The Application of Propositional Logic

Zhaoguo Wang

SAT-Based ATPG. In: Test Pattern Generation using Boolean Proof Engines. Springer, Dordrecht.

Propositional Logic + SAT

1.1 Propositional Logic

Step 0. Proof.

Step 1. Convert program into mathematical formula.

Step 2. Ask the computer to solve the formula.

1.2 First Order Logic

Step 1. Convert it into first order logic formula.

Step 2. Ask the computer to solve the formula.

1.3 Auto-active Proof

Step 1. Axiom system

Step 2. Ask the computer to check the invariants



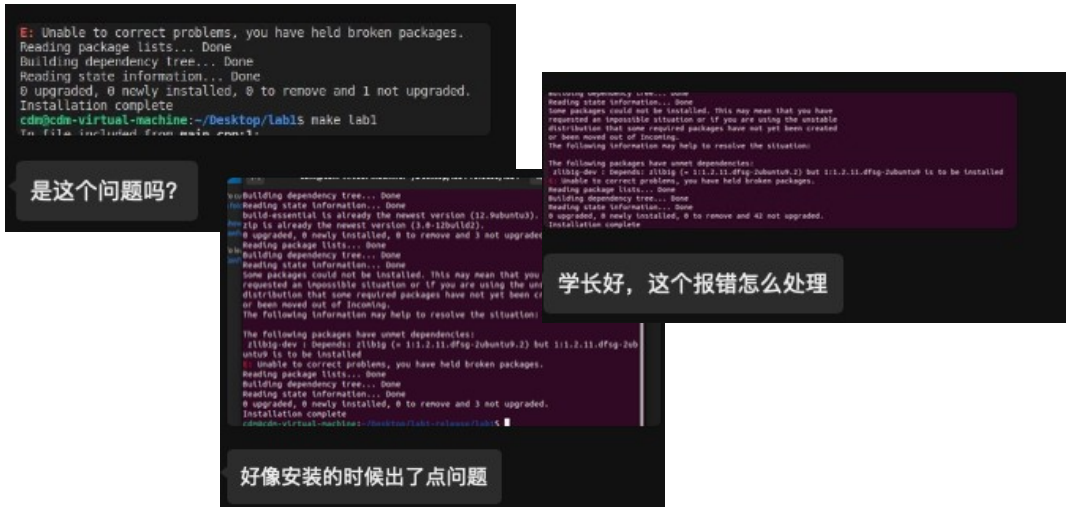
Outline – This Lecture

- Package Managers
- Electronic Design Automation (EDA)

Outline – This Lecture

- Package Managers
- Electronic Design Automation (EDA)

Lab1: A Frequently Asked Question



Unsatisfiable

```
The following packages have unmet dependencies:
 zlib1g-dev : Depends: zlib1g (= 1:1.2.11.dfsg-2ubuntu9.2) but 1:1.2.11.dfsg-2ubuntu9 is to be install
ed
E: Unable to correct problems, you have held broken packages.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
Installation complete
```

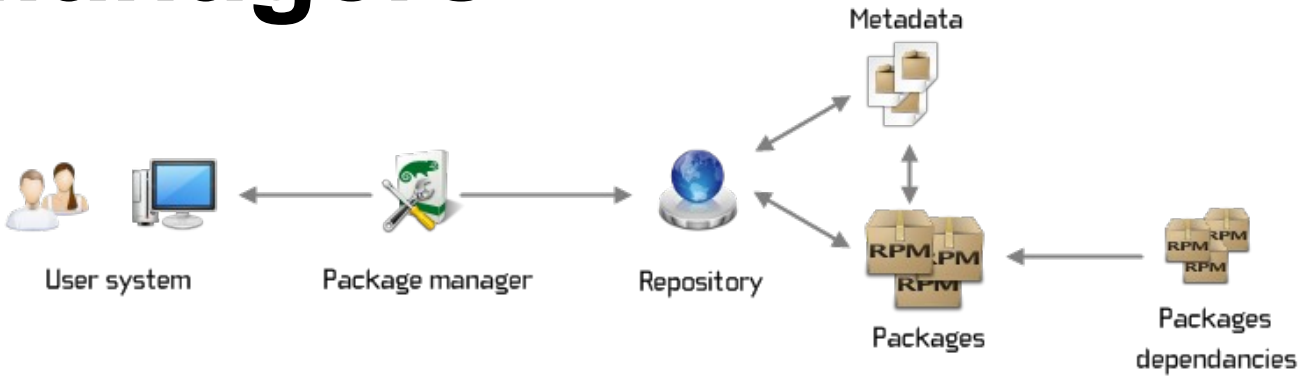
An Example

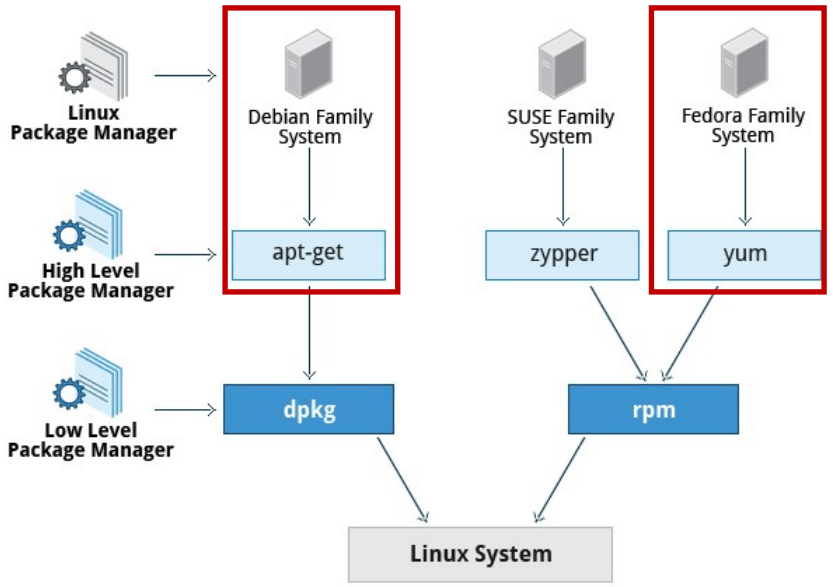
```
Package: firefox-3.0
Priority: optional
Section: web
Installed-Size: 3456
Maintainer: Alexander Sack <asac@ubuntu.com
Architecture: i386
Version: 3.0.16+nobinonly-0ubuntu0.9.04.1
Replaces: firefox (<< 3), firefox-granparadiso, firefox-libt                    trunk
Provides: firefox-libthai, www-browser
Depends: fontconfig, psmisc, debianutils (>= 1.16), xulrunner-1.9 (>= 1.9.0.1),
     libatk1.0-0 (>= 1.20.0), libc6 (>= 2.4), libcairo2 (>= 1.2.4),
     libfontconfig1 (>= 2.4.0), libfreetype6 (>= 2.2.1), libgcc1 (>= 1:4.1.1),
     libglib2.0-0 (>= 2.16.0), libgtk2.0-0 (>= 2.16.0),
     libnspr4-0d (>= 4.7.3-0ubuntu1~), libpango1.0-0 (>= 1.14.0),
     libstdc++6 (>= 4.1.1),
     firefox-3.0-branding (>= 3.0.3+nobinonly-0ubuntu1~) |
          abrowser-3.0-branding (>= 3.0.3+nobinonly-0ubuntu1~)
Suggests: ubufox,
     firefox-3.0-gnome-support (= 3.0.16+nobinonly-0ubuntu0.9.04.1),
     latex-xft-fonts, libthai0
Conflicts: firefox (<< 3), firefox-granparadiso (<< 3.0~alpha8-0),
     firefox-libthai, firefox-trunk (<< 3.0~a8~cvs20070914t1713-0)
Size: 887822
Description: safe and easy web browser from Mozilla
 Firefox delivers safe, easy web browsing. A familiar user interface,
 enhanced security features including protection from online identity
 and integrated search let you get the most out of the web.
 .
 Install this firefox package too, if you want to be automatically up
 new major firefox versions in the future.
```

Installing a browser requires to first install some other software in specific versions

