

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

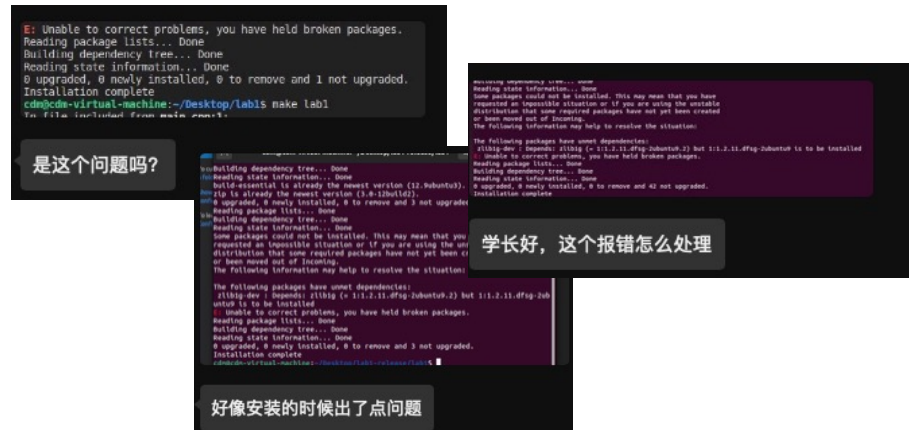
- Package Managers
- Electronic Design Automation (EDA)

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 installed  
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-libtrunk
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

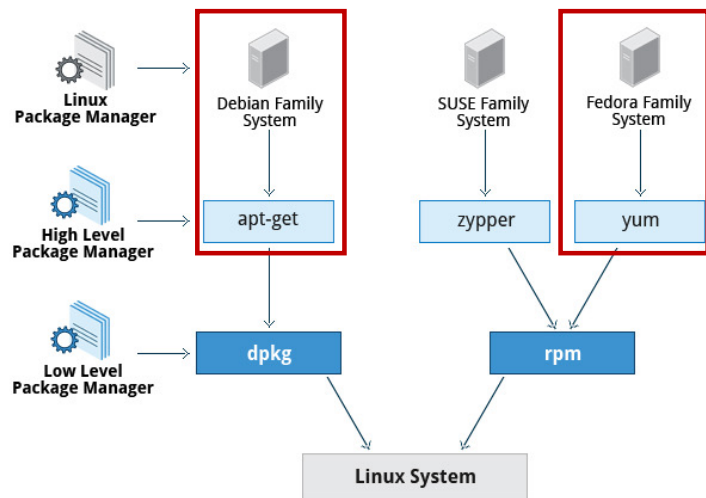
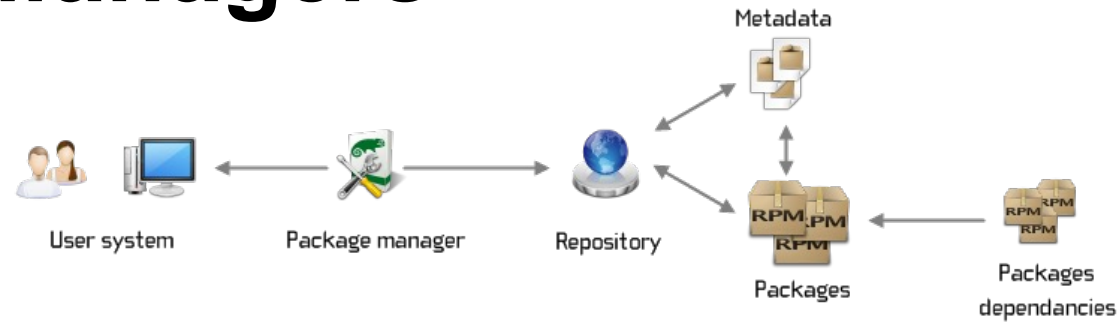
Firefox delivers safe, easy web browsing. A familiar user interface, enhanced security features including protection from online identity theft and integrated search let you get the most out of the web.

