are **clauses**.
$\omega = \ell_1 \vee \ldots \vee \ell_k$, where ℓ_i are **literals**.
$\ell = x$ or $\ell = \neg x$, where x is a **boolean variable**.

We can see formulas and clauses as sets.

When is φ SAT?

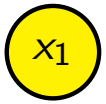
- $\varphi = \mathit{true}$ iif $\forall \omega \in \varphi, \omega = \mathit{true}$
- $\omega = \mathit{true}$ iif $\exists \ell \in \omega, \ell = \mathit{true}$

Naive approach

$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

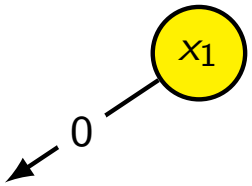
Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

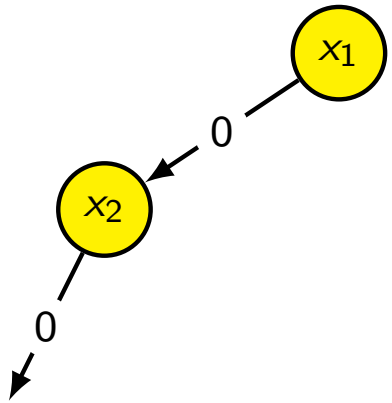
Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

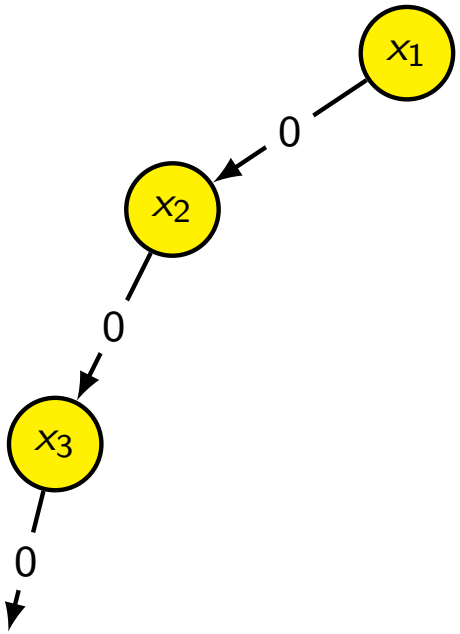
Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

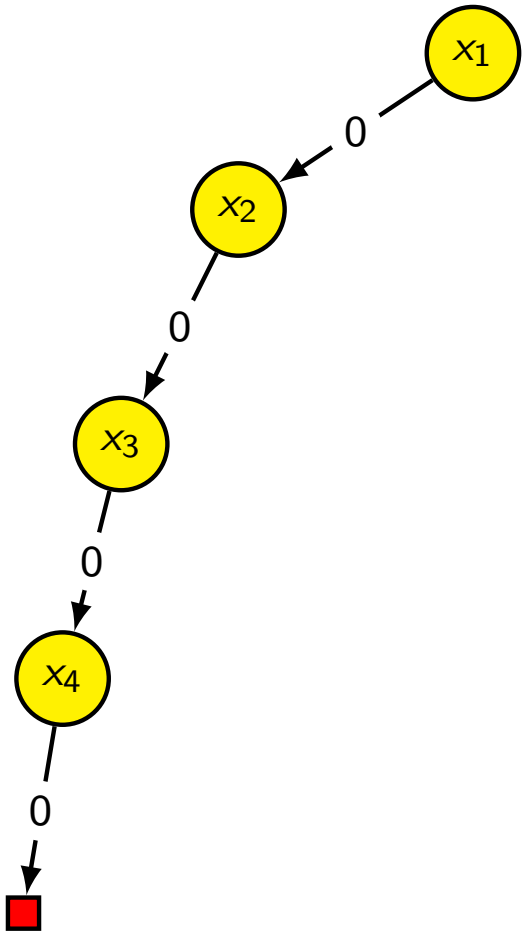
Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

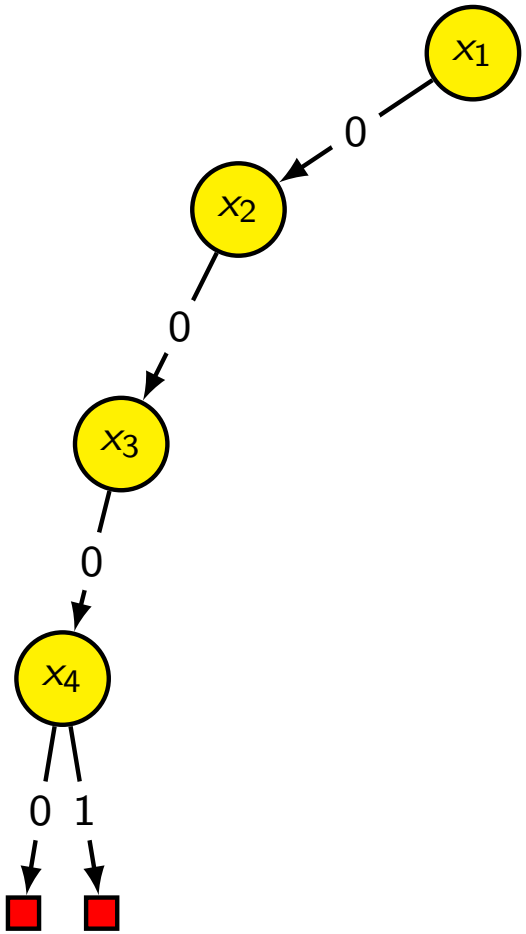
Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

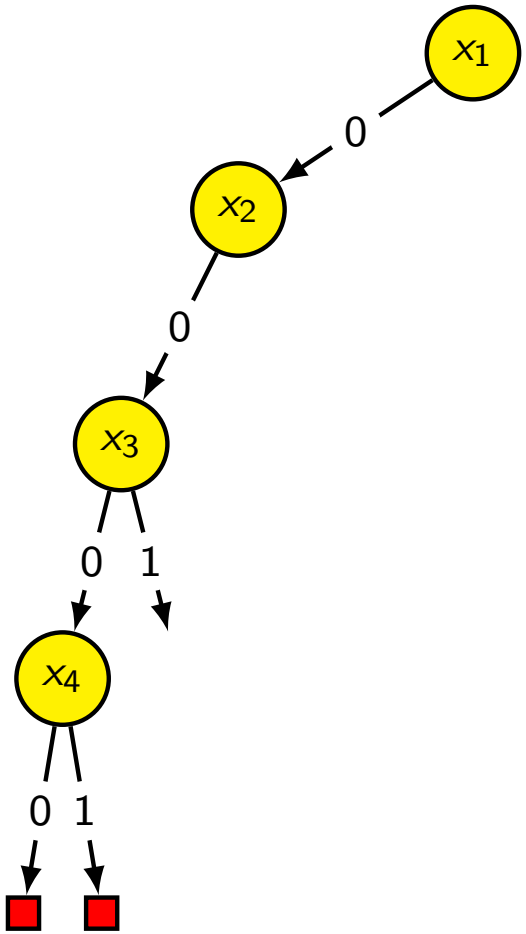
Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

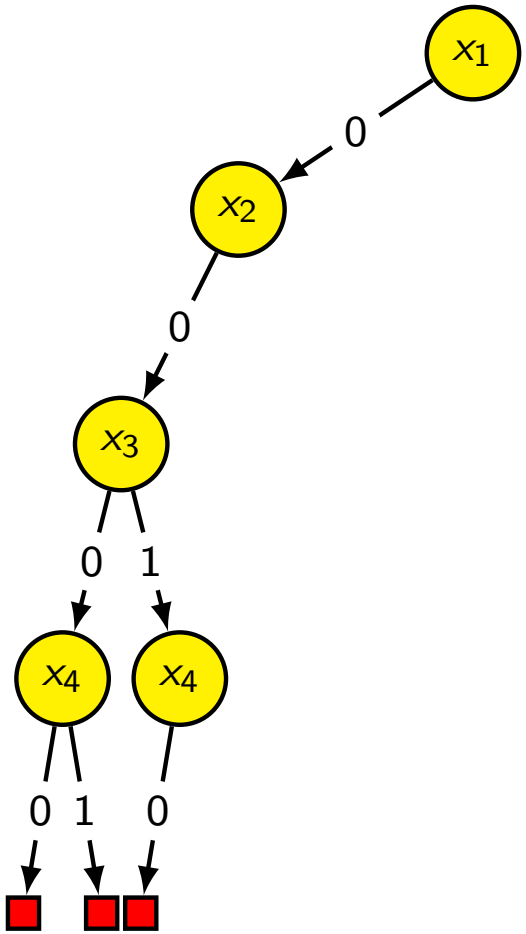
Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

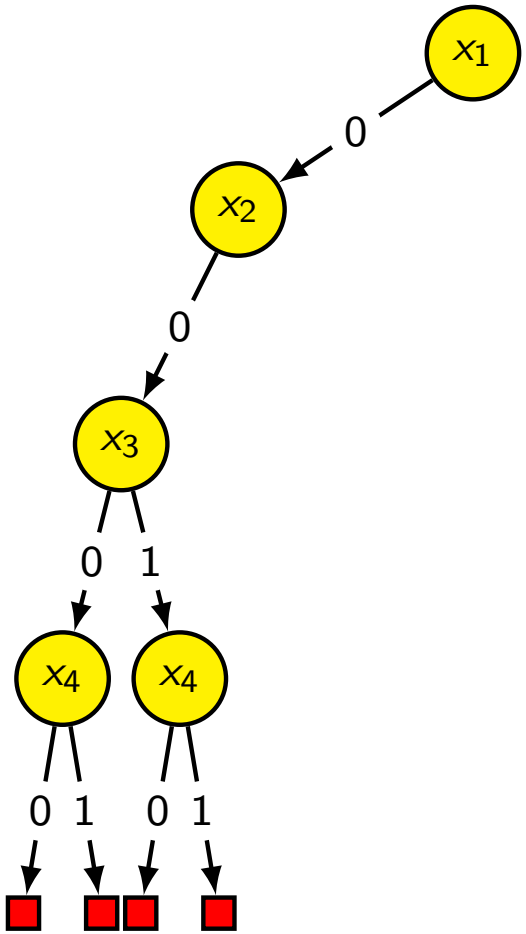
Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

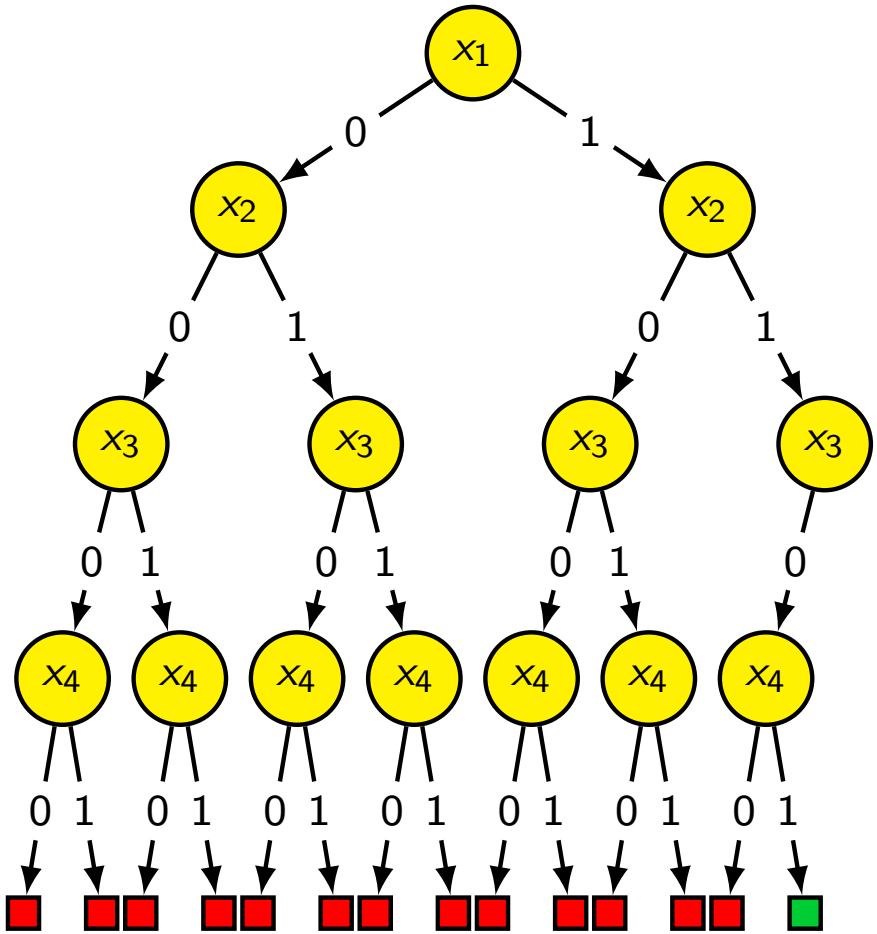
Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

Naive approach



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

Sequential Resolution

Algorithms

- Davis, Putnam, Logemann, and Loveland (**DPLL**) [DP60, DLL62]
- Conflict Driven Clause Learning (**CDCL**) [MSS$^+$99, ZMMM01]
  - Derived from DPLL

Sequential Resolution

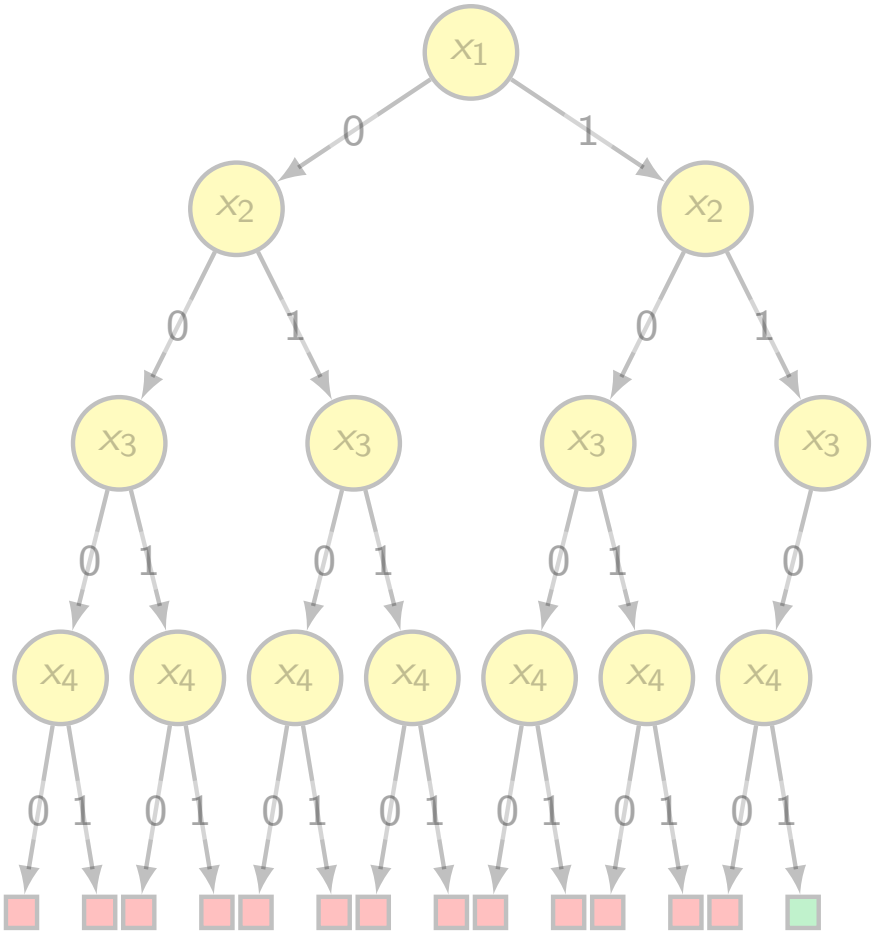
Algorithms

- Davis, Putnam, Logemann, and Loveland (**DPLL**) [DP60, DLL62]
- Conflict Driven Clause Learning (**CDCL**) [MSS$^+$99, ZMMM01]
  - Derived from DPLL

Efficiency in practice

- Unit propagation
- Decision and polarity
- Clause learning
- Restart
- Preprocessing

