Outline

Logic and Modelling

Satisfiability-checking in propositional logic has many applications.

Logic and Modelling

Satisfiability-checking in propositional logic has many applications.

There is a gap between real-life problems and their representation in propositional logic.

Logic and Modelling

Satisfiability-checking in propositional logic has many applications.

There is a gap between real-life problems and their representation in propositional logic.

Many application domains have special modelling languages for describing applications. Descriptions written in these languages can then be translated to propositional logic ...

Logic and Modelling

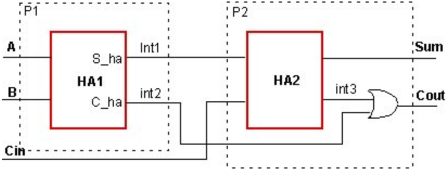
Satisfiability-checking in propositional logic has many applications.

There is a gap between real-life problems and their representation in propositional logic.

Many application domains have special modelling languages for describing applications. Descriptions written in these languages can then be translated to propositional logic . . .

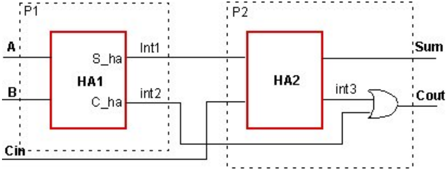
because propositional logic is not convenient for modelling.

Circuit Design



Circuit: propositional logic

Circuit Design



Circuit: propositional logic

```
library ieee;
use ieee.std_logic_1164.all;
entity FULL_ADDER is
  port (A, B, Cin :  in std_logic;
    Sum, Cout :  out std_logic);
end FULL_ADDER;
architecture BEHAV_FA of FULL_ADDER is
signal int1, int2, int3:  std_logic;
begin
P1:  process (A, B)
  begin
    int1<= A xor B;
    int2<= A and B;
  end process;
P2:  process (int1, int2, Cin)
  begin
    Sum <= int1 xor Cin;
    int3 <= int1 and Cin;
    Cout <= int2 or int3;
  end process;
end BEHAV_FA;
```

Design: high-level description (VHDL)

Scheduling

	All Second Year Timetable 2009-2010				Level 2
Printable Timetable	Monday	Tuesday	Wednesday	Thursday	Friday
08:00	-	-	-	-	-
09:00	MATH20701 CRAW TH.1	COMP20051 1.1	GCOMP20340[B] G23 GCOMP20340[A] IT407 HCOMP20411[A] G23 BMAN20890 MBS EAST B8	FCOMP20411[A] G23 FCOMP20081[B] G23 HCOMP20010 UNIX BMAN10621 ROSCOE 1.007	ICOMP20340[B] UNIX ICOMP20340[A] IT407 FCOMP20051[A w3+] G23 GCOMP20411[A] UNIX HCOMP20081[B] G23
10:00	BMAN20880† SIMON B (B.41) COMP20340 1.1 MATH20701 Mans Coop G20	BMAN21061 CRAW TH.1 GCOMP20010 G23 FCOMP20241[w3+] Toot 1 BMAN10621 1.1	GCOMP20340[B] G23 GCOMP20340[A] IT407 HCOMP20411[A] G23	BMAN10621 ROSCOE 1.007 FCOMP20411[A] G23 FCOMP20081[B] G23 HCOMP20010 UNIX	MATH20701 RENO C016 ICOMP20340[B] UNIX ICOMP20340[A] IT407 FCOMP20051[A w3+] G23 GCOMP20411[A] UNIX HCOMP20081[B] G23
11:00	BMAN20871 MBS EAST B8 MATH29631 SACKVILLE F047 MATH10141 SIMON 3	BMAN21061 CRAW TH.1 GCOMP20010 G23 FCOMP20241[w3+] Toot 1 BMAN10621 1.1	COMP20081[A] 1.1 F+ICOMP20081[B] G23 MATH29631 RENO G002 BMAN10621 ROSCOE 1.008	GCOMP20051[A w3+] G23 ICOMP20010 UNIX EEEN20027 RENO C009 MATH20111 TURING G.107	HCOMP20340[B] UNIX HCOMP20340[A] IT407 ICOMP20081[B] G23 ICOMP20411[A] G23 FCOMP20241 LF15 MATH10141 RENO C016
12:00	BMAN21061 ROSCOE 1.008 EEEN20019 RENO C002 MATH20411 SCH BLACKETT	COMP-PASS LF15 MATH20411 TURING G.107	G+HCOMP20081[B] G23 MATH10141 RENO C016 MATH20701 SCH MOS	MATH20111 TURING G.207 GCOMP20051[A w3+] G23 ICOMP20010 UNIX	MATH20201 UNI PL B HCOMP20340[B] UNIX HCOMP20340[A] IT407 ICOMP20081[B] G23 ICOMP20411[A] G23
13:00	FCOMP20340[A] IT407 FCOMP20340[B] UNIX GCOMP20081[B] G23 ICOMP20051[A w3+] G23 MATH20411 TURING G.107	COMP20411 1.1	-	COMP20141 1.1 MATH20701 TURING G.107	EEEN20019 SSB A16
14:00	BMAN20880 SIMON 3 (3.40) EEEN20019 RENO C009 MATH20111 TURING G.207 FCOMP20340[A] IT407 FCOMP20340[B] UNIX GCOMP20081[B] G23 ICOMP20051[A w3+] G23	EEEN-LAB ? COMP20411 1.1	-	BMAN21061 CRAW TH.2 MATH20201 ROSC A	COMP20141 1.1 EEEN20019 SSB A16
15:00	HCOMP20051[A w3+] G23 FCOMP20010 UNIX BMAN20880 SIMON 3 (3.40)	2nd Yr Tutorial GCOMP20241[w3+] Toot 1 EEEN-LAB ?	-	COMP20051 1.1	COMP20010 1.1 MATH29631 SACKVILLE G037
16:00	MATH20201 RENO C016 HCOMP20051[A w3+] G23 FCOMP20010 UNIX	CARS20021 UNI PL B MATH20411 SCH BLACKETT GCOMP20241[w3+] Toot 1 EEEN-LAB ?	-	COMP20081 1.1 BMAN20890 CRAW TH.2 2nd Yr Tutorial	EEEN20027 RENO C009 MATH20111 ZOCHONIS TH.B (G.7)
17:00	-	CARS20021 UNI PL B	-	BMAN20890 CRAW TH.2	-
Notes	†BMAN20880 weeks 8,9 & 10				

Constraints on Solutions

Registration Week Timetables

Year 1

- All First Years
- All Single Hons (+CBA/IC) A+W+X+Y+Z
- All Single Hons (-CBA/IC) W+X+Y+Z
- Group A - (CBA + IC)
- Group B - (CSwBM: C+D)
- Group C - (CSwBM)
- Group D - (CSwBM)
- Group E - (CSE)
- Group M - (CM)
- Group W - (CS,SE,DC,AI)
- Group X - (CS,SE,DC,AI)
- Group Y - (CS,SE,DC,AI)
- Group Z - (CS,SE,DC,AI)
- Lab grouping A+Z
- Lab grouping C+X
- Lab grouping D+E+Y
- Lab grouping D+Y
- Lab grouping M+W
- Service Units
- Taking BMAN courseunits A+B

Year 2

- All Second Year
- Joint Hons (CM)
- Joint Hons (CSE)
- Joint Hons (CSwBM)
- Lab Group F
- Lab Group G
- Lab Group H
- Lab Group I
- Single Hons (CBA)
- Single Hons (CS, SE, DC, AI)

Year 3

- All Former SoI
- All Third Years
- Joint Hons (CM)
- Joint Hons (CSwBM)
- Single Hons (CBA)
- Single Hons (Computer Science)
- Single Hons (Internet Computing)
- Single Hons (Software Engineering - Informatics)

Room Timetables

UG Teaching Rooms

G33	24 seats
Advisory	? seats
LF5	9 seats
LF6	9 seats
LF15	70 seats
LF17	27 seats
IT406	24 seats
IT407	100 seats

PG Teaching Rooms

2.19	100 seats
2.15	40 seats

UG Labs

Toot 1	40 seats
Toot 0	28 seats
Collab 2	4 Pods seats
Collab 1	8 Pods seats
PEVELab	? seats
G23	65 seats
3rdLab	61 seats
UNIX	70 seats

[All labs]

Meeting Rooms

1.20	? seats
2.33	15 seats
Atlas 1	28 seats
Atlas 2	22 seats
IT401	24 seats
Mercury	24 seats

Rooms should have a sufficient number of seats.

A teacher cannot teach two courses at the same time.

Andrei cannot teach at 9am.

State-changing systems

Our main interest from now on is modelling state-changing systems.

Informally	
At each time moment, the system is in a particular state.	
The system state is changing in time. There are actions (controlled or not) that change the state.	

State-changing systems

Our main interest from now on is modelling state-changing systems.

Informally	Formally
At each time moment, the system is in a particular state.	This state can be characterised by values of some variables, called the state variables.
The system state is changing in time. There are actions (controlled or not) that change the state.	Actions change values of some state variables.

Examples

- Reactive systems.
  Reactive systems are systems whose role is to maintain an ongoing interaction with their environment rather than produce some final value upon termination. Typical examples of reactive systems are air traffic control system, programs controlling mechanical devices such as a train, a plane, or ongoing processes such as a nuclear reactor.

Examples

- Reactive systems.
  Reactive systems are systems whose role is to maintain an ongoing interaction with their environment rather than produce some final value upon termination. Typical examples of reactive systems are air traffic control system, programs controlling mechanical devices such as a train, a plane, or ongoing processes such as a nuclear reactor.
- Concurrent systems.
  Concurrency is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other. A typical example is a computer operating system.

Reasoning about state-changing systems

1. Build a formal model of this state-changing system which describes, in particular, functioning of the system, or some abstraction thereof.

Reasoning about state-changing systems

1. Build a formal model of this state-changing system which describes, in particular, functioning of the system, or some abstraction thereof.
2. Use a logic to specify and verify properties of the system.

Propositional Logic of Finite Domains (PLFD)

Our first step to modelling state-changing systems is to introduce a logic in which we can express values of variables in state.

Propositional Logic of Finite Domains (PLFD)

Our first step to modelling state-changing systems is to introduce a logic in which we can express values of variables in state.

PLFD is a family of logics. Each instance of PLFD is characterised by

- a set X of variables;
- a mapping dom, such that for every $x \in X$, $dom(x)$ is a non-empty finite set, called the domain for x.

Syntax of PLFD

Formulas

- If x is a variable and $v \in dom(x)$ is a value in the domain of x, then $x = v$ is a formula, also called atomic formula, or simply atom.

Syntax of PLFD

Formulas

- If x is a variable and $v \in dom(x)$ is a value in the domain of x, then $x = v$ is a formula, also called atomic formula, or simply atom.
- Other formulas are built from atomic formulas as in propositional logic, using the connectives $\top$, $\bot$, $\wedge$, $\vee$, $\neg$, $\rightarrow$, and $\leftrightarrow$.

Semantics

- Interpretation for a set of variables X is a mapping I from X to the set of values such that for all $x \in X$ we have $I(x) \in dom(x)$.

Semantics

- Interpretation for a set of variables X is a mapping I from X to the set of values such that for all $x \in X$ we have $I(x) \in dom(x)$.
- Extend interpretations to mappings from formulas to boolean values.
  1. $I(x = v) = 1$ if and only if $I(x) = v$.
  2. If A is not atomic, then as for propositional formulas.

Semantics

- Interpretation for a set of variables X is a mapping I from X to the set of values such that for all $x \in X$ we have $I(x) \in dom(x)$.
- Extend interpretations to mappings from formulas to boolean values.
  1. $I(x = v) = 1$ if and only if $I(x) = v$.
  2. If A is not atomic, then as for propositional formulas.
- The definitions of truth, models, validity, satisfiability, and equivalence are defined exactly as in propositional logic.

Example

Let a variable x range over the domain $\{a, b, c\}$, that is $dom(x) = \{a, b, c\}$. Then the following formula is valid:

$$\neg x = a \rightarrow x = b \vee x = c.$$

Example

Let a variable x range over the domain $\{a, b, c\}$, that is $dom(x) = \{a, b, c\}$. Then the following formula is valid:

$$\neg x = a \to x = b \lor x = c.$$

But if $dom(x) = \{a, b, c, d\}$, then this formula is not valid. Indeed,

$$\{x \mapsto d\} \not\models \neg x = a \to x = b \lor x = c.$$

Propositional Logic as PLFD

The domain for each variable is $\{0, 1\}$. Instead of atoms p use $p = 1$.

One can also use $p = 0$ for $\neg p$, since $p = 0$ is equivalent to $\neg(p = 1)$.

Propositional Logic as PLFD

The domain for each variable is $\{0, 1\}$. Instead of atoms p use $p = 1$.

One can also use $p = 0$ for $\neg p$, since $p = 0$ is equivalent to $\neg(p = 1)$.

This transformation preserves models. For example, models of

$$p \wedge q \rightarrow \neg r$$

are exactly the models of

$$p = 1 \wedge q = 1 \rightarrow r = 0.$$

Propositional variables in PLFD

We say that p is a boolean variable if $dom(p) = \{0, 1\}$.

When we have an instance of PLFD where both boolean and non-boolean variables are used, we will use boolean variables as in propositional logic:

- p instead of $p = 1$;
- $\neg p$ instead of $p = 0$.

Translation of PLFD into Propositional Logic

- Introduce a propositional variable x_v for each variable x and value $v \in dom(x)$.
- Replace every atom $x = v$ by x_v;
- Add domain axiom for each variable x:

$$(x_{v_1} \vee \ldots \vee x_{v_n}) \wedge \bigwedge_{i<j} (\neg x_{v_i} \vee \neg x_{v_j}),$$

where $dom(x) = \{v_1, \ldots, v_n\}$.

Example

Let x range over the domain $\{a, b, c\}$. To check satisfiability of the following formula

$$\neg(x = b \vee x = c).$$

we have to check satisfiability of the set of formulas

$$(x_a \vee x_b \vee x_c) \wedge (\neg x_a \vee \neg x_b) \wedge (\neg x_a \vee \neg x_c) \wedge (\neg x_b \vee \neg x_c) \wedge$$
$$\neg(x_b \vee x_c).$$

Preservation of models

Suppose that I is a propositional model of all the domain axioms. Define a PLFD interpretation I' as follows:

$$I'(x) = v \stackrel{\text{def}}{=} I \models x_v .$$

Theorem

Let F' be a PLFD formula and F be obtained by translating F' to propositional logic. If $I \models F$, then $I' \models F'$.

Real-life modelling

Formalisation of numerous arguments used in 2003

The arguments used the following propositional variables.

1. can_start_war: one can start a war against Iraq;
2. is_guilty: Iraq is guilty;
3. has_WMD: Iraq has weapons of mass destruction.

Formalisation in propositional logic

If Iraq has weapons of mass destruction, then it is guilty.

$\text{has\_WMD} \rightarrow \text{is\_guilty}$

Formalisation in propositional logic

If Iraq has weapons of mass destruction, then it is guilty.

$\text{has\_WMD} \rightarrow \text{is\_guilty}$

If Iraq has no weapons of mass destruction, we cannot start a war.

$\neg\text{has\_WMD} \rightarrow \neg\text{can\_start\_war}$

Formalisation in propositional logic

If Iraq has weapons of mass destruction, then it is guilty.

$\text{has\_WMD} \rightarrow \text{is\_guilty}$

If Iraq has no weapons of mass destruction, we cannot start a war.

$\neg\text{has\_WMD} \rightarrow \neg\text{can\_start\_war}$

We want to check whether, under the above assumptions, it is possible that a war started against a country that is not guilty.

can_start_war

$\neg\text{is\_guilty}$

Formalisation in propositional logic

If Iraq has weapons of mass destruction, then it is guilty.	$\text{has\_WMD} \rightarrow \text{is\_guilty}$
If Iraq has no weapons of mass destruction, we cannot start a war.	$\neg\text{has\_WMD} \rightarrow \neg\text{can\_start\_war}$
We want to check whether, under the above assumptions, it is possible that a war started against a country that is not guilty.	can_start_war $\neg\text{is\_guilty}$

This set of formulas is unsatisfiable

Add a third value to a variable

At the UN, Colin Powell holds a model vial of anthrax, while arguing that Iraq is likely to possess WMDs (5 February 2003)

Add a third value to a variable

At the UN, Colin Powell holds a model vial of anthrax, while arguing that Iraq is likely to possess WMDs (5 February 2003)

Now let us consider a slightly different situation, when the domain of the variable has_WMD consists of the values *yes*, *no*, and a third value, for example, *suspected*.

Formalisation in propositional logic of finite domains

If Iraq has weapons of mass destruction, then it is guilty.	has_WMD $=$ *yes* $\rightarrow$ is_guilty

Formalisation in propositional logic of finite domains

If Iraq has weapons of mass destruction, then it is guilty.	$\text{has\_WMD} = \textit{yes} \rightarrow \text{is\_guilty}$
If Iraq has no weapons of mass destruction, we cannot start a war.	$\text{has\_WMD} = \textit{no} \rightarrow \neg\text{can\_start\_war}$

Formalisation in propositional logic of finite domains

If Iraq has weapons of mass destruction, then it is guilty.	$\mathsf{has\_WMD} = \mathit{yes} \rightarrow \mathsf{is\_guilty}$
If Iraq has no weapons of mass destruction, we cannot start a war.	$\mathsf{has\_WMD} = \mathit{no} \rightarrow \neg\mathsf{can\_start\_war}$
We want to check whether, under the above assumptions, it is possible that a war started against a country that is not guilty.	$\mathsf{can\_start\_war}$ $\neg\mathsf{is\_guilty}$

Translation to Propositional Logic

$\text{has\_WMD}_{yes} \rightarrow \text{is\_guilty}$
$\text{has\_WMD}_{no} \rightarrow \neg\text{can\_start\_war}$
can_start_war
$\neg\text{is\_guilty}$
$\text{has\_WMD}_{yes} \vee \text{has\_WMD}_{no}$
$\vee\text{has\_WMD}_{suspected}$
$\neg\text{has\_WMD}_{yes} \vee \neg\text{has\_WMD}_{no}$
$\neg\text{has\_WMD}_{yes} \vee \neg\text{has\_WMD}_{suspected}$
$\neg\text{has\_WMD}_{no} \vee \neg\text{has\_WMD}_{suspected}$

Translation to Propositional Logic

$\text{has\_WMD}_{yes} \rightarrow \text{is\_guilty}$
$\text{has\_WMD}_{no} \rightarrow \neg\text{can\_start\_war}$
can_start_war
$\neg\text{is\_guilty}$
$\text{has\_WMD}_{yes} \lor \text{has\_WMD}_{no}$
$\lor \text{has\_WMD}_{suspected}$
$\neg\text{has\_WMD}_{yes} \lor \neg\text{has\_WMD}_{no}$
$\neg\text{has\_WMD}_{yes} \lor \neg\text{has\_WMD}_{suspected}$
$\neg\text{has\_WMD}_{no} \lor \neg\text{has\_WMD}_{suspected}$

This set is satisfiable. Satisfiability can be established by unit propagation.

Translation to Propositional Logic

$\text{has\_WMD}_{yes} \rightarