SAT Solvers: A Computer Science Perspective

By Pouria Moradpour
With Special Thanks To
Dr. Malihe Yousofzadeh & Dr. Jaafar Almasizadeh

A Linear Solver

01

A Vault!

An Analogy for Boolean Satisfiablity

02

Transformation

Defining T and Seeing it in Practice

03

DAG

Creating a Directed, Acyclic Graph (DAG)

SAT Solver Algorithms

04

Using DFS

A Simple Brute Force Algorithm

05

Techniques

General Techniques Used in SAT Solver Algorithms

06

DPLL

to Solve B-SAT Problems more Cleverly

A Linear Solver

01

A Vault!

SAT as a Giant Logic Puzzle: The Lock and Key Analogy

02

Transformation

Transforming General Formulas

Formula Transformation

- We work in a minimal logic fragment:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi$$

- Translate other connectives using:

$$\begin{aligned}
T(p) &= p \\
T(\phi_1 \wedge \phi_2) &= T(\phi_1) \wedge T(\phi_2) \\
T(\phi_1 \to \phi_2) &= \neg(T(\phi_1) \wedge \neg T(\phi_2)) \\
T(\neg\phi) &= \neg T(\phi) \\
T(\phi_1 \vee \phi_2) &= \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))
\end{aligned}$$

- Semantically equivalent to original formula

An Example

$$\varphi = p \land \neg(q \lor \neg p)$$

An Example

φ = p ∧ ¬(q ∨ ¬p)

T(φ)?

$$T(p) = p$$

$$T(\phi_1 \wedge \phi_2) = T(\phi_1) \wedge T(\phi_2)$$

$$T(\phi_1 \rightarrow \phi_2) = \neg(T(\phi_1) \wedge \neg T(\phi_2))$$

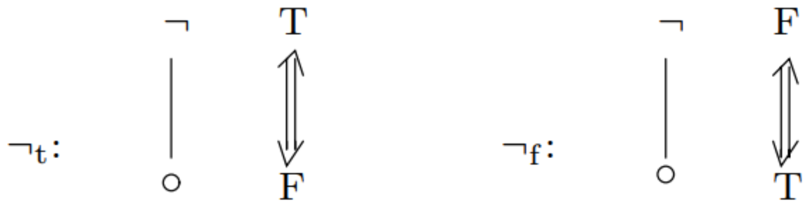
$$T(\neg\phi) = \neg T(\phi)$$

$$T(\phi_1 \vee \phi_2) = \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))$$

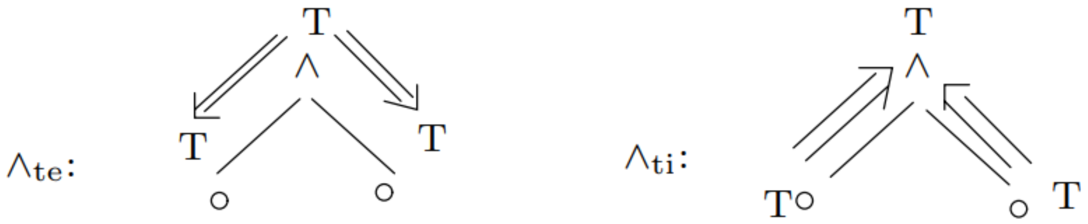
03

Constructing a DAG

Moving From Parse Trees to DAGs, Representing a Witness to Satisfiability

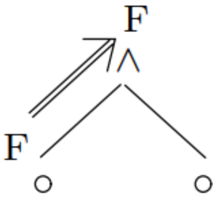


forcing laws for negation

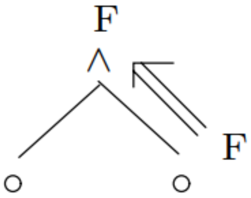


true conjunction forces true conjuncts   true conjunctions force true conjunction

$\wedge_{\mathrm{fl}}$:

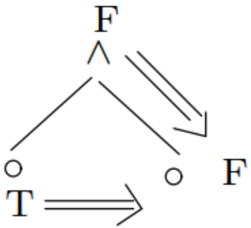


$\wedge_{\mathrm{fr}}$:

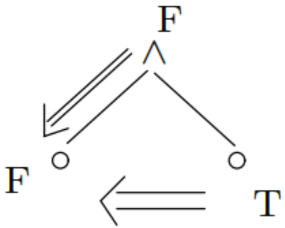


false conjuncts
force false conjunction

$\wedge_{\mathrm{fll}}$:



$\wedge_{\mathrm{frr}}$:



false conjunction and true conjunct
force false conjunction

An Example

$$T(\varphi) = p \land \neg\neg(\neg q \land \neg\neg p)$$

Witness to Satisfiability

- All nodes marked consistently
- Bottom-up re-check confirms consistency

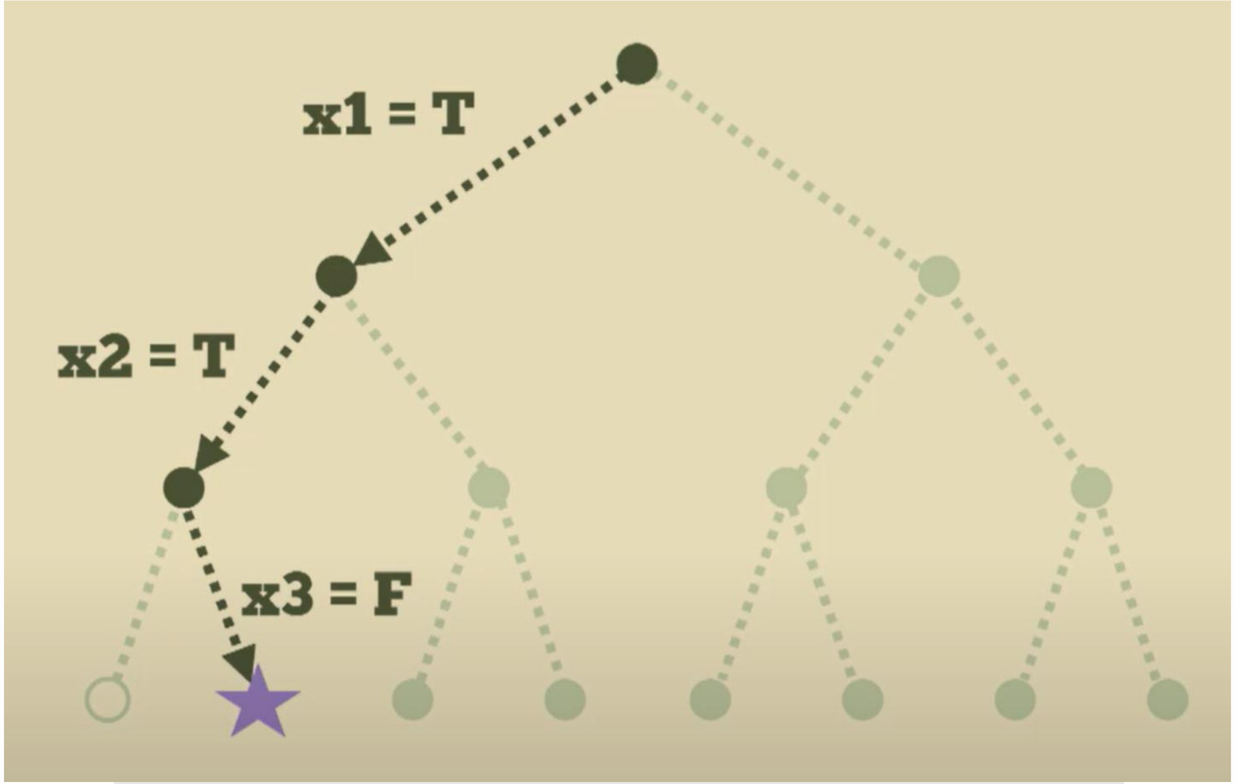
- Limitations of Our Linear Solver

SAT Solver Algorithms

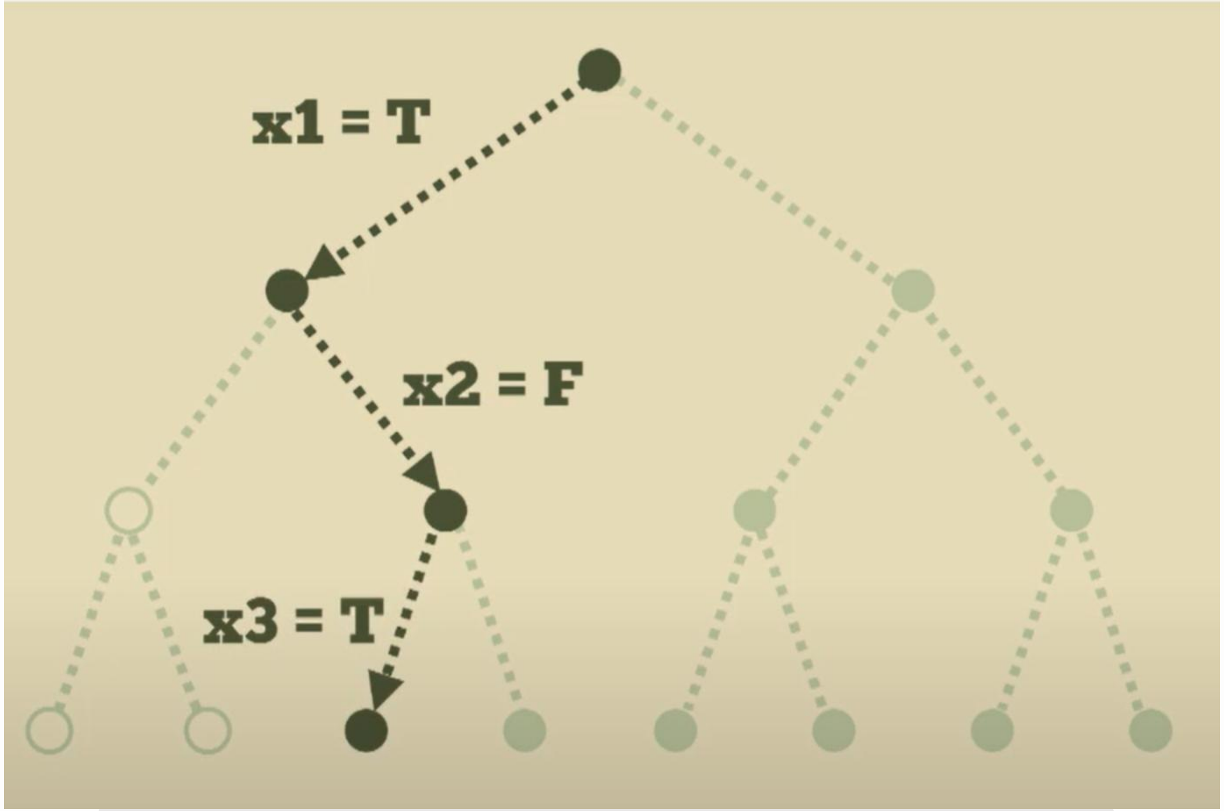
04

Using DFS

A Simple Brute-Force Algorithm Just to Gain an Intuition



CR: ClojureTV



CR: ClojureTV

05

Techniques

To Come up With Faster Solutions

Genral Techniques

Pre-processing

cleaning and simplifying upfront

Backtracking

systematic search through the solution space

Unit Propagation

chaining logical consequences

06

DPLL

A Classic Solver, and a Demonstration of the Techniques

Example For Unit Propagation & Backtracking

x7 = false
x8 = false

x9 = true
x10 = true

x1 = false

x2 = true, x3 = true, x4 = true

x5 = true, x6 = false, x6 = true

```
(and (or x1 x2)
     (or x1 x3 x8)
     (or (not x2) (not x3) x4)
     (or (not x4) x5 x7)
     (or (not x4) x6 x8)
     (or (not x5) (not x6))
     (or x7 (not x8))
     (or x7 (not x9) x10))
```

CR: ClojureTV

Example For Unit Propagation & Backtracking

x7 = false
x8 = false

x9 = true
x10 = true

x1 = true

x2 = true, x3 = false, x4 = false

x5 = true, x6 = false

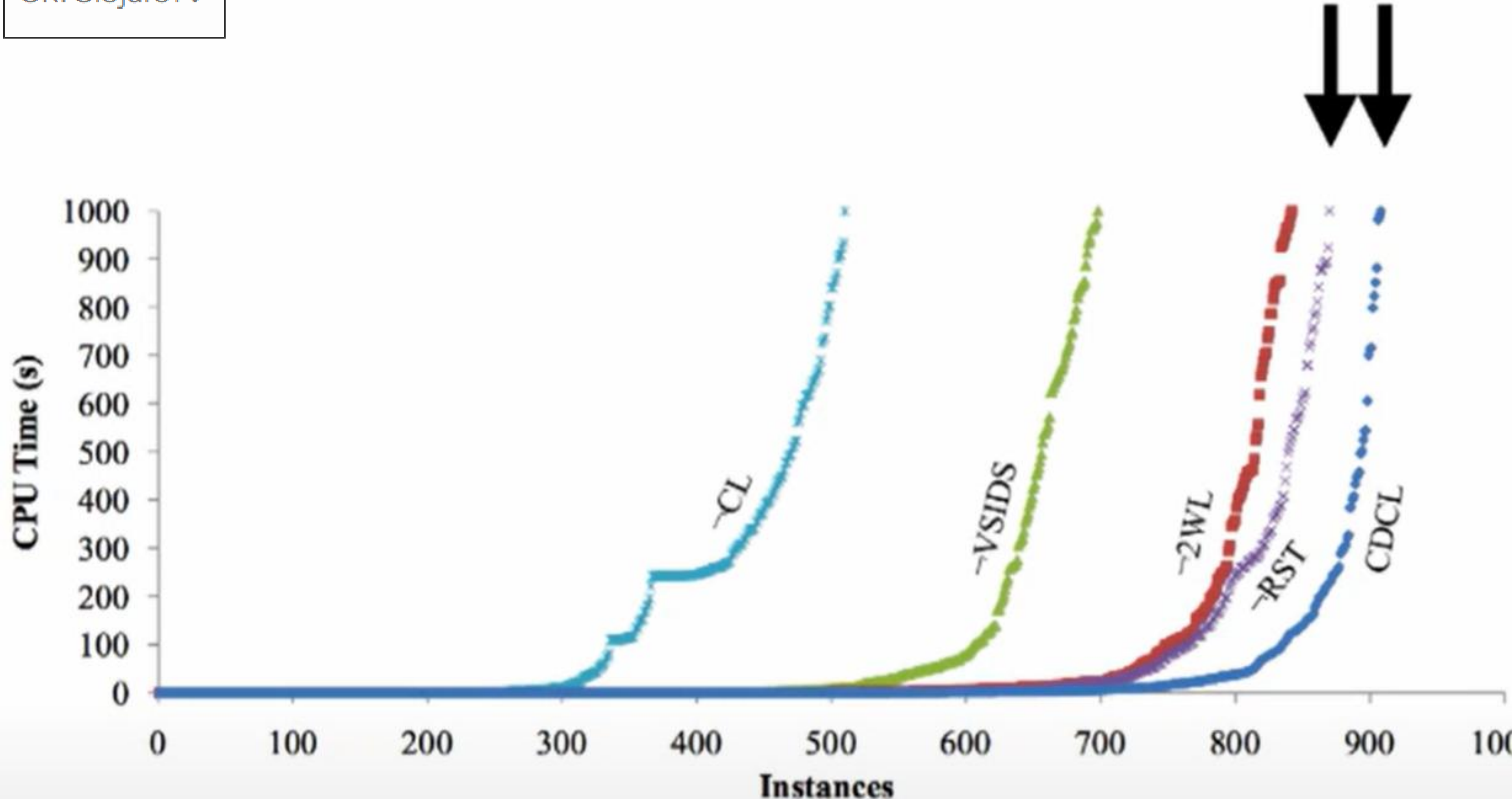
```
(and (or x1 x2)
     (or x1 x3 x8)
     (or (not x2) (not x3) x4
     (or (not x4) x5 x7)
     (or (not x4) x6 x8)
     (or (not x5) (not x6))
     (or x7 (not x8))
     (or x7 (not x9) x10))
```

CR: ClojureTV

An Example for PLE in DPLL

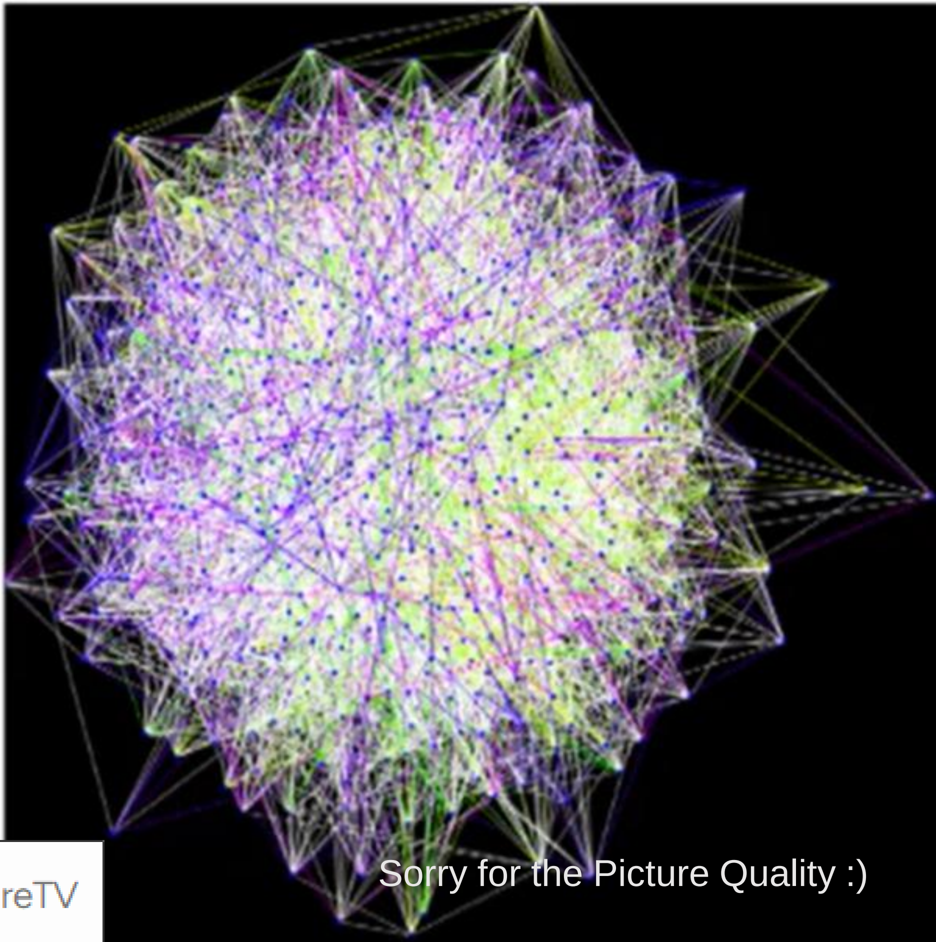
$$\varphi = (x \vee y) \wedge (x \vee z) \wedge (w \vee z) \wedge (\neg w \vee y)$$

CR: ClojureTV



Industrial

Random



CR: ClojureTV

Sorry for the Picture Quality :)

Thanks!

Find these slides, the ClojureTV Lecture, etc. Here:

CREDITS: This presentation template was created by **Slidesgo,** and includes icons, infographics & images by **Freepik**