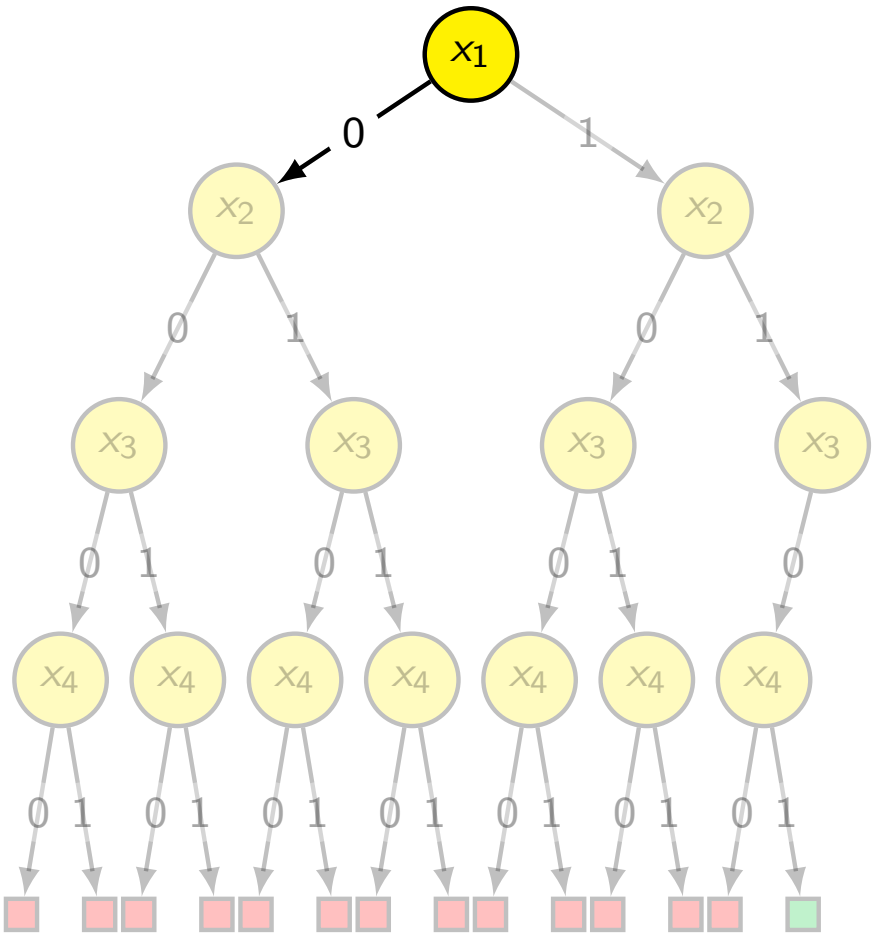
Unit Propagation



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

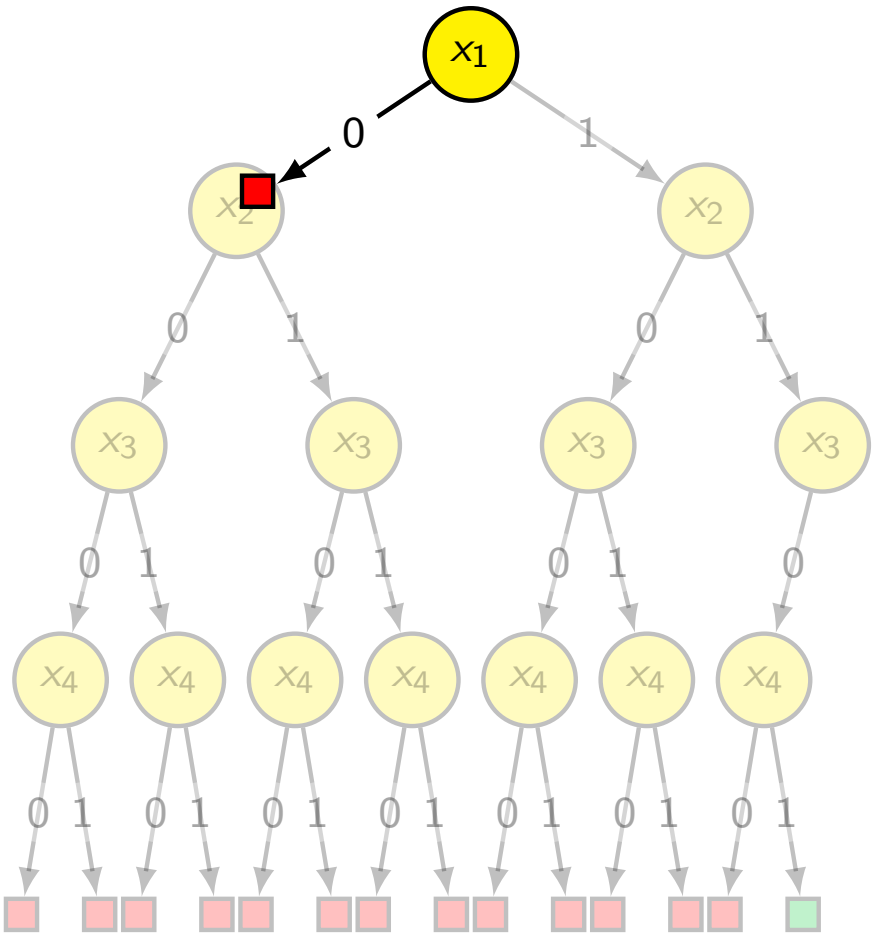
Unit Propagation



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

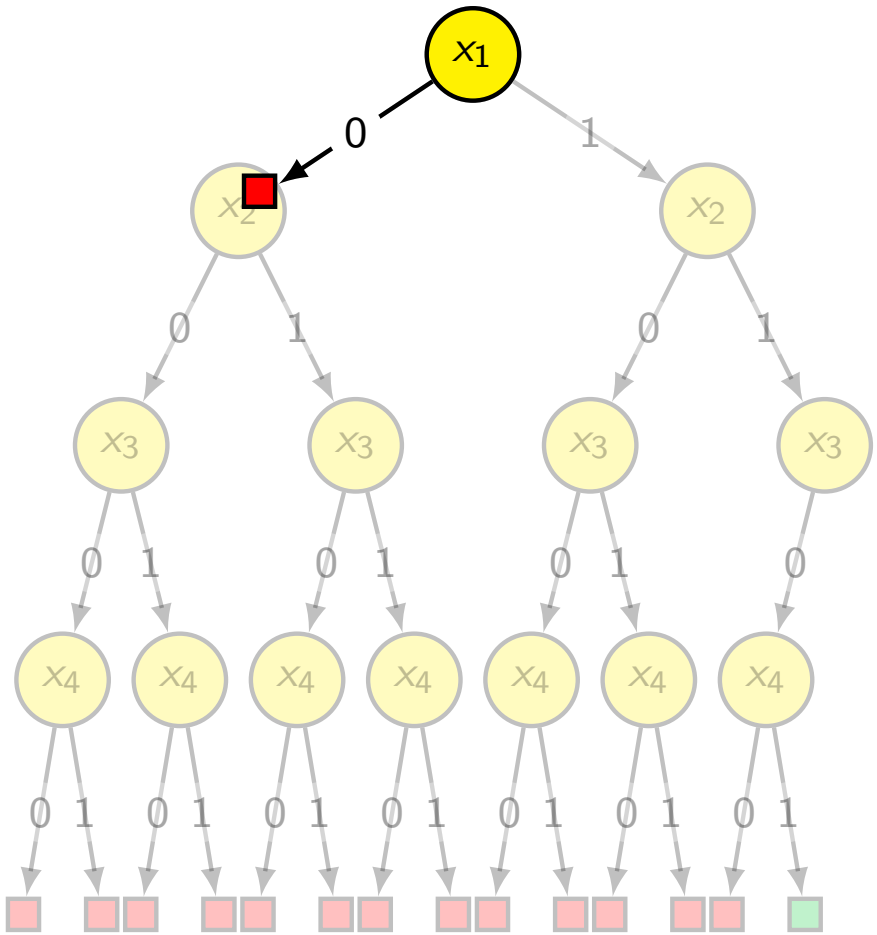
Unit Propagation



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

Unit Propagation & Clause Learning



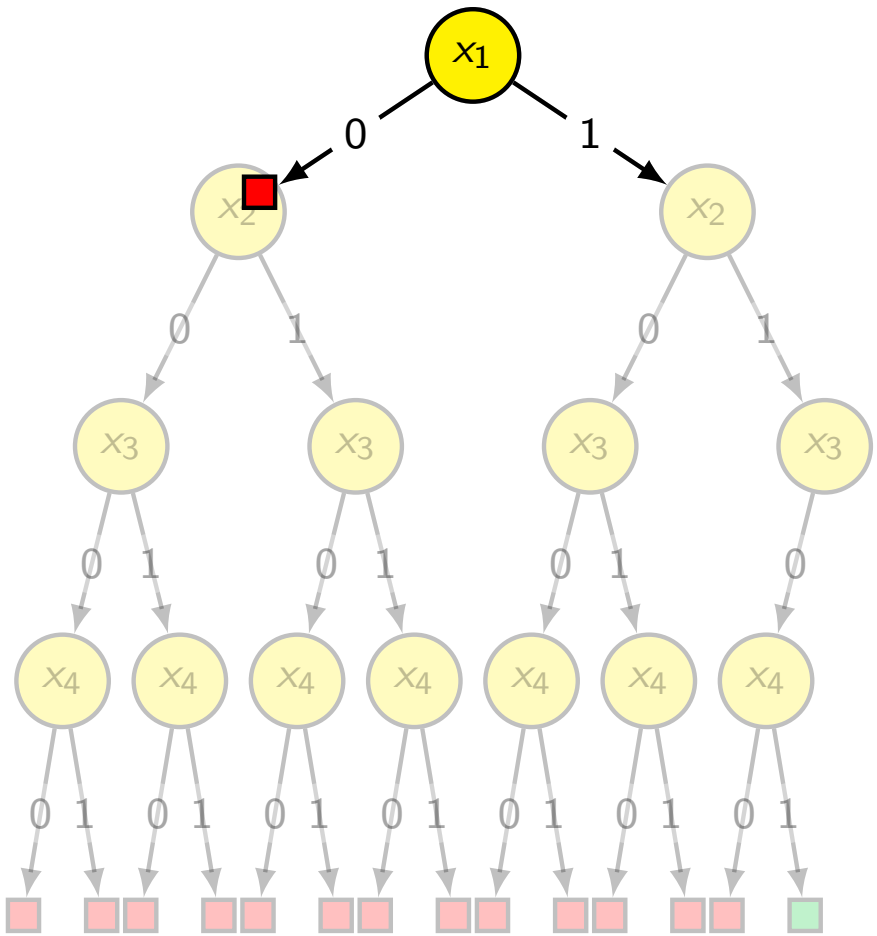
$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

learnt clauses

- $\omega_8 = \{x_1\}$

Unit Propagation & Clause Learning



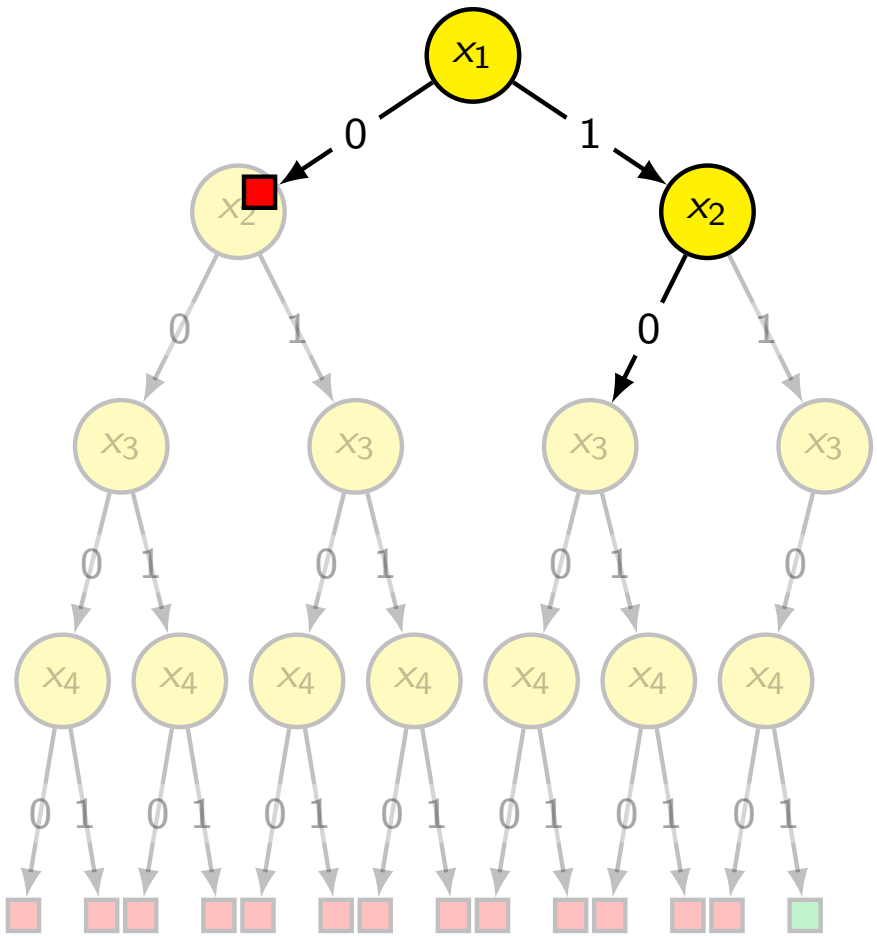
$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

learnt clauses

- $\omega_8 = \{x_1\}$

Unit Propagation & Clause Learning



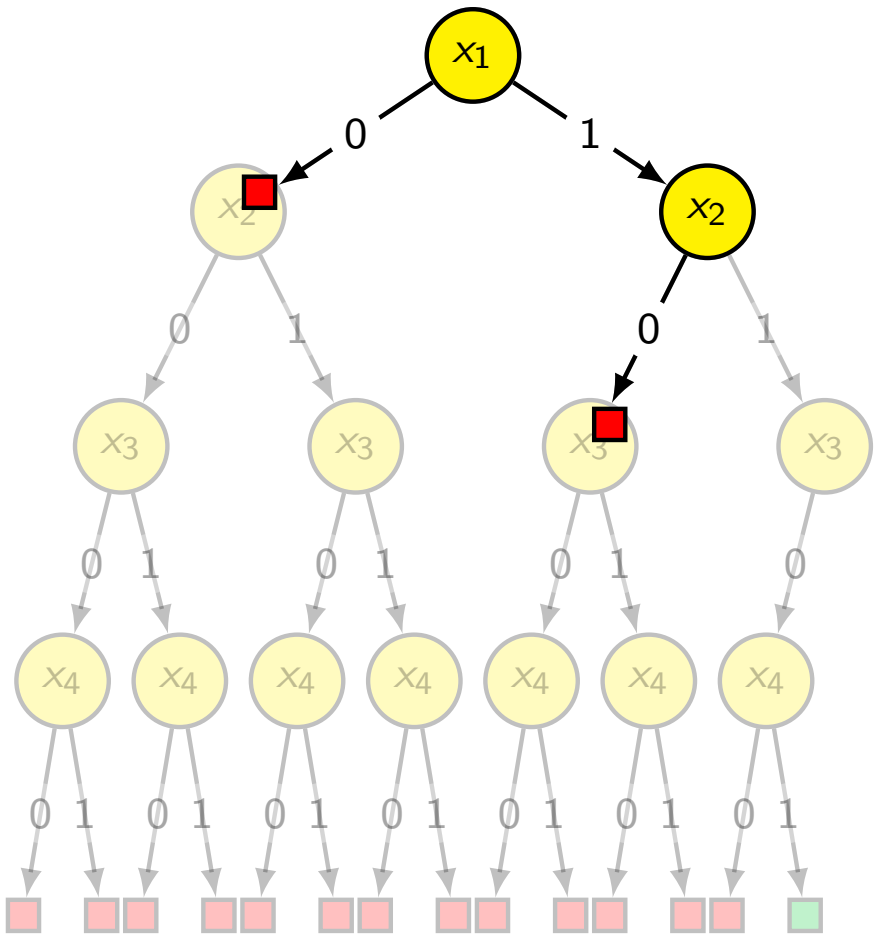
$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