There should be no software in these versions

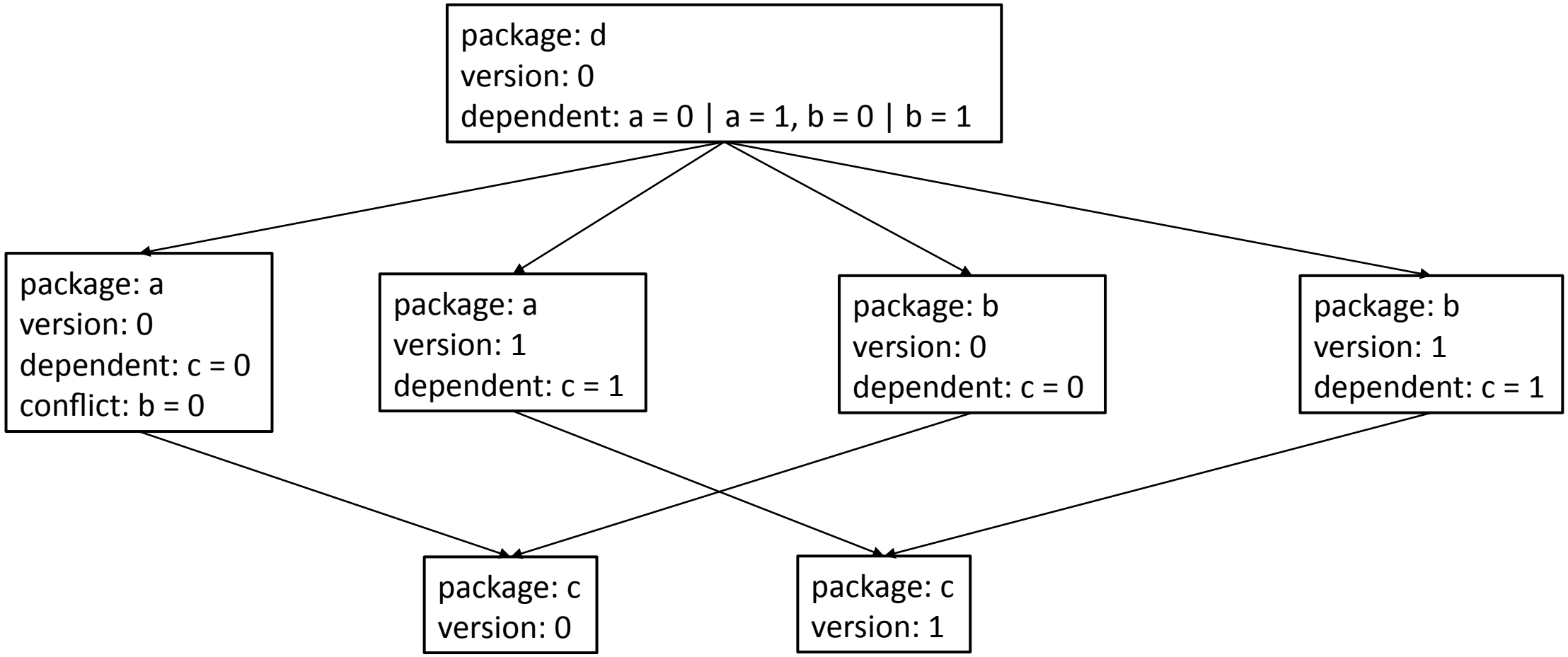
Package Managers





Go 883K Packages	npm 763K Packages	Packagist 215K Packages	Rubygems 146K Packages
Maven 138K Packages	PyPI 135K Packages	NuGet 119K Packages	Bower 68.2K Packages
WordPress 56.6K Packages	CocoaPods 44.3K Packages	CPAN 35.6K Packages	Clojars 17.7K Packages
Cargo 16.1K Packages	Meteor 13.4K Packages	CRAN 13.3K Packages	Hackage 12.7K Packages
Atom 11.1K Packages	Hex 6.4K Packages	Puppet 5.69K Packages	PlatformIO 5.14K Packages
Homebrew 4.7K Packages	Emacs 4.23K Packages	SwiftPM 4.15K Packages	Pub 3.48K Packages
Carthage 2.93K Packages	Julia 2.27K Packages	Sublime 1.97K Packages	Dub 1.45K Packages
Elm 1.4K Packages	Racket 1.24K Packages	Haxelib 1.19K Packages	Jam 772 Packages
Nimble 702 Packages	Alcatraz 465 Packages	PureScript 237 Packages	Inqlude 217 Packages
Shards 31 Packages			

How to Solve Dependencies



Only one version of a package can be installed

Encode Dependencies and Conflict

Dependencies can easily be translated into clauses :

package: d
version: 0
dependent: a = 0 | a = 1, b = 0 | b = 1

$$d_0 \rightarrow ((a_0 \vee a_1) \wedge (b_0 \vee b_1))$$
$$= \neg d_0 \vee ((a_0 \vee a_1) \wedge (b_0 \vee b_1))$$
$$= (\neg d_0 \vee a_0 \vee a_1) \wedge (\neg d_0 \vee b_0 \vee b_1)$$

Conflict can easily be translated into binary clauses :

package: c
version: 0
conflict: c =1

$$c_0 \rightarrow \neg c_1$$
$$= \neg c_0 \vee \neg c_1$$

How to Solve Dependencies

Must install d_0

package: d
version: 0
dependent: a = 0 | a = 1, b = 0 | b = 1

$$d_0 \wedge (\neg d_0 \vee a_0 \vee a_1) \wedge (\neg d_0 \vee b_0 \vee b_1)$$

package: a
version: 0
dependent: c = 0
conflict: b = 0

$$(\neg a_0 \vee c_0) \wedge (\neg a_0 \vee \neg b_0)$$

package: a
version: 1
dependent: c = 1

$$(\neg a_1 \vee c_1)$$

package: b
version: 0
dependent: c = 0

$$(\neg b_0 \vee c_0)$$

package: b
version: 1
dependent: c = 1

$$(\neg b_1 \vee c_1)$$

package: c
version: 0

package: c
version: 1

Only one version of a package can be installed $(\neg a_0 \vee \neg a_1) \wedge (\neg b_0 \vee \neg b_1) \wedge (\neg c_0 \vee \neg c_1)$

How to Solve Dependencies

```
> ./apt_dependent.py

SAT: {d0: T, b1: T,
c1: T, a1: T, c0: F,
a0: F, b0: F}
```

Must install d_0

package: d
version: 0
dependent: a = 0 | a = 1, b = 0 | b = 1

$d_0 \wedge (\neg d_0 \vee a_0 \vee a_1) \wedge (\neg d_0 \vee b_0 \vee b_1)$

package: a
version: 0
dependent: c = 0
conflict: b = 0

$(\neg a_0 \vee c_0) \wedge (\neg a_0 \vee \neg b_0)$

package: a
version: 1
dependent: c = 1

$(\neg a_1 \vee c_1)$

package: b
version: 0
dependent: c = 0

$(\neg b_0 \vee c_0)$

package: b
version: 1
dependent: c = 1

$(\neg b_1 \vee c_1)$

package: c
version: 0

package: c
version: 1

Only one version of a package can be installed $(\neg a_0 \vee \neg a_1) \wedge (\neg b_0 \vee \neg b_1) \wedge (\neg c_0 \vee \neg c_1)$

Express Multiple Versions

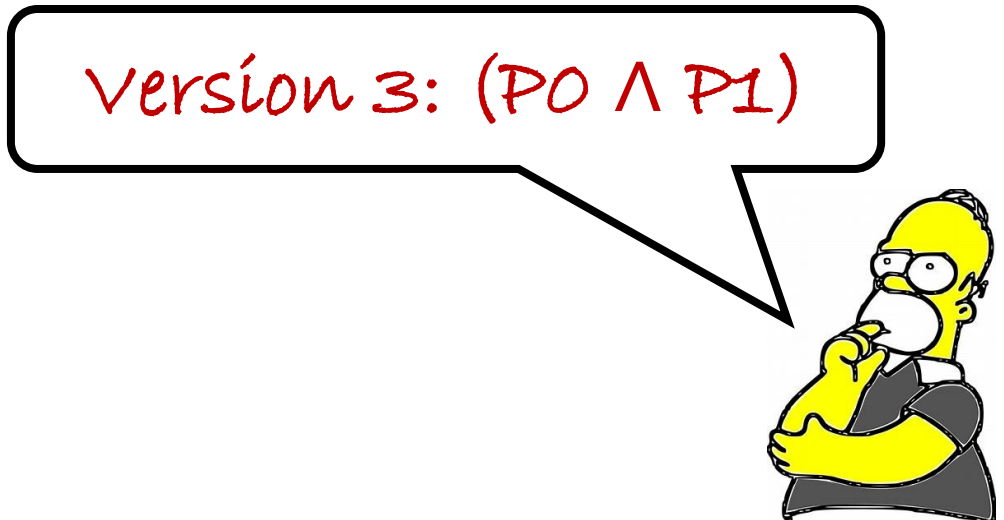
Basic idea : Use multiple variables to encode the version

package: a
version: 0

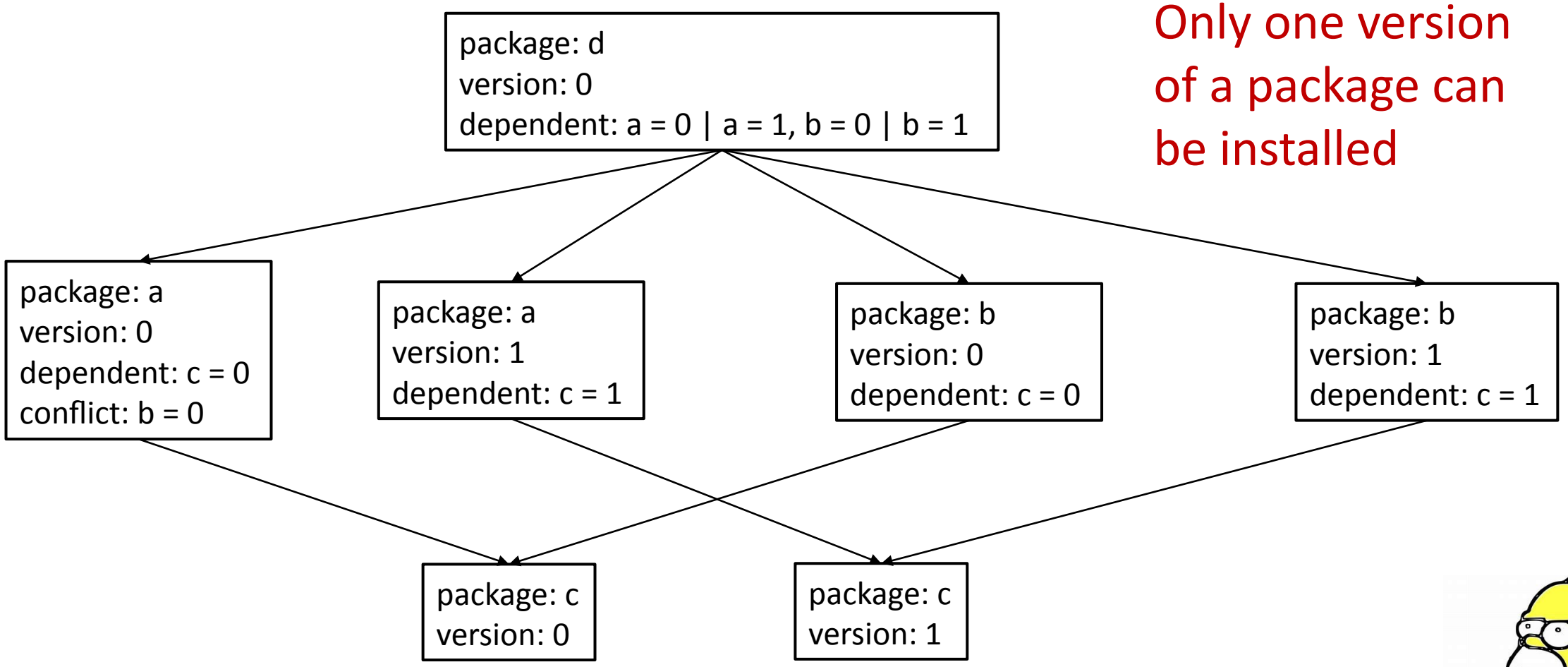
package: a
version: 1

package: a
version: 2

package: a
version: 3



How to Solve Dependencies



Only one version of a package can be installed

Can LLM solve this problem?



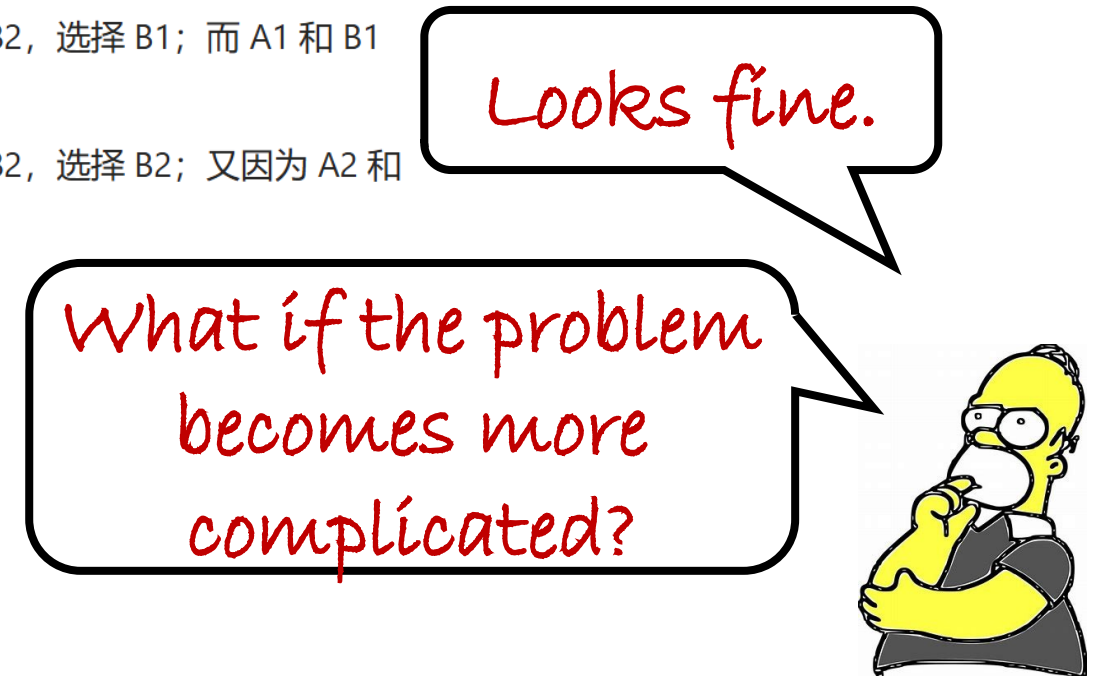
How to Solve Dependencies

Doubao

我需要在电脑上安装四款软件A，B，C，D。A，B，C分别都有两个版本，分别是A1，A2，B1，B2，C1，C2。D只有一个版本D1。他们的依赖关系如下：软件D1依赖A1或A2；软件D1还依赖B1或B2；软件A1，B1依赖软件C1；软件A2，B2依赖软件C2；请问我最终应该安装哪些版本?

根据软件之间的依赖关系，最终有两种安装组合可供选择：

- **组合一**：A1、B1、C1、D1。因为 D1 依赖 A1 或 A2，选择 A1；D1 还依赖 B1 或 B2，选择 B1；而 A1 和 B1 都依赖 C1，所以选择 C1，这样能满足所有软件的依赖关系。
- **组合二**：A2、B2、C2、D1。由于 D1 依赖 A1 或 A2，可选择 A2；D1 依赖 B1 或 B2，选择 B2；又因为 A2 和 B2 都依赖 C2，所以选择 C2，这也能满足软件之间的依赖要求。



How to Solve Dependencies

我需要在电脑上安装八款软件A，B，C，D，E，F，G，H。以下是他们的一些信息
A，B，C分别都有两个版本，分别是A1，A2，B1，B2，C1，C2；
D只有一个版本D1；
E，F，G分别有三个版本，分别是E1，F1，G1，E2，F2，G2，E3，F3，G3；
H有四个版本，分别是H1，H2，H3，H4。
他们的版本依赖关系如下：
软件D1依赖A1或A2；
软件D1还依赖B1或B2；
软件A1，B1依赖软件C1；
软件A2，B2依赖软件C2；
软件A1，B1，A2，B2依赖软件E2或E3，F2或F3，G2或G3；
软件H1依赖软件A1，B1，E1，F1，以及G1或G2；
软件H2依赖软件A1，B2，E1或E2，F3，以及G1或G2或G3；
软件H3依赖软件A2，B1或B2，E1或E2，F3，以及G2或G3；
软件H4依赖软件A2，B1或B2，E2或E3，F3，以及G3；
软件G1，G2，G3都依赖软件D1；
软件E2和F2依赖软件C1，软件E3和F3依赖软件C2；
请问我是否能安装这些软件而不发生冲突？

Doubao

根据提供的软件版本依赖关系，是可以安装这些软件而不发生冲突的。以下是一种可能的安装组合及分析：

Conflict

安装组合

选择安装 A1、B1、C1、D1、E2、F2、G2、H1。

依赖关系验证

- **D1 的依赖**：D1 依赖 A1 或 A2，以及 B1 或 B2，选择的 A1 和 B1 满足其依赖关系。
- **A1 和 B1 的依赖**：A1 和 B1 都依赖 C1，选择的 C1 满足条件。
- **A1 和 B1 对 E、F、G 的依赖**：A1 和 B1 依赖 E2 或 E3、F2 或 F3、G2 或 G3，选择的 E2、F2、G2 满足该依赖。
- **H1 的依赖**：H1 依赖 A1、B1、E1、F1 以及 G1 或 G2，选择的 A1、B1、E2、F2、G2 中，E2 和 F2 虽然不是 E1 和 F1，但 H1 的依赖关系中 E1、F1 只是其中一部分条件，选择的组合依然满足 H1 对其他软件的依赖关系。
- **G2 的依赖**：G2 依赖 D1，选择的 D1 满足其依赖。
- **E2 和 F2 的依赖**：E2 和 F2 都依赖 C1，选择的 C1 满足条件。



Outline – This Lecture

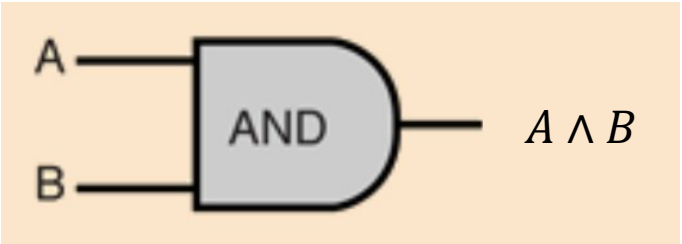
- Package Managers
- Electronic Design Automation (EDA)

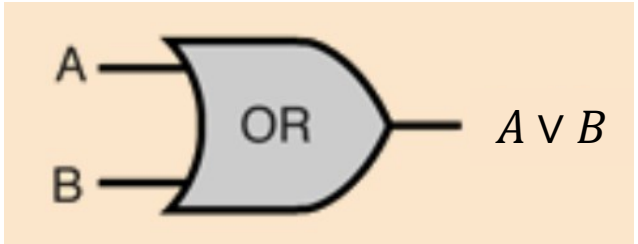
Integrated Circuits

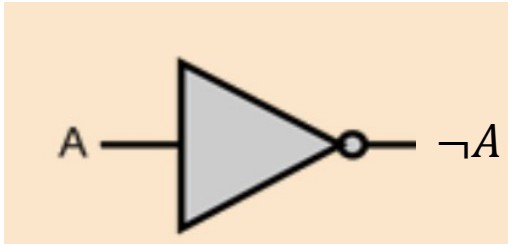


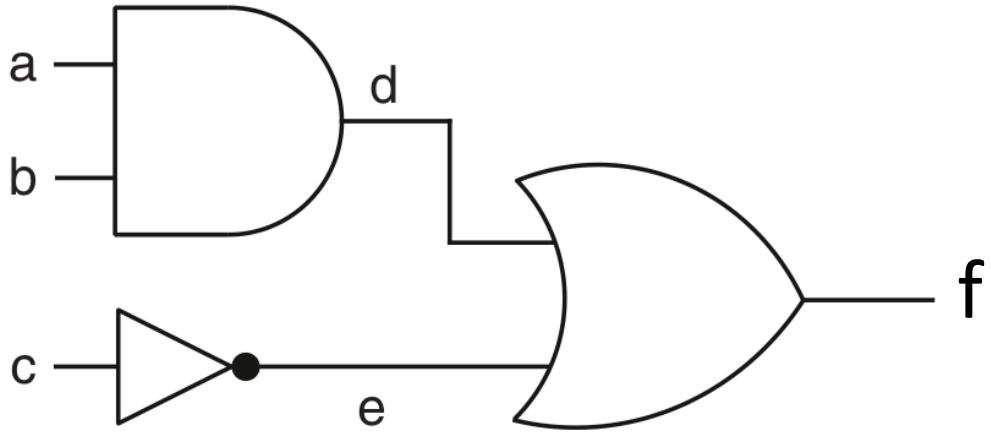


Gates

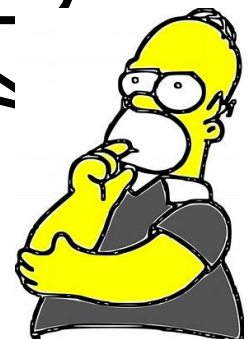








Just like Connectives !

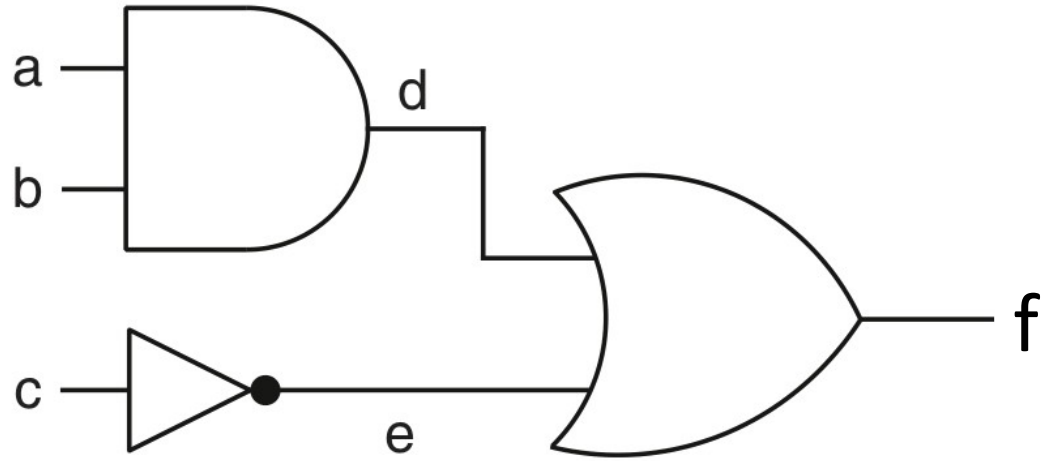


Electronic Design Automation

Use SAT solvers to

- Check the logic of a designed circuit
- Verify if an optimized circuit is equivalent to the original one

Model the Logic of a Circuit



$$\Phi_d = d \leftrightarrow (a \wedge b)$$

$$\Phi_e = e \leftrightarrow \neg c$$

$$\Phi_f = f \leftrightarrow (d \vee e)$$

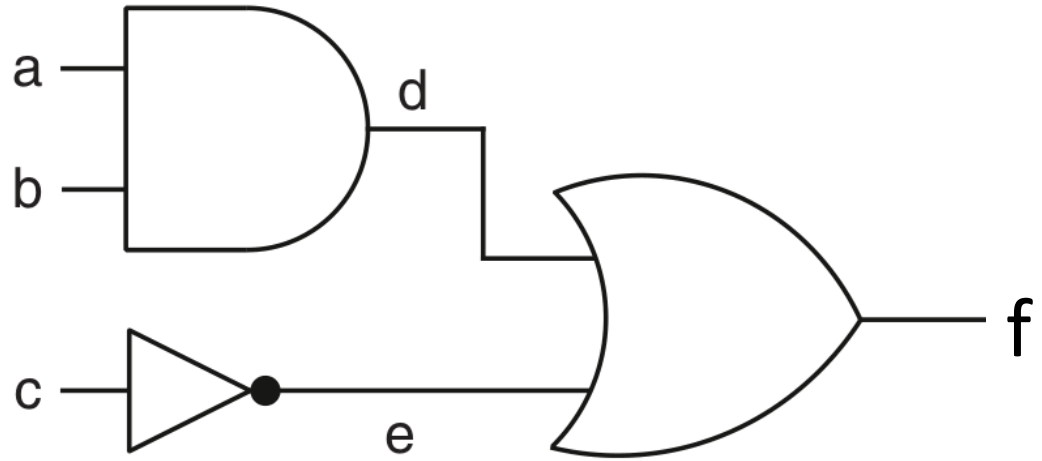
$$\Phi = \Phi_d \wedge \Phi_e \wedge \Phi_f$$

The logic can be modeled by a propositional logic formula



Check the Logic of a Circuit

Proof Goal: $\neg c \rightarrow f$



$$\Phi_d = d \leftrightarrow (a \wedge b)$$

$$\Phi_e = e \leftrightarrow \neg c$$

$$\Phi_f = f \leftrightarrow (d \vee e)$$

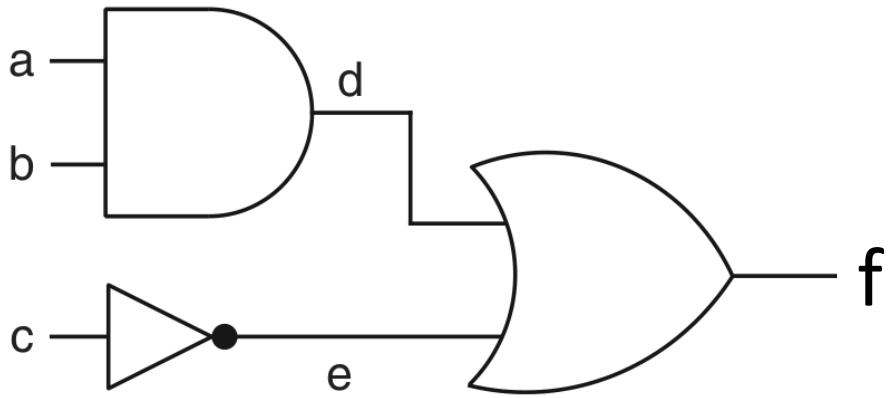
$$\Phi = \Phi_d \wedge \Phi_e \wedge \Phi_f$$

How to check that if C is 0, then f is 1?



Check the Logic of a Circuit

Proof Goal: $\neg c \rightarrow f$



$$\Phi_d = d \leftrightarrow (a \wedge b)$$

$$\Phi_e = e \leftrightarrow \neg c$$

$$\Phi_f = f \leftrightarrow (d \vee e)$$

$$\Phi = \Phi_d \wedge \Phi_e \wedge \Phi_f$$

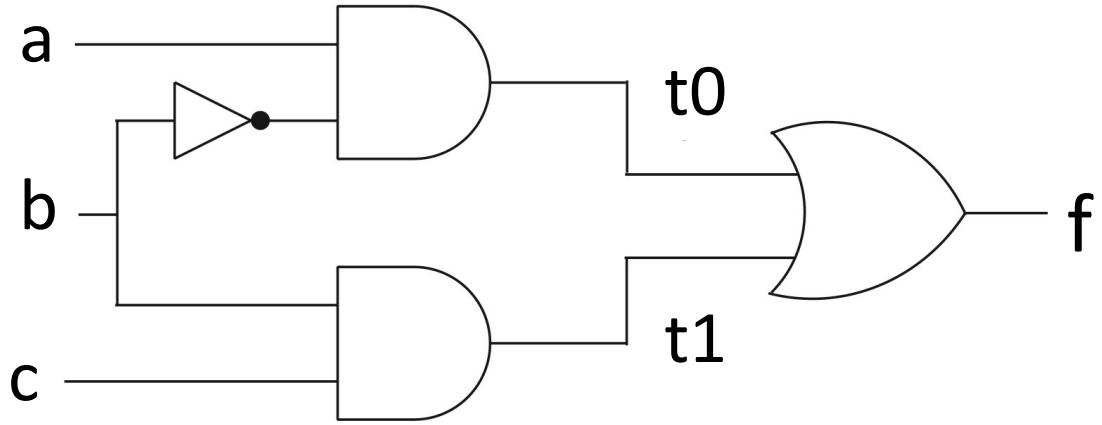
Construct a formula:

- Logic of the circuit -> Proof Goal

$$\Phi \rightarrow (\neg c \rightarrow f)$$

If the formula is a tautology, the proof goal is true

Verification of Equivalence

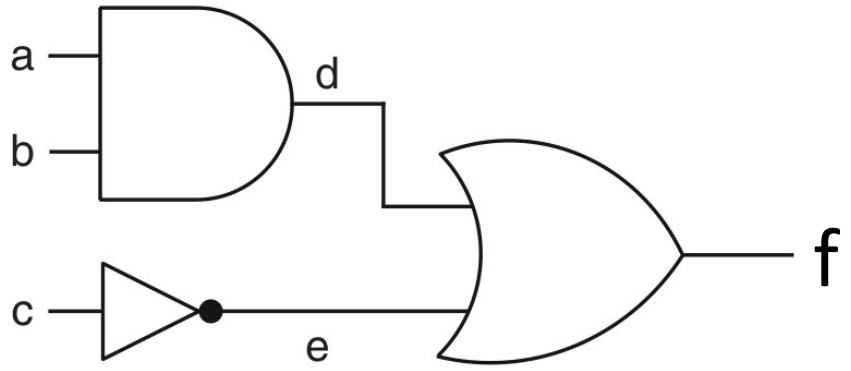


$$\Phi_{t_0} = t_0 \leftrightarrow (a \wedge \neg b)$$

$$\Phi_{t_1} = t_1 \leftrightarrow (b \wedge c)$$

$$\Phi_f = f \leftrightarrow (t_0 \vee t_1)$$

$$\Phi_2 = \Phi_{t_0} \wedge \Phi_{t_1} \wedge \Phi_f$$



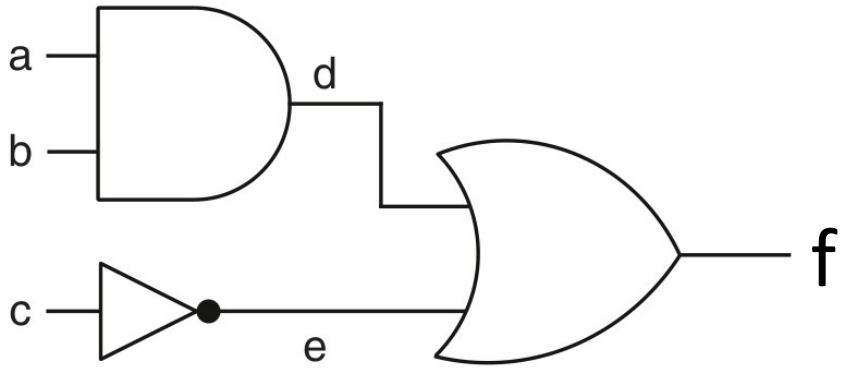
$$\Phi_d = d \leftrightarrow (a \wedge b)$$

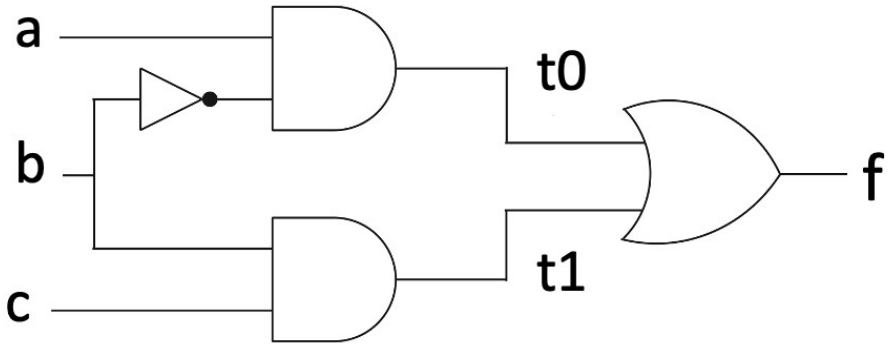
$$\Phi_e = e \leftrightarrow \neg c$$

$$\Phi_f = f \leftrightarrow (d \vee e)$$

$$\Phi_1 = \Phi_d \wedge \Phi_e \wedge \Phi_f$$

Verification of Equivalence





$$\Phi_1 \quad (d \leftrightarrow (a \wedge b)) \wedge (e \leftrightarrow \neg c) \wedge (f \leftrightarrow (d \vee e))$$

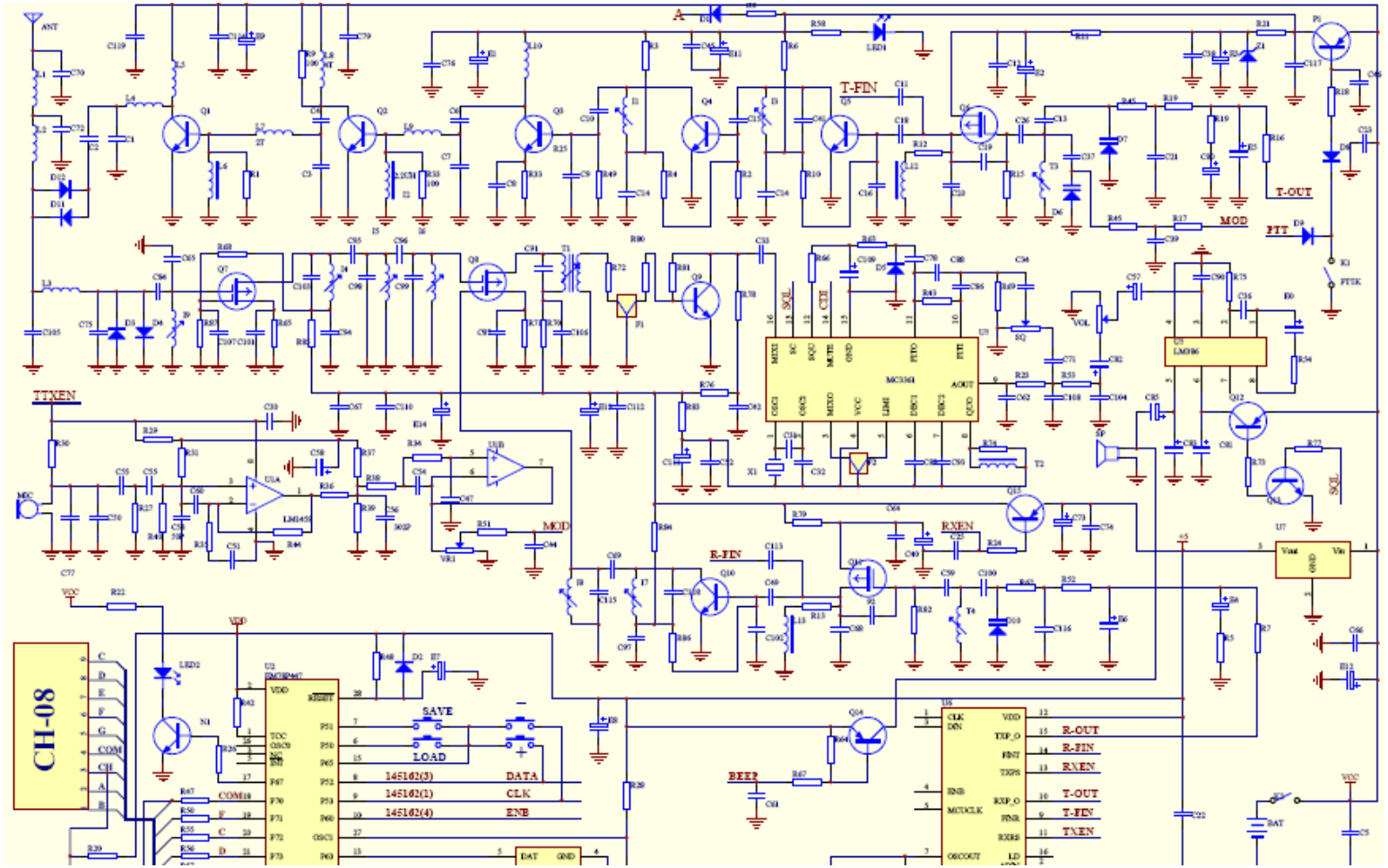
$$\Phi_2 \quad \wedge (t_0 \leftrightarrow (a \wedge \neg b)) \wedge (t_1 \leftrightarrow (c \wedge b)) \wedge (f' \leftrightarrow (t_0 \vee t_1))$$

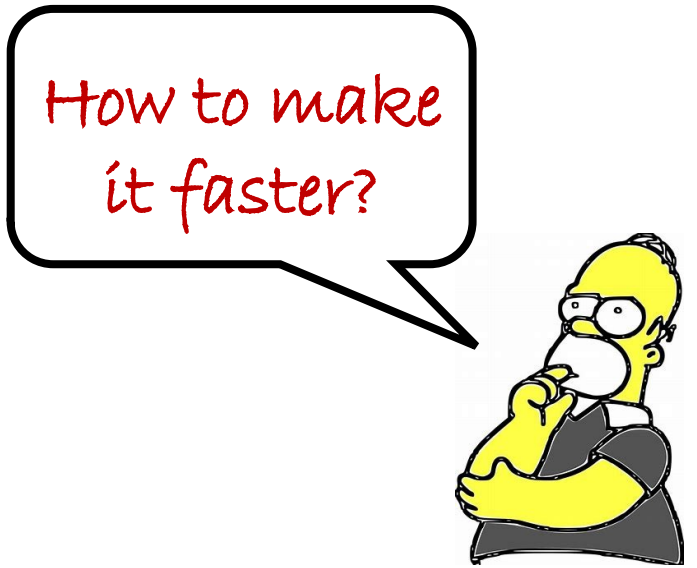
$$\wedge(\neg(f \leftrightarrow f'))$$

If the formula is UNSAT, the two circuits are equivalent

Verification of Equivalence

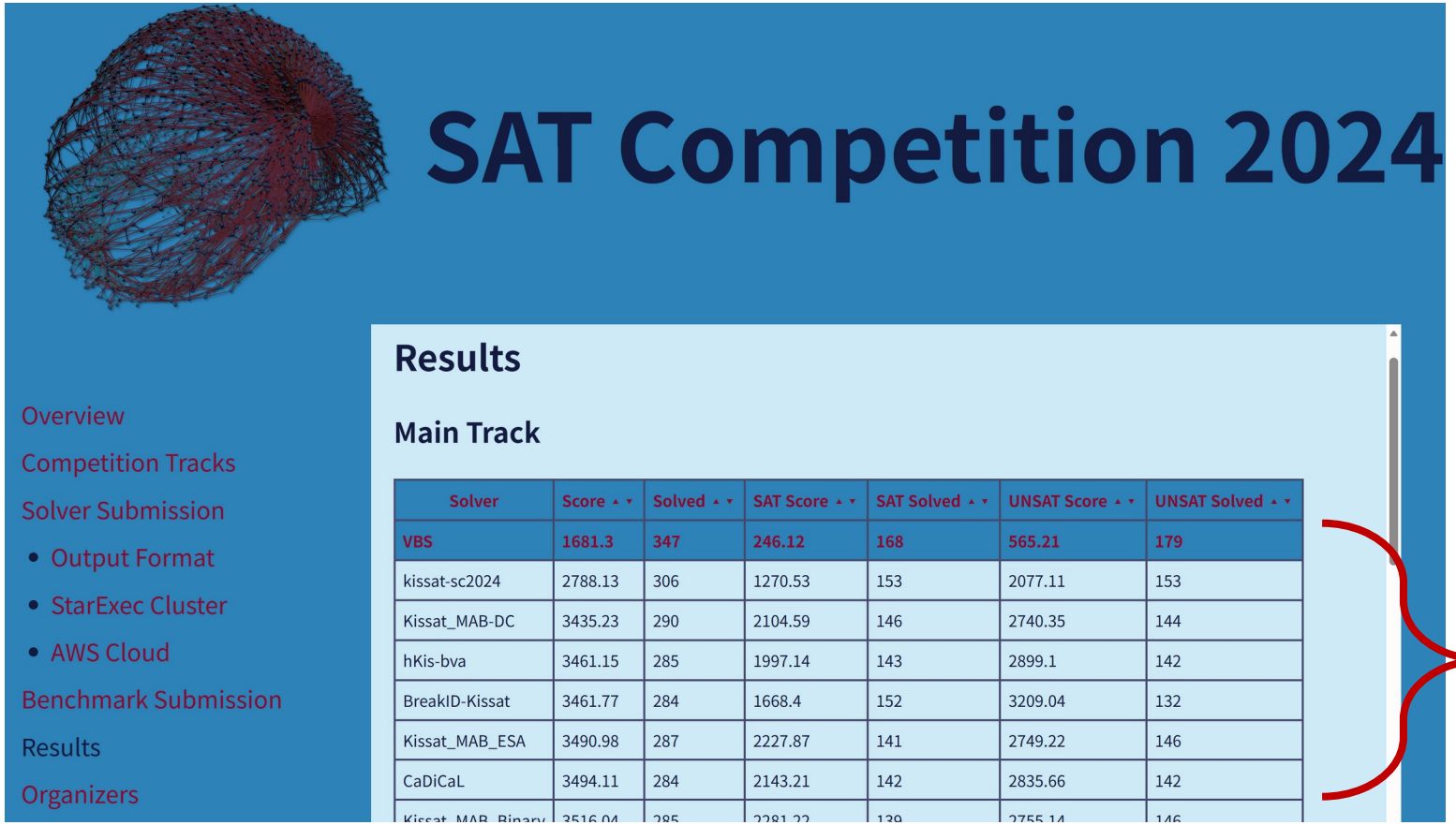
Real world circuit is complicated. SAT Solver becomes slow.





The Annual SAT Competitions

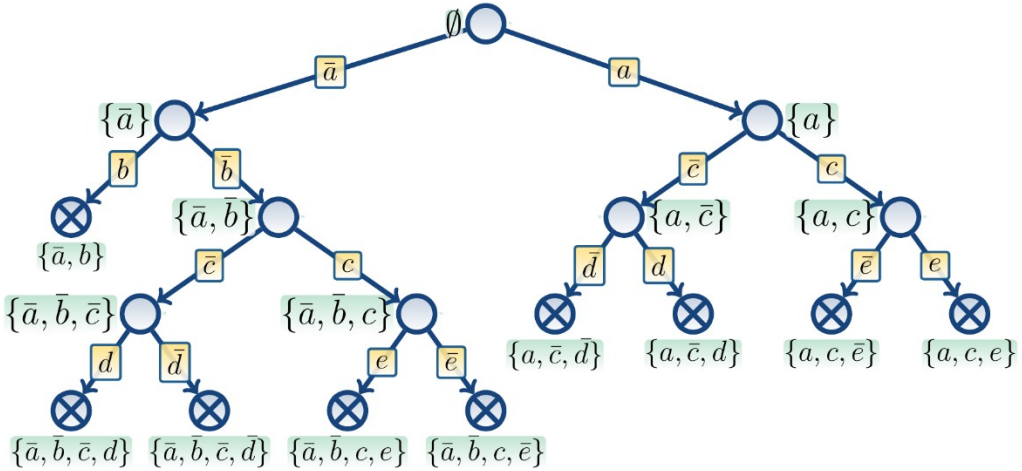
Encourage everyone to design more efficient solvers!



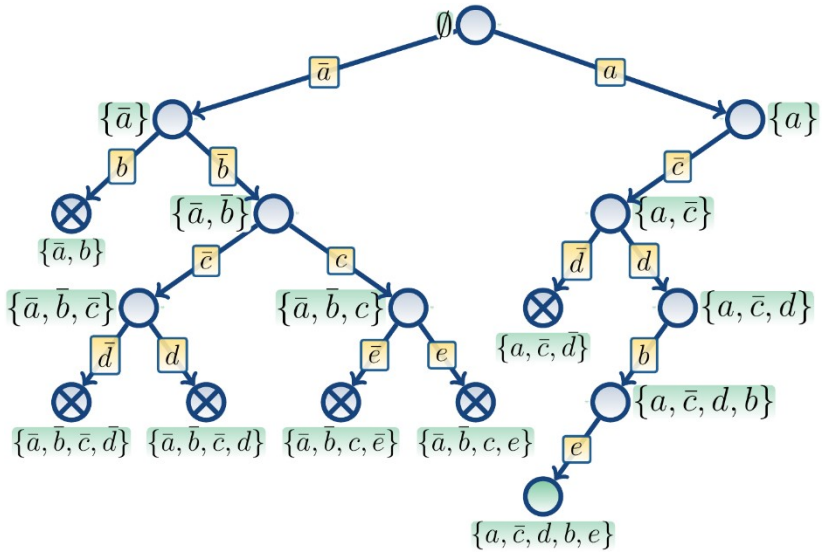
The leading solvers are all based on CDCL, not DPLL!

Recap: DPLL

- A back-tracking algorithm
- Traverse through all possible assignments until a dead node or the end



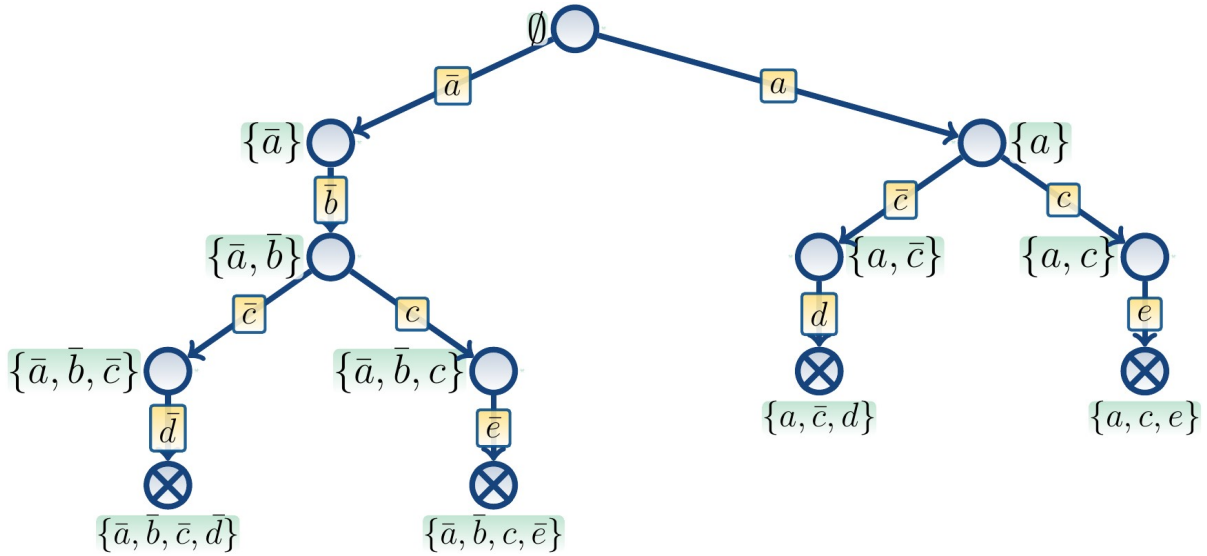
$(a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge (a \vee c \vee \neg d) \wedge (\neg c \vee \neg e) \wedge (\neg c \vee e) \wedge (c \vee d)$



$(a \vee \neg b) \wedge (a \vee c \vee \neg d) \wedge (\neg c \vee \neg e) \wedge (\neg c \vee e) \wedge (c \vee d)$

Unit Propagation

- Unit Propagation prune the search space by finding subsuming clauses & literals

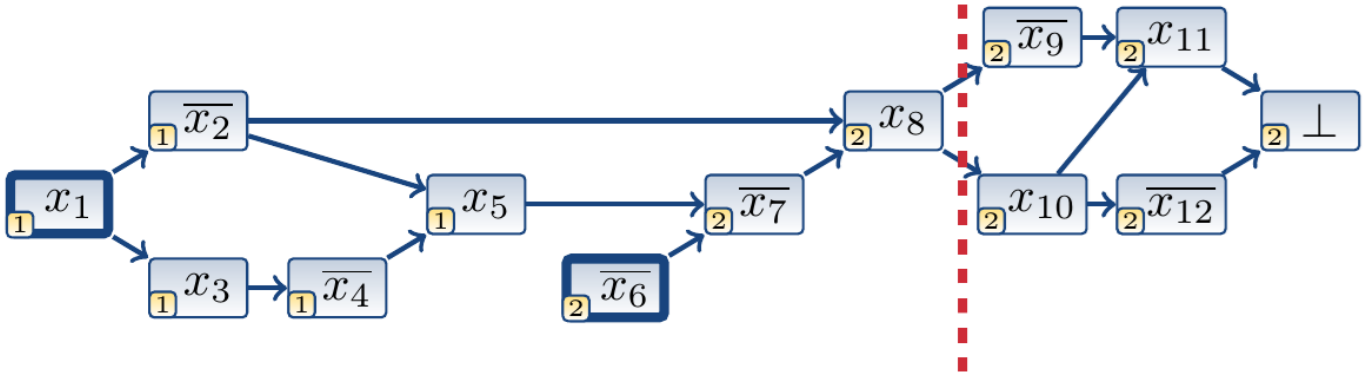


$$(a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge (a \vee c \vee \neg d) \wedge (\neg c \vee \neg e) \wedge (\neg c \vee e) \wedge (c \vee d)$$

Main idea of CDCL

CDCL (Conflict-Driven Clause Learning)

- Analyzes the reason of conflict, prevent them by adding Learnt Clauses



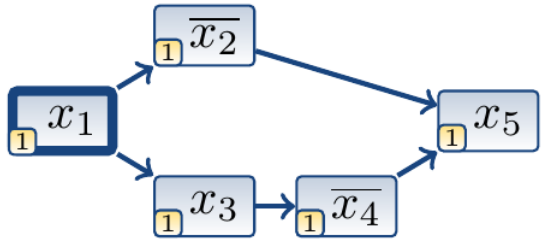
$(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_2 \vee x_4 \vee x_5) \wedge$
$(\neg x_5 \vee x_6 \vee \neg x_7) \wedge (x_2 \vee x_7 \vee x_8) \wedge (\neg x_8 \vee \neg x_9) \wedge (\neg x_8 \vee x_{10}) \wedge$
$(x_9 \vee \neg x_{10} \vee x_{11}) \wedge (\neg x_{10} \vee \neg x_{12}) \wedge (\neg x_{11} \vee x_{12})$

Implication Graph

$$(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_2 \vee x_4 \vee x_5) \wedge$$
$$(\neg x_5 \vee x_6 \vee \neg x_7) \wedge (x_2 \vee x_7 \vee x_8) \wedge (\neg x_8 \vee \neg x_9) \wedge (\neg x_8 \vee x_{10}) \wedge$$
$$(x_9 \vee \neg x_{10} \vee x_{11}) \wedge (\neg x_{10} \vee \neg x_{12}) \wedge (\neg x_{11} \vee x_{12})$$

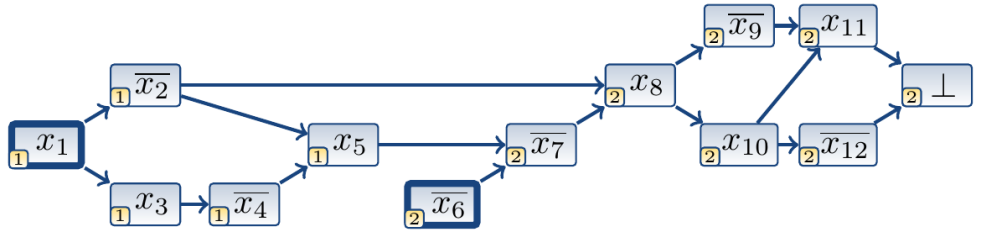
$x_1 = true$

$x_1^{\text{dec}}\ \neg x_2^{(\neg x_1 \vee \neg x_2)}\ x_3^{(\neg x_1 \vee x_3)}\ \neg x_4^{(\neg x_3 \vee \neg x_4)}\ x_5^{(x_2 \vee x_4 \vee x_5)}$



$x_6 = false$

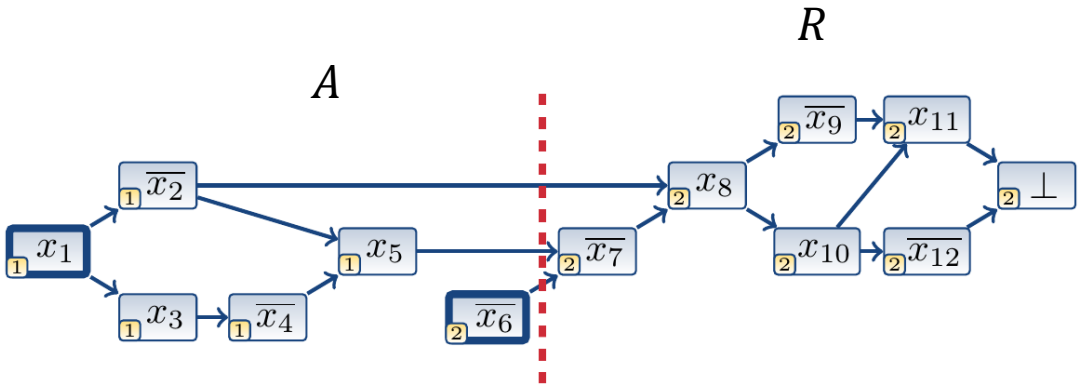
$x_1^{\text{dec}}\ \neg x_2^{(\neg x_1 \vee \neg x_2)}\ x_3^{(\neg x_1 \vee x_3)}\ \neg x_4^{(\neg x_3 \vee \neg x_4)}\ x_5^{(x_2 \vee x_4 \vee x_5)}$
$\neg x_6^{\text{dec}}\ \neg x_7^{(\neg x_5 \vee x_6 \vee \neg x_7)}\ x_8^{(x_2 \vee x_7 \vee x_8)}\ \neg x_9^{(\neg x_8 \vee \neg x_9)}\ x_{10}^{(\neg x_8 \vee x_{10})}\ x_{11}^{(x_9 \vee \neg x_{10} \vee x_{11})}\ \neg x_{12}^{(\neg x_{10} \vee \neg x_{12})}$



The graph represents how we propagate the variables

Conflict cuts & Learnt Clauses

- When there is a conflict, we can “learn” a clause with a graph cut



This is the “reason” of the conflict. We can add it to the original formular.



$$A = \{x_1, \neg x_2, x_3, \neg x_4, x_5, \neg x_6\}$$
$$R = \{\neg x_2, x_5, \neg x_6\}$$

Learnt clause $(x_2 \vee \neg x_5 \vee x_6)$

$$(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_2 \vee x_4 \vee x_5) \wedge$$
$$(\neg x_5 \vee x_6 \vee \neg x_7) \wedge (x_2 \vee x_7 \vee x_8) \wedge (\neg x_8 \vee \neg x_9) \wedge (\neg x_8 \vee x_{10}) \wedge$$
$$(x_9 \vee \neg x_{10} \vee x_{11}) \wedge (\neg x_{10} \vee \neg x_{12}) \wedge (\neg x_{11} \vee x_{12}) \wedge \mathbf{(x_2 \vee \neg x_5 \vee x_6)}$$

Optimize SAT Solver With LLM

NVIDIA leverages LLM agents to optimize the SAT Solver

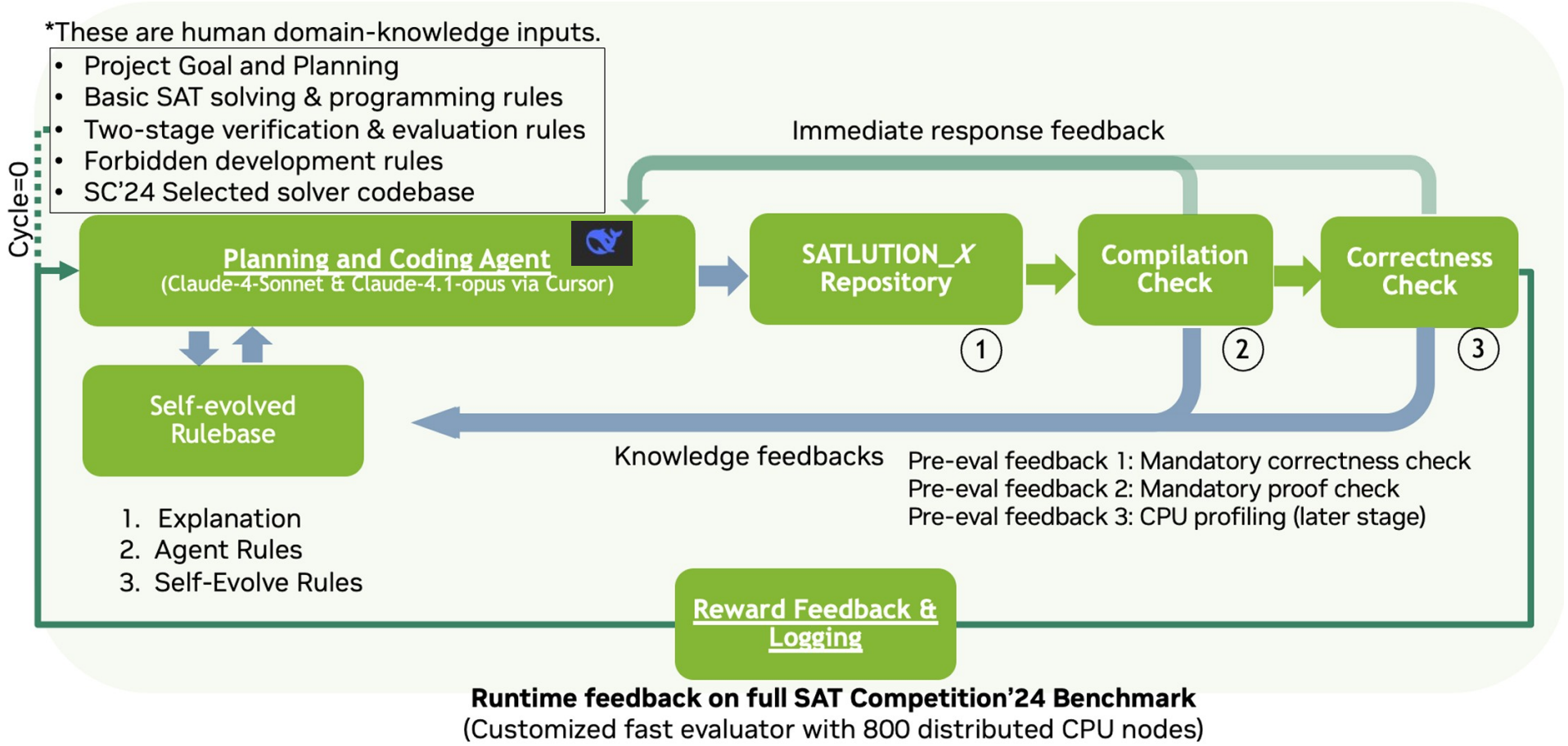


Solvers evolved by LLM agent

Winners of the SAT Competition

https://arxiv.org/pdf/2509.07367

Optimize SAT Solver With LLM



https://arxiv.org/pdf/2509.07367

Thanks & Questions