Install this firefox package too, if you want to be automatically updated to 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

Maven
138K Packages

WordPress
56.6K Packages

Cargo
16.1K Packages

Atom
11.1K Packages

Homebrew
4.7K Packages

Carthage
2.93K Packages

Elm
1.4K Packages

Nimble
702 Packages

Shards
31 Packages

npm
763K Packages

PyPI
135K Packages

CocoaPods
44.3K Packages

Meteor
13.4K Packages

Hex
6.4K Packages

Emacs
4.23K Packages

Julia
2.27K Packages

Racket
1.24K Packages

Alcatraz
465 Packages

PackageList
215K Packages

NuGet
119K Packages

CPAN
35.6K Packages

CRAN
13.3K Packages

Puppet
5.69K Packages

SwiftPM
4.15K Packages

Sublime
1.97K Packages

Haxelib
1.19K Packages

PureScript
237 Packages

Rubygems
146K Packages

Bower
68.2K Packages

Clojars
17.7K Packages

Hackage
12.7K Packages

PlatformIO
5.14K Packages

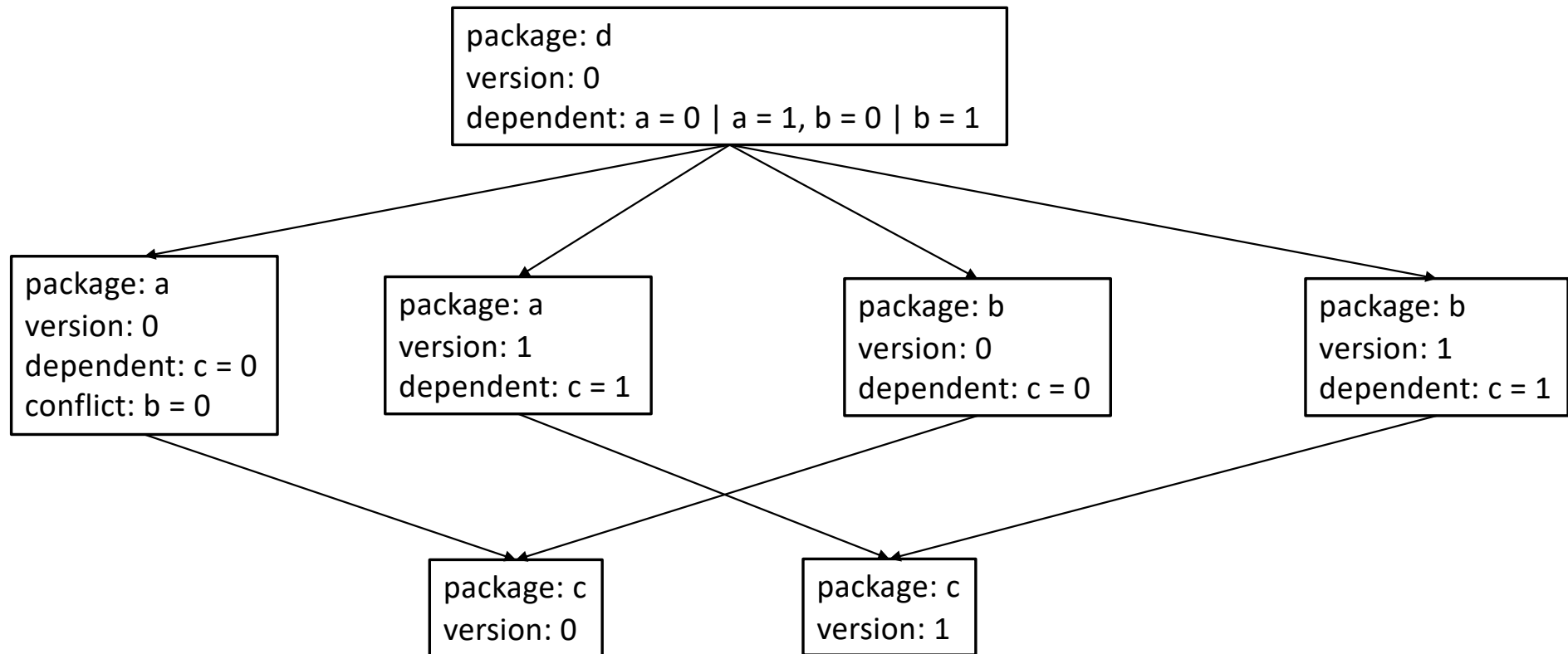
Pub
3.48K Packages

Dub
1.45K Packages

Jam
772 Packages

Inlude
217 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: $a = 0 \mid a = 1, b = 0 \mid b = 1$

$$\begin{aligned}d_0 &\rightarrow ((a_0 \vee a_1) \wedge (b_0 \vee b_1)) \\&= \neg d_0 \vee ((a_0 \vee a_1) \wedge (b_0 \vee b_1)) \\&= (\neg d_0 \vee a_0 \vee a_1) \wedge (\neg d_0 \vee b_0 \vee b_1)\end{aligned}$$

Conflict can easily be translated into binary clauses :

package: c
version: 0
conflict: $c = 1$

$$\begin{aligned}c_0 &\rightarrow \neg c_1 \\&= \neg c_0 \vee \neg c_1\end{aligned}$$

How to Solve Dependencies

Must install d_0

package: d
version: 0
dependent: $a = 0 \mid a = 1, b = 0 \mid b = 1$

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

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

package: a
version: 1
dependent: $c = 1$

package: b
version: 0
dependent: $c = 0$

package: b
version: 1
dependent: $c = 1$

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

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

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

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

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: $a = 0 \mid a = 1, b = 0 \mid b = 1$

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

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

package: a
version: 1
dependent: $c = 1$

package: b
version: 0
dependent: $c = 0$

package: b
version: 1
dependent: $c = 1$

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

$(\neg a_1 \vee c_1)$

$(\neg b_0 \vee c_0)$

$(\neg b_1 \vee c_1)$

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

Version 3: $(P0 \wedge P1)$



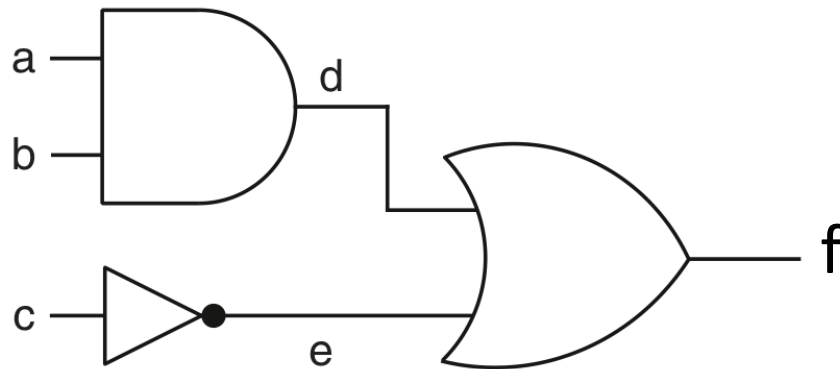
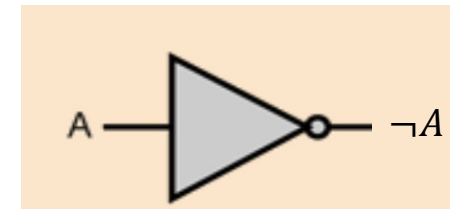
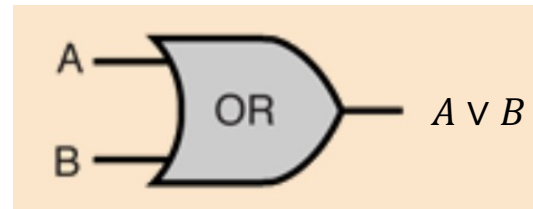
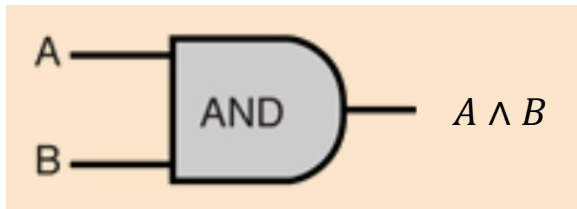
Outline – This Lecture

- Package Managers
- Electronic Design Automation (EDA)

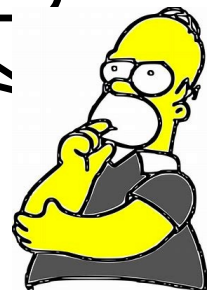
Integrated Circuits



Gates



Just like Connectives!

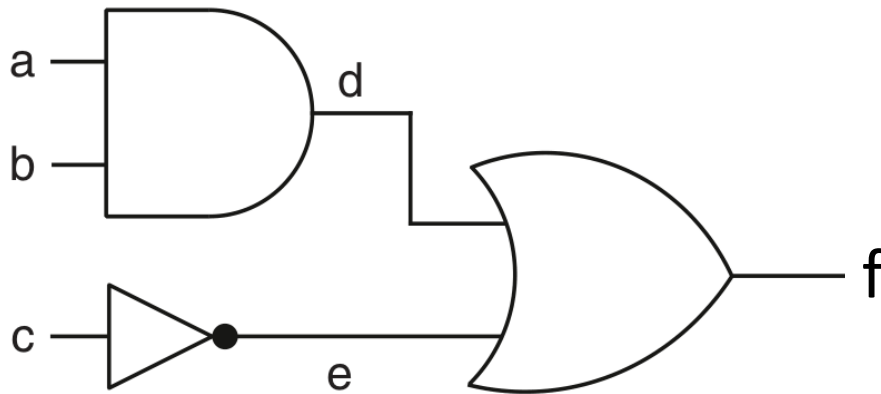


Electronic Design Automation

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



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

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

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

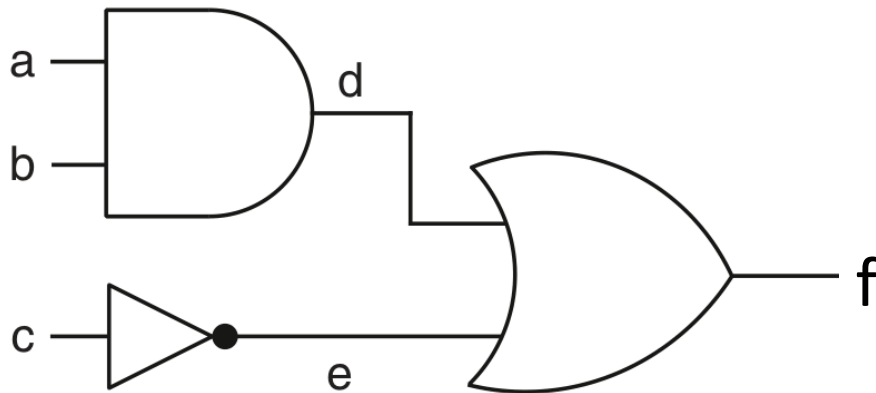
$$\Phi = \Phi_d \wedge \Phi_e \wedge \Phi_f$$

The logic can be modeled
by a propositional logic
formula



Check the Logic of a Circuit

Proof Goal: $\neg c \rightarrow f$



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

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

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

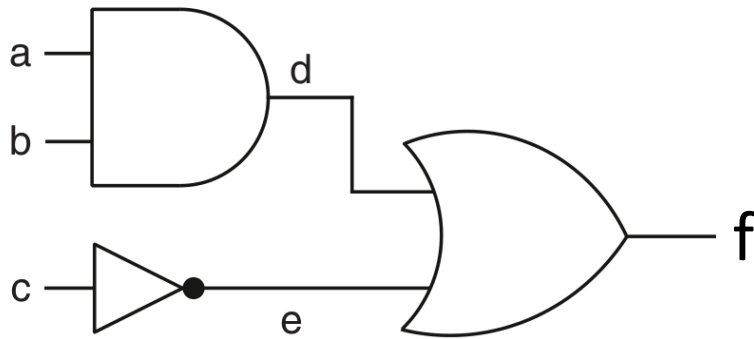
$$\Phi = \Phi_d \wedge \Phi_e \wedge \Phi_f$$

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



Check the Logic of a Circuit

Proof Goal: $\neg c \rightarrow f$



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

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

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

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

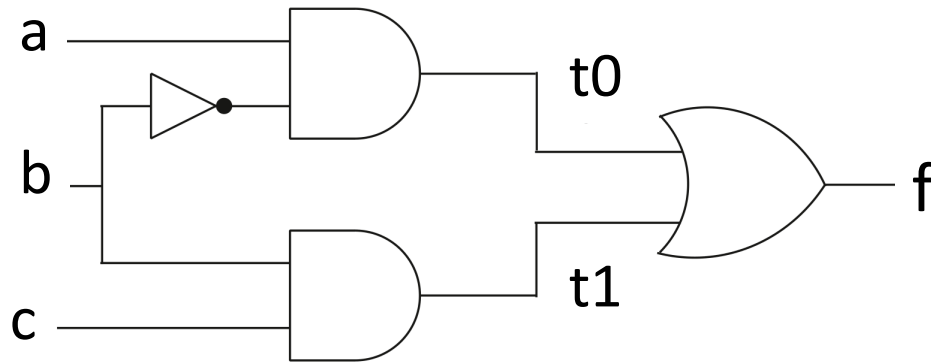
Construct a formula:

- Logic of the circuit $\rightarrow$ Proof Goal

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

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

Verification of Equivalence

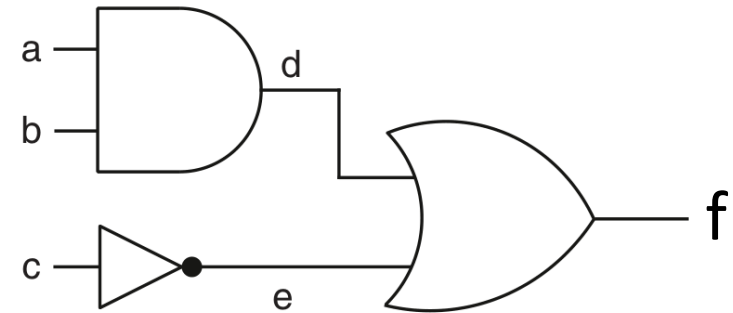


$$\Phi_{t_0} = t_0 \leftrightarrow (a \wedge \neg b)$$

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

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

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



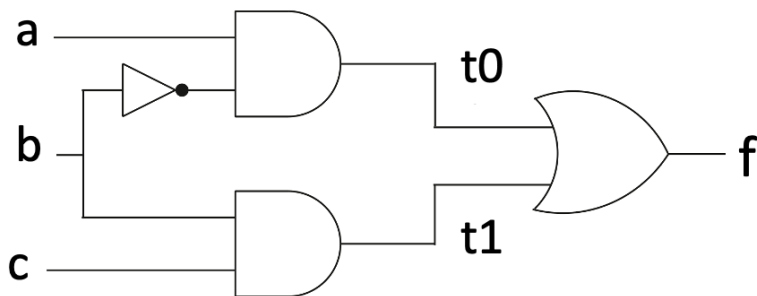
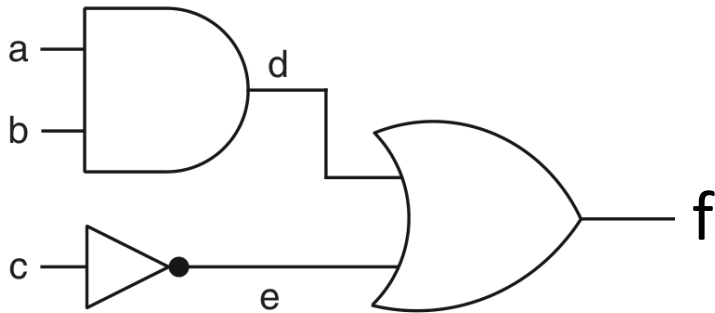
$$\Phi_d = d \leftrightarrow (a \wedge b)$$

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

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

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

Verification of Equivalence



Φ_1

$$\begin{aligned} & (d \leftrightarrow (a \wedge b)) \\ & \wedge (e \leftrightarrow \neg c) \\ & \wedge (f \leftrightarrow (d \vee e)) \end{aligned}$$

Φ_2

$$\begin{aligned} & \wedge (t_0 \leftrightarrow (a \wedge \neg b)) \\ & \wedge (t_1 \leftrightarrow (c \wedge b)) \\ & \wedge (f' \leftrightarrow (t_0 \vee t_1)) \\ & \wedge (f \oplus f') \end{aligned}$$

If the formula is UNSAT, the two circuits are equivalent

Thanks & Questions