learnt clauses

- $\omega_8 = \{x_1\}$

Unit Propagation & Clause Learning



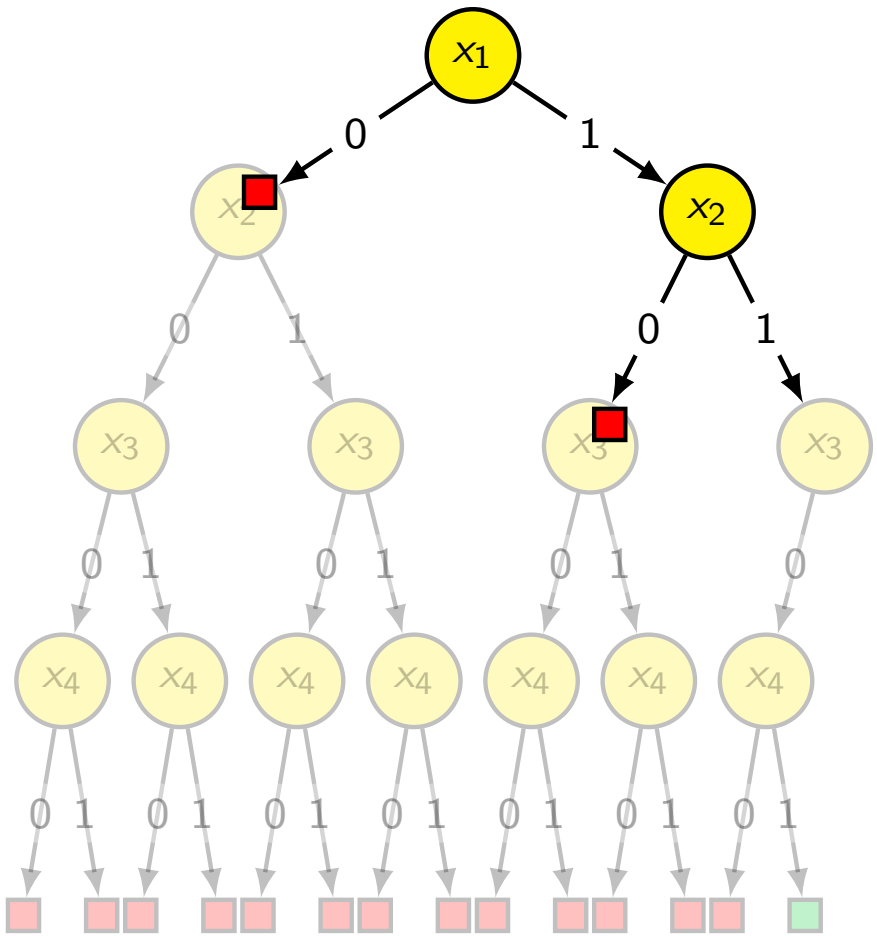
$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

learnt clauses

- $\omega_8 = \{x_1\}$
- $\omega_9 = \{x_2\}$

Unit Propagation & Clause Learning



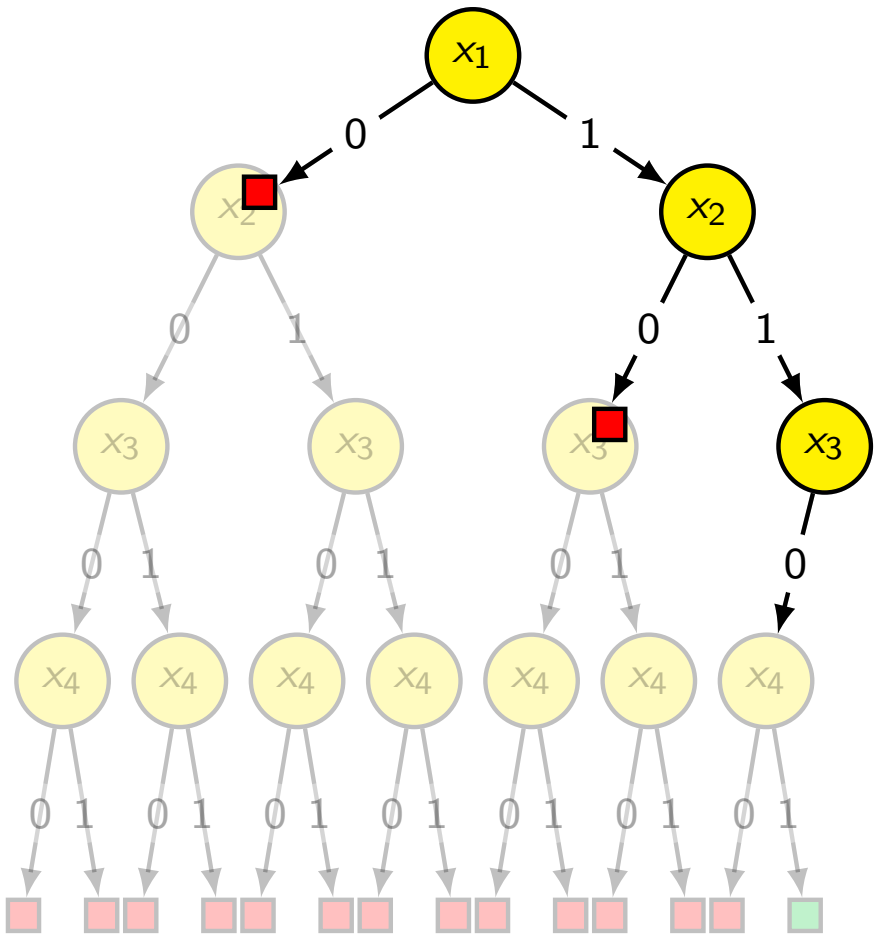
$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

learnt clauses

- $\omega_8 = \{x_1\}$
- $\omega_9 = \{x_2\}$

Unit Propagation & Clause Learning



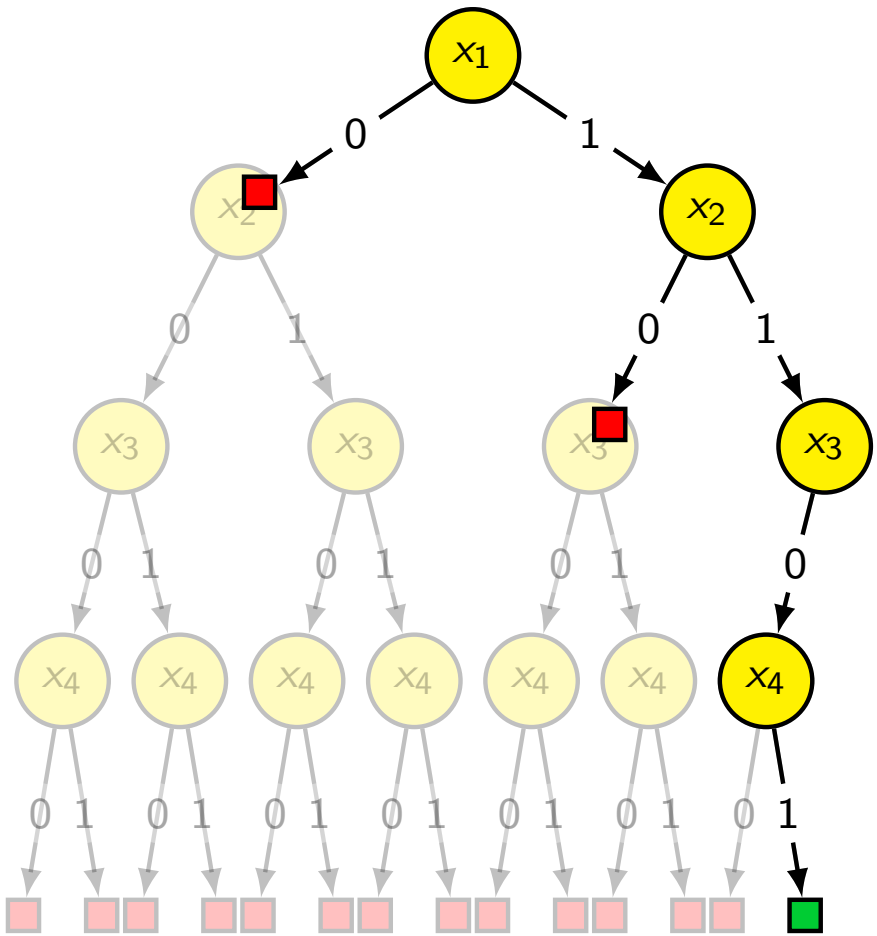
$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

learnt clauses

- $\omega_8 = \{x_1\}$
- $\omega_9 = \{x_2\}$

Unit Propagation & Clause Learning



$$\varphi = \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6$$

- $\omega_1 = \{x_1, x_2, x_3, x_4\}$
- $\omega_2 = \{x_1, \neg x_4\}$
- $\omega_3 = \{x_1, x_4\}$
- $\omega_4 = \{x_2, \neg x_4\}$
- $\omega_5 = \{x_2, x_4\}$
- $\omega_6 = \{x_3, x_4\}$

learnt clauses

- $\omega_8 = \{x_1\}$
- $\omega_9 = \{x_2\}$

Decision and Polarity

Decision

1. Occurrence in the initial problem
2. Occurrence in smaller clauses
3. **Occurrence in conflicts**

Decision and Polarity

Decision

1. Occurrence in the initial problem
2. Occurrence in smaller clauses
3. **Occurrence in conflicts**

- **VSIDS** [MMZ+01]:
  - Variables involved in conflict are rewarded
- **LRB** [LGPC16]:
  - Decision as a MAB problem
  - ERWA algorithm

Decision and Polarity

Decision

1. Occurrence in the initial problem
2. Occurrence in smaller clauses
3. **Occurrence in conflicts**

- **VSIDS** [MMZ+01]:
  - Variables involved in conflict are rewarded
- **LRB** [LGPC16]:
  - Decision as a MAB problem
  - ERWA algorithm

Polarity

1. False
2. Literals occurrence
3. **Phase (or progress) saving**

Decision and Polarity

Decision

1. Occurrence in the initial problem
2. Occurrence in smaller clauses
3. **Occurrence in conflicts**

- **VSIDS** [MMZ+01]:
  - Variables involved in conflict are rewarded
- **LRB** [LGPC16]:
  - Decision as a MAB problem
  - ERWA algorithm

Polarity

1. False
2. Literals occurrence
3. **Phase (or progress) saving**

- **Phase saving** [PD07]:
  - Each assignment is stored
  - The polarity used is the one stored
  - Functions as a cache

Parallel Resolution

Motivations

- Bigger and bigger instances
- No more faster CPU
- Multicore architectures

Parallel Resolution

Motivations

- Bigger and bigger instances
- No more faster CPU
- Multicore architectures

State of the art solutions

Portfolio [HJS09]	Divide-and-Conquer (DC) [ZBH96]
Parallel propagation [HW12]	Formula decomposition [HMSW11]

Parallel Resolution

Motivations

- Bigger and bigger instances
- No more faster CPU
- Multicore architectures

State of the art solutions

Portfolio [HJS09]	**Divide and conquer (DC)** [ZBH96]
Parallel propagation [HW12]	Formula decomposition [HMSW11]

Parallel Resolution

Motivations

- Bigger and bigger instances
- No more faster CPU
- Multicore architectures

SAT Competition 2016

Rank	Name	Type
1	treengeling	DC
2	plingeling	PF
3	cmsat5_run_parallel	PF
4	glucose-syrup	PF
5	cbpenelope	PF
6	csspenelope	PF
7	penelope_48	PF
8	Priss	PF
9	tbParaGlueminisat	PF
10	gluco_par	PF
11	ampharos-sat	DC

State of the art solutions

 **Portfolio** [HJS09]	**Divide and conquer (DC)** [ZBH96]
Parallel propagation [HW12]	Formula decomposition [HMSW11]

Portfolio & Divide and Conquer

worker: sequential solver + workflow

Portfolio & Divide and Conquer

worker: sequential solver + workflow

Portfolio

Multiple workers competing on the entire search space [HJS09]

- Diversification and intensification

Divide and Conquer (DC)

Divide the search space during the execution [ZBH96]

- Load balancing and work stealing

Portfolio & Divide and Conquer

worker: sequential solver + workflow

Portfolio

Multiple workers competing on the entire search space [HJS09]

- Diversification and intensification

Divide and Conquer (DC)

Divide the search space during the execution [ZBH96]

- Load balancing and work stealing

Criteria	**Portfolio**	**DC**
Learnt clauses exchange	possible	possible
Worker search spaces	same	different
Scalable	no*	yes
Easy to implement	yes	no

Portfolio & Divide and Conquer

worker: sequential solver + workflow

Portfolio

Multiple workers competing on the entire search space [HJS09]

- Diversification and intensification

Divide and Conquer (DC)

Divide the search space during the execution [ZBH96]

- Load balancing and work stealing

Criteria	**Portfolio**	**DC**
Learnt clauses exchange	possible	possible
Worker search spaces	same	different
Scalable	no*	yes
Easy to implement	yes	no

Possible contributions:

There also exist **hybrid approaches** (more complex).

Learnt Clauses Exchange

In practice not all the learnt clauses can be exchanged.

- Possible hardware congestion
- Too many memory used
- Slowdown the workers unit propagation

Learnt Clauses Exchange

In practice not all the learnt clauses can be exchanged.

- Possible hardware congestion
- Too many memory used
- Slowdown the workers unit propagation

Selecting clauses

- Sequential based measures (size, activity, LBD [AS09]).
- Adaptive thresholds for these measures [HJS11].

Selecting emitters

- Choose workers allowed to communicate [LHJS12].

Learnt Clauses Exchange

In practice not all the learnt clauses can be exchanged.

- Possible hardware congestion
- Too many memory used
- Slowdown the workers unit propagation

Selecting clauses

- Sequential based measures (size, activity, LBD [AS09]).
- Adaptive thresholds for these measures [HJS11].

Selecting emitters

- Choose workers allowed to communicate [LHJS12].

Possible contributions:

Which clauses should be exchanged? And between which workers?

Difficulties Faced by Contributions

Difficulties Faced by Contributions

Problem 1

- Concurrent programming $\rightarrow$ specific skills
- Implementation vs theoretical efficiency

Difficulties Faced by Contributions

Problem 1

- Concurrent programming $\rightarrow$ specific skills
- Implementation vs theoretical efficiency

Problem 2

- Contribution $\rightarrow$ one component
- Evaluation $\rightarrow$ complete solver

Difficulties Faced by Contributions

Problem 1

- Concurrent programming $\rightarrow$ specific skills
- Implementation vs theoretical efficiency

Problem 2

- Contribution $\rightarrow$ one component
- Evaluation $\rightarrow$ complete solver

Problem 3

- An implementation $\rightarrow$ single composition policy
- New heuristic $\times$ other mechanisms $\rightarrow$ difficult

Example of using GP-SAT

A **generic**, **modular**, and **efficient** framework to build parallel SAT solvers.

Example of using GP-SAT

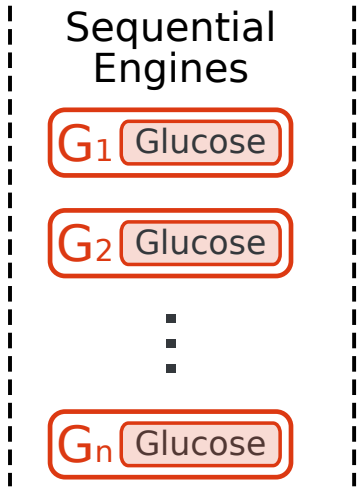
A **generic**, **modular**, and **efficient** framework to build parallel SAT solvers.

- `glucose-syrup` [AS14]

Example of using GP-SAT

A **generic**, **modular**, and **efficient** framework to build parallel SAT solvers.

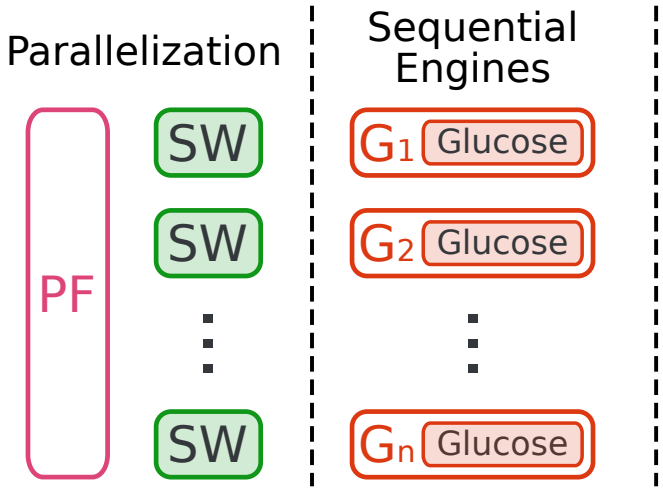
- `glucose-syrup` [AS14]
  - `glucose` [AS09]



Example of using GP-SAT

A **generic**, **modular**, and **efficient** framework to build parallel SAT solvers.

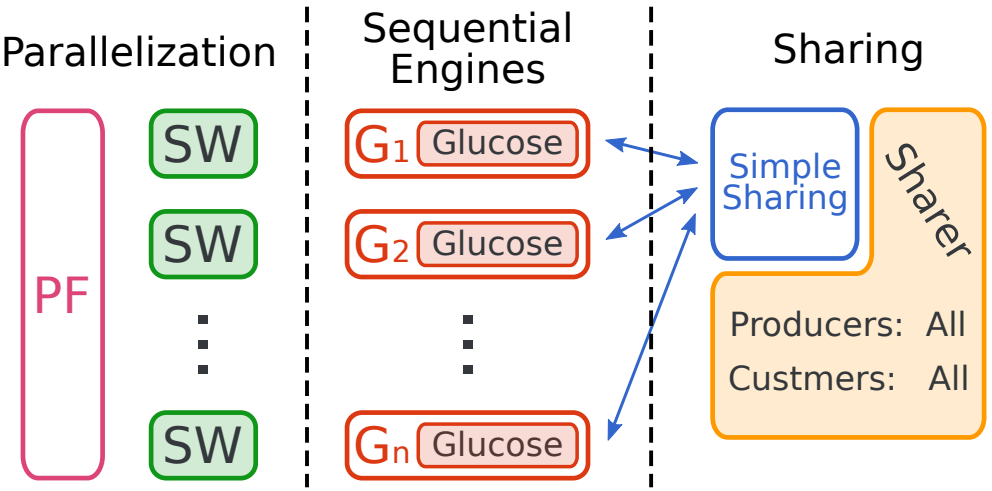
- `glucose-syrup` [AS14]
  - `glucose` [AS09]
  - Portfolio



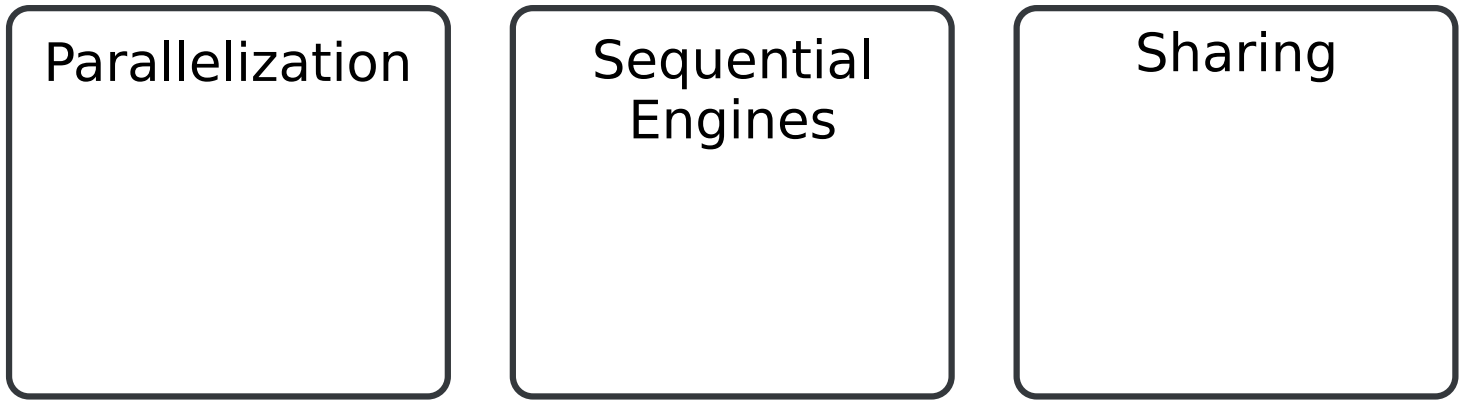
Example of using GP-SAT

A **generic**, **modular**, and **efficient** framework to build parallel SAT solvers.

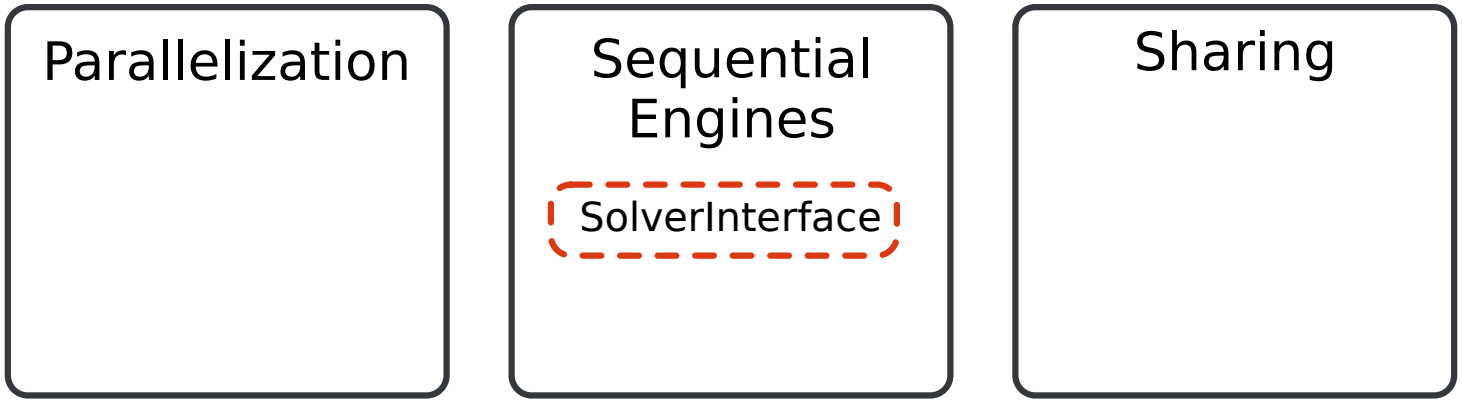
- `glucose-syrup` [AS14]
  - `glucose` [AS09]
  - Portfolio
  - All to all



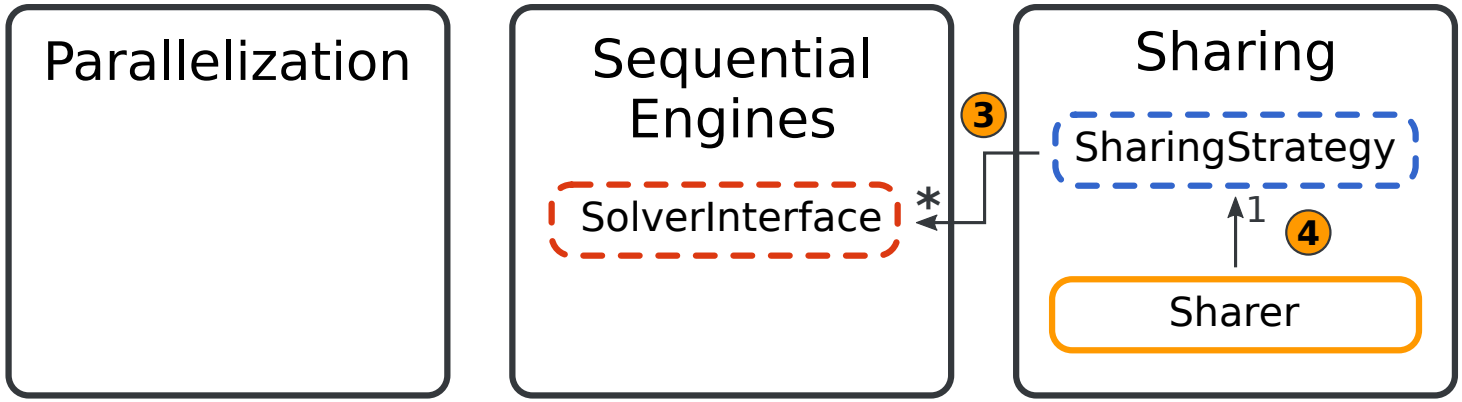
Architecture of GP-SAT



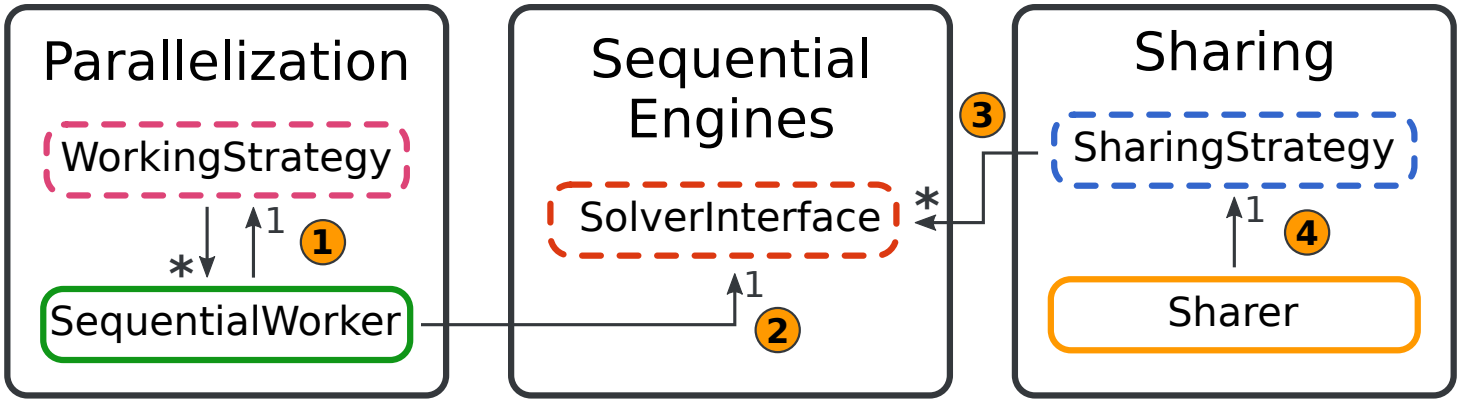
Architecture of GP-SAT



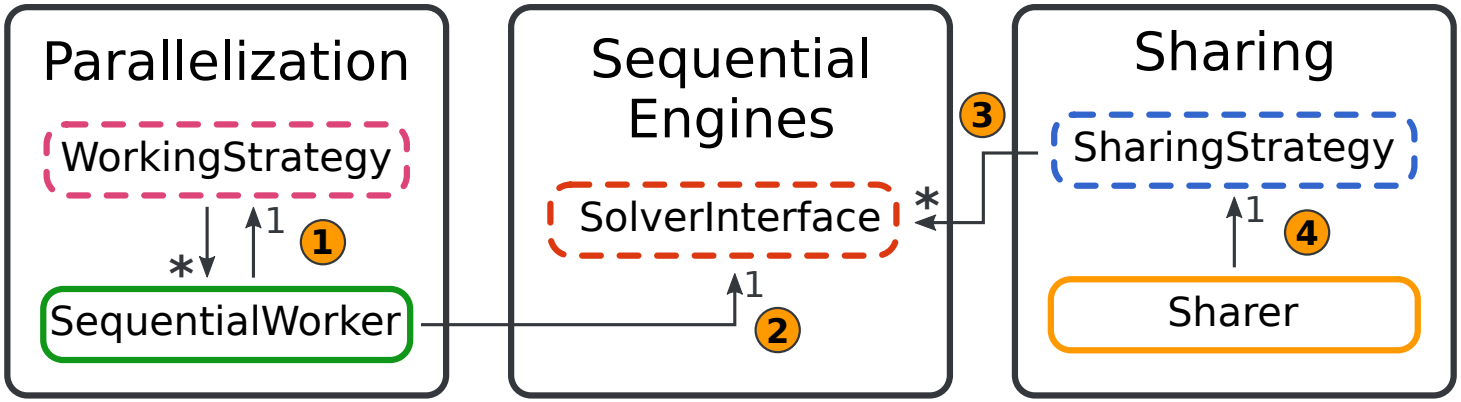
Architecture of GP-SAT

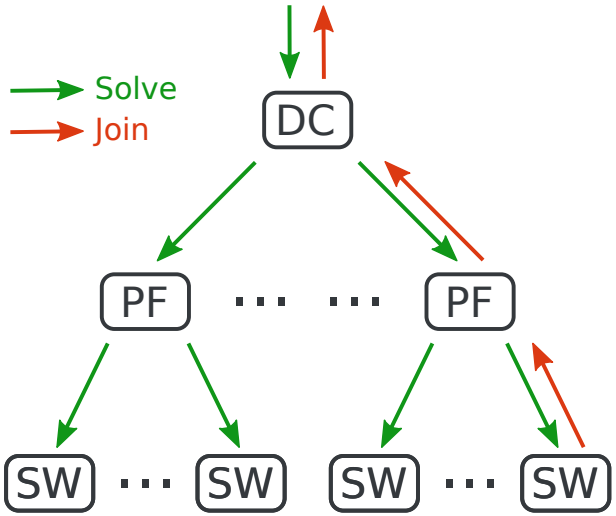


Architecture of GP-SAT



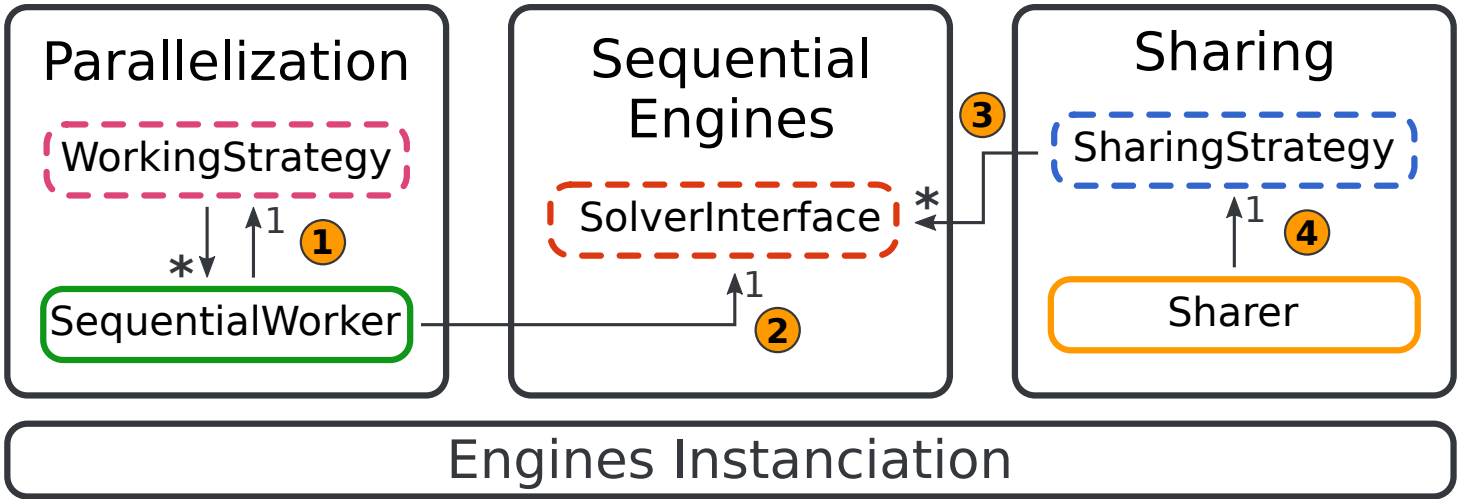
Architecture of GP-SAT

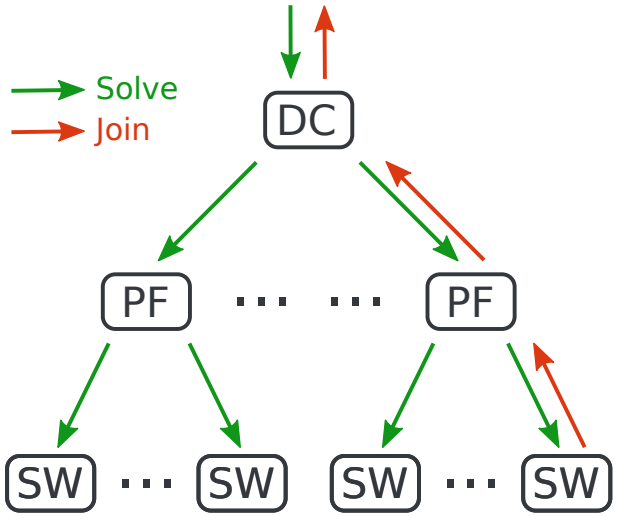




Exemple of a Divide-and-Conquer of Portfolio.

Architecture of GP-SAT





Exemple of a Divide-and-Conquer of Portfolio.

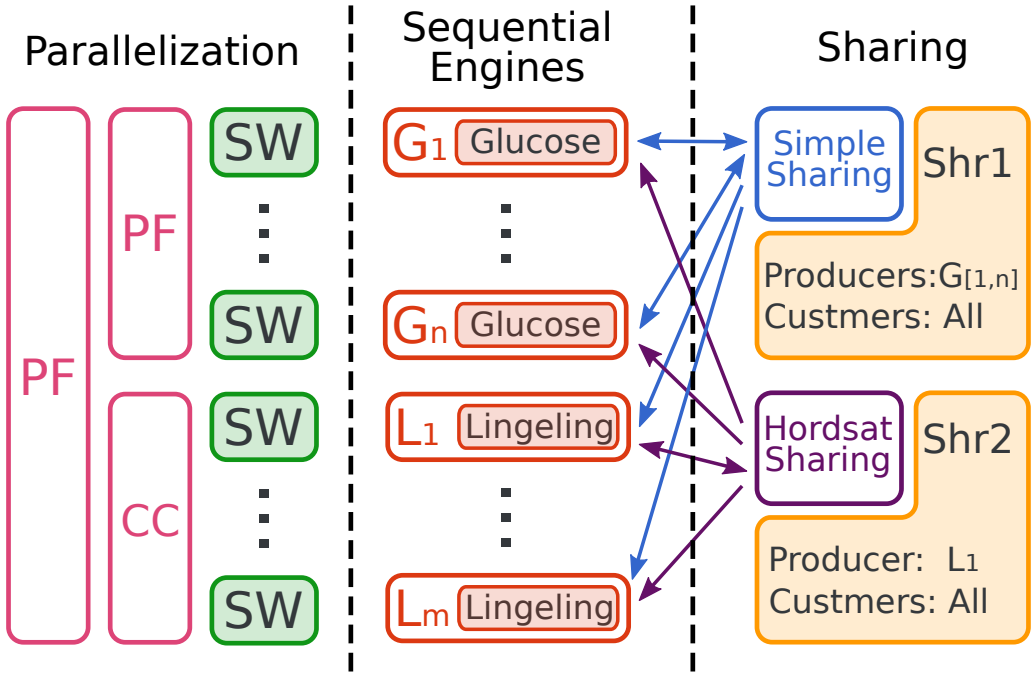
Others Implentations using GP-SAT

- `Hordesat` [BSS15]: `Lingeling` [Bie16] + Portfolio + special sharing.
- `Treengeling` [Bie16]: `Lingeling` + Cube-and-Conquer + simple sharing of units.

Others Implentations using GP-SAT

- `Hordesat` [BSS15]: `Lingeling` [Bie16] + Portfolio + special sharing.
- `Treengeling` [Bie16]: `Lingeling` + Cube-and-Conquer + simple sharing of units.

Solver mixing all the strategies.



Experiments & Results

Parameters

40 **processors:**	Intel Xeon E7-2860
memory:	500 Go
#cores:	36
#instances:	100 (SAT Race 2015)
#run(s):	1
timeout:	5000s

Experiments & Results

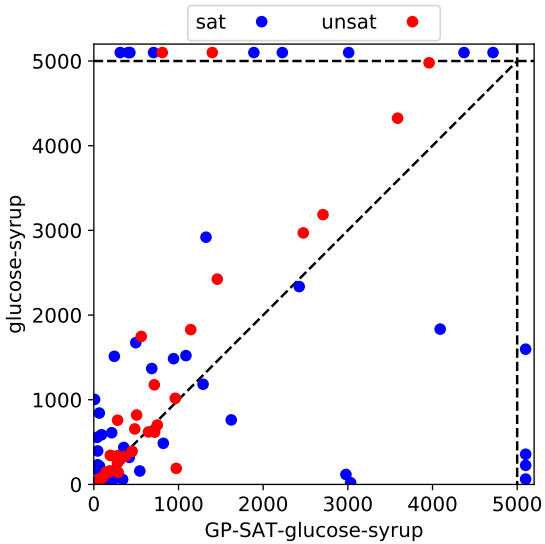
Parameters

40 **processors:** Intel Xeon E7-2860
memory: 500 Go

#cores: 36
#instances: 100 (SAT Race 2015)
#run(s): 1
timeout: 5000s

#solved instances

Solver	All	unsat	sat
`glucose-syrup`	71	30	41
`GP-SAT-glucose`	**78**	**32**	**46**



Dots over the line are good ones.

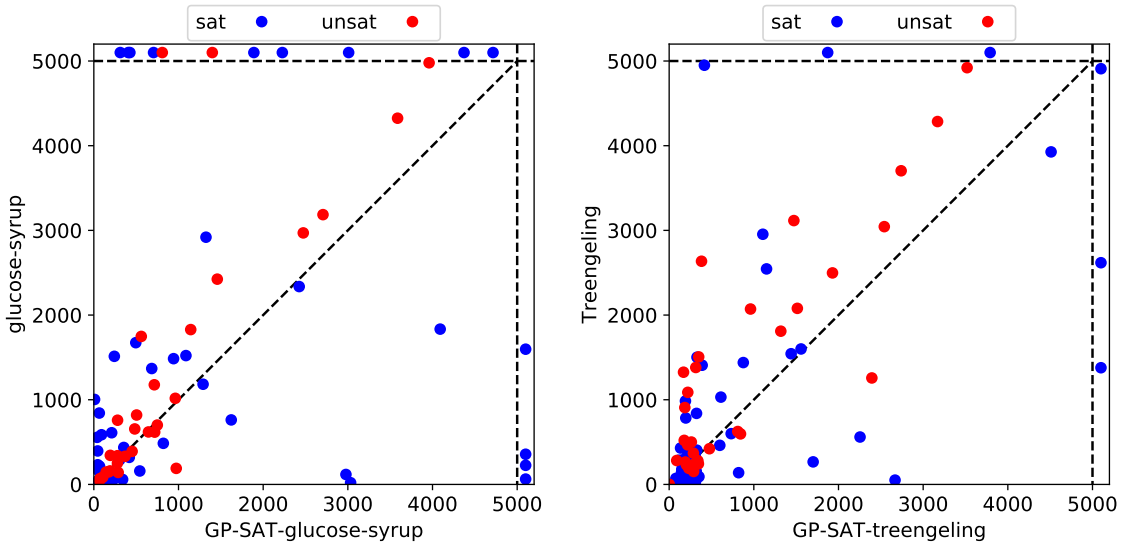
Experiments & Results

Parameters

40 **processors:**	Intel Xeon E7-2860
memory:	500 Go
#cores:	36
#instances:	100 (SAT Race 2015)
#run(s):	1
timeout:	5000s

#solved instances

Solver	All	unsat	sat
`glucose-syrup`	71	30	41
`GP-SAT-glucose`	**78**	**32**	**46**
`Treengeling`	**82**	**32**	**50**
`GP-SAT-treengeling`	81	**32**	49



Dots over the line are good ones.

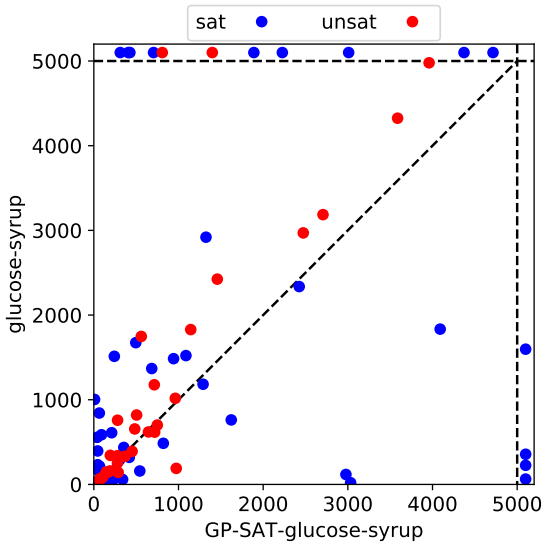
Experiments & Results

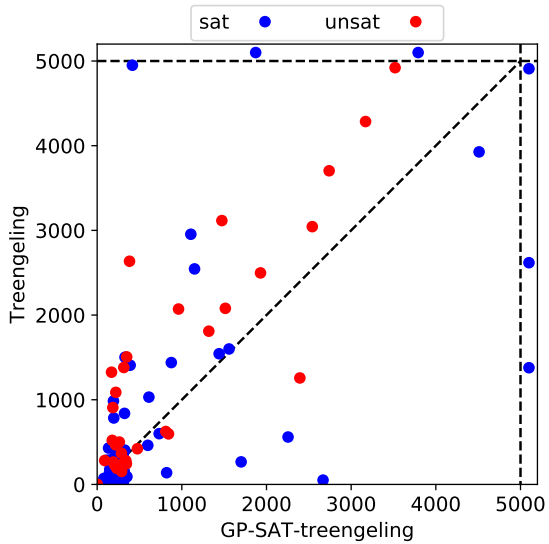
Parameters

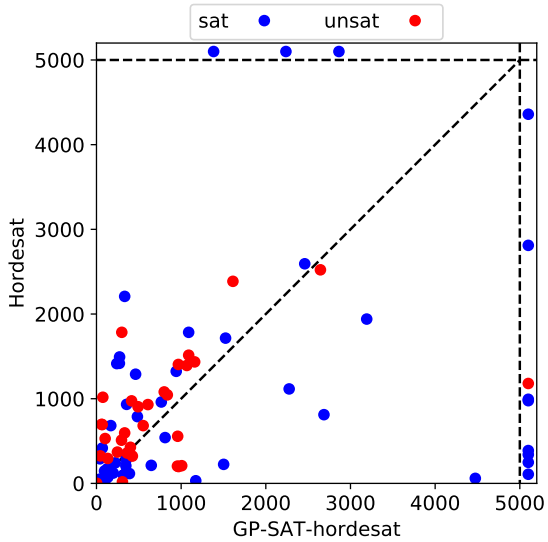
40 **processors:**	Intel Xeon E7-2860
memory:	500 Go
#cores:	36
#instances:	100 (SAT Race 2015)
#run(s):	1
timeout:	5000s

#solved instances

Solver	All	unsat	sat
glucose-syrup	71	30	41
GP-SAT-glucose	**78**	**32**	**46**
Treengeling	**82**	**32**	**50**
GP-SAT-treengeling	81	**32**	49
Hordesat	**80**	**32**	**48**
GP-SAT-hordesat	74	31	43







Dots over the line are good ones.

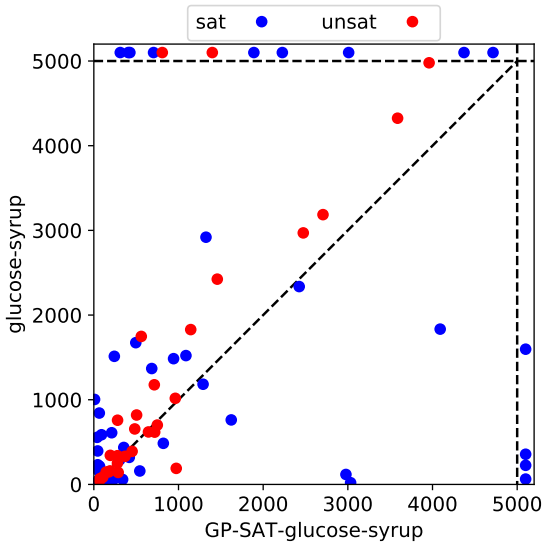
Experiments & Results

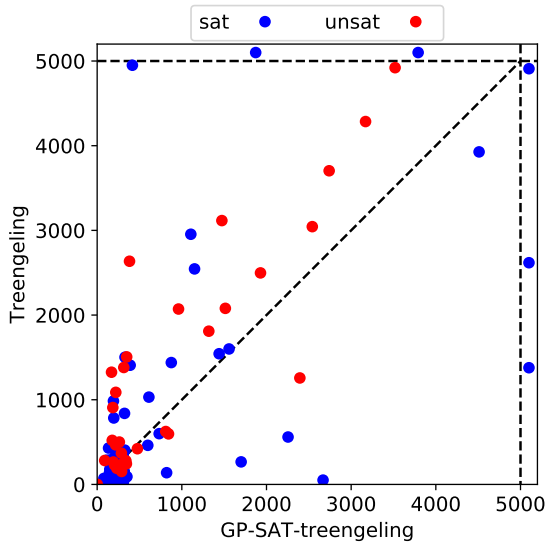
Parameters

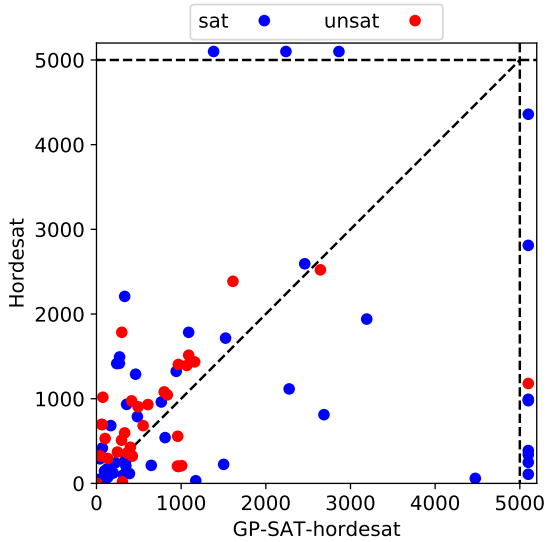
40 **processors:**	Intel Xeon E7-2860
memory:	500 Go
#cores:	36
#instances:	100 (SAT Race 2015)
#run(s):	1
timeout:	5000s

#solved instances

Solver	All	unsat	sat
glucose-syrup	71	30	41
GP-SAT-glucose	**78**	**32**	**46**
Treengeling	**82**	**32**	**50**
GP-SAT-treengeling	81	**32**	49
Hordesat	**80**	**32**	**48**
GP-SAT-hordesat	74	31	43







Dots over the line are good ones.

Conclusion & Perspectives

- GP-SAT is **modular** and **generic**.
- Experiments shows that GP-SAT is **efficient**.
- **The proof of concept is validated**: good results for a prototype.

Conclusion & Perspectives

- GP-SAT is **modular** and **generic**.
- Experiments shows that GP-SAT is **efficient**.
- **The proof of concept is validated**: good results for a prototype.

- Bench with more instances (SAT Competition 2016).
- Toward a more generic architecture.
- **Implementing more strategies to test the architecture.**
- **Participating to the SAT Competition 2017.**
- **Use GP-SAT for our own contributions.**

Thank you!


Gilles Audemard and Laurent Simon.
Predicting learnt clauses quality in modern sat solvers.
In *IJCAI*, volume 9, pages 399–404, 2009.


Gilles Audemard and Laurent Simon.
Lazy clause exchange policy for parallel sat solvers.
In *int. conf. on Theory and Applications of Satisfiability Testing*, pages 197–205. Springer, 2014.


Armin Biere, Marijn Heule, and Hans van Maaren.
Handbook of satisfiability, volume 185.
IOS press, 2009.


Armin Biere.
Splatz, lingeling, plingeling, treengeling, yalsat entering the sat competition 2016.
SAT COMPETITION 2016, page 44, 2016.


Tomáš Balyo, Peter Sanders, and Carsten Sinz.
Hordesat: A massively parallel portfolio sat solver.
In *int. conf. on Theory and Applications of Satisfiability Testing*, pages 156–172. Springer, 2015.


Stephen A Cook.
The complexity of theorem-proving procedures.
In *3rd annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.


Martin Davis, George Logemann, and Donald Loveland.
A machine program for theorem-proving.
Commun. ACM, 5(7):394–397, July 1962.


Martin Davis and Hilary Putnam.
A computing procedure for quantification theory.
J. ACM, 7(3):201–215, July 1960.


Youssef Hamadi, Said Jabbour, and Lakhdar Sais.
Manysat: a parallel sat solver.
Journal on Satisfiability, Boolean Modeling and Computation, 6:245–262, 2009.


Youssef Hamadi, Said Jabbour, and Jabbour Sais.
Control-based clause sharing in parallel sat solving.
In *Autonomous Search*, pages 245–267. Springer, 2011.


Youssef Hamadi, Joao Marques-Silva, and Christoph M Wintersteiger.
Lazy decomposition for distributed decision procedures.
arXiv preprint arXiv:1111.0371, 2011.


Antti EJ Hyvärinen and Christoph M Wintersteiger.
Approaches for multi-core propagation in clause learning satisfiability solvers.
Technical report, Technical Report MSR-TR-2012-47, 2012.


Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki.
Learning rate based branching heuristic for sat solvers.
In *Theory and Applications of Satisfiability Testing*, pages 123–140. Springer, 2016.


Nadjib Lazaar, Youssef Hamadi, Said Jabbour, and Michèle Sebag.
Cooperation control in Parallel SAT Solving: a Multi-armed Bandit Approach.
Research Report RR-8070, INRIA, September 2012.


Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
Chaff: Engineering an efficient sat solver.
In *38th annual Design Automation Conference*, pages 530–535. ACM, 2001.


Joao P Marques-Silva, Karem Sakallah, et al.
Grasp: A search algorithm for propositional satisfiability.
IEEE Trans. on Computers, 48(5):506–521, 1999.


Knot Pipatsrisawat and Adnan Darwiche.
A lightweight component caching scheme for satisfiability solvers.
In *Theory and Applications of Satisfiability Testing–SAT 2007*, pages 294–299. Springer, 2007.


Hantao Zhang, Maria Paola Bonacina, and Jieh Hsiang.
PSATO: a distributed propositional prover and its application to quasigroup problems.
Journal of Symbolic Computation, 21(4):543–560, 1996.


Lintao Zhang, Conor F Madigan, Matthew H Moskewicz, and Sharad Malik.
Efficient conflict driven learning in a boolean satisfiability solver.
In *IEEE/ACM int. conf. on Computer-aided design*, pages 279–285. IEEE Press, 2001.

Clause Learning

Let $x_1 = 0$, $x_4 = 0$, $x_8 = 1$ be the reasons of a conflict.

$$\varphi \equiv \varphi \wedge \omega_{\text{learnt}}, \text{ where } \omega_{\text{learnt}} = \{x_1, x_4, \neg x_8\}$$

Problem

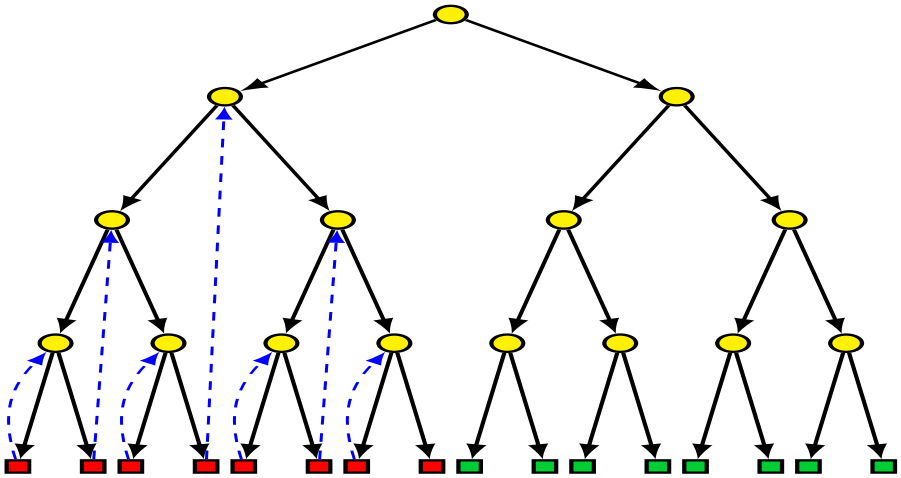
Too many clauses are learnt
$\rightarrow$ not all of them can be kept.

Selection criteria

- Activity: keep clauses used for propagation/conflict resolution.
- Size: longer clauses are less used for propagation.
- **Literal Block Distance (LBD)** [AS09]:
  - number of decision level gathered by a clause.

Restart

Motivations



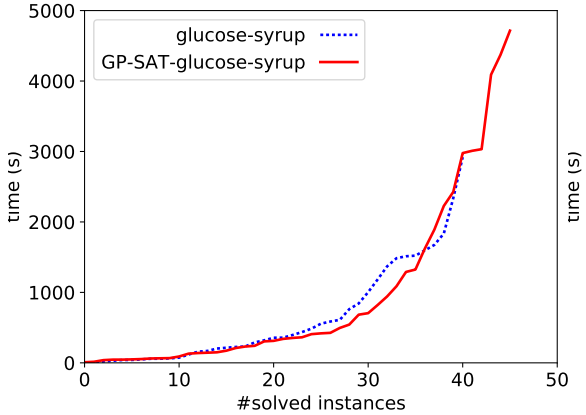
Escape from hard zones (heavy-tailed) [BHvM09].

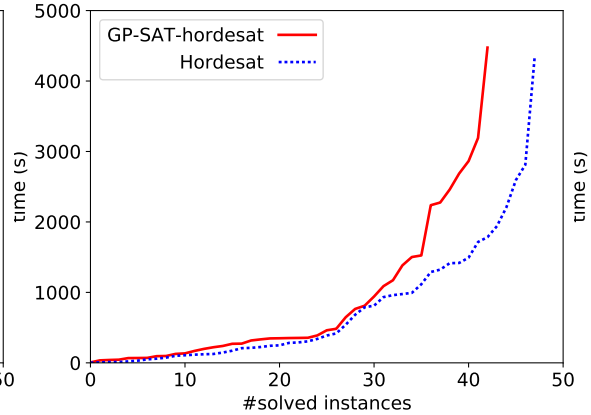
Strategies

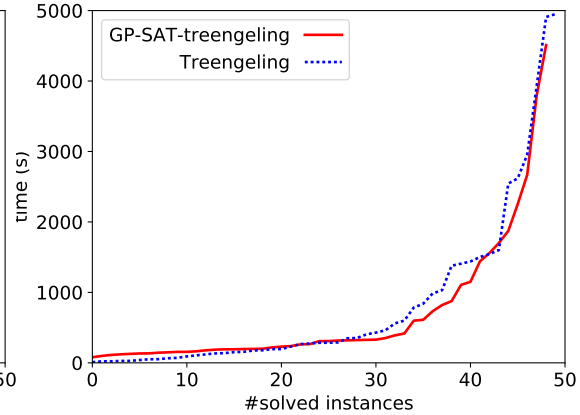
- #conflicts: arithmetic, geometric, Luby progressions.
- Progress saving: prevents from softness.
- **Average LBD**: are we in a good region for learning?
- **SAT/UNSAT**: are we close to find a solution?

Cactus Plots

sat







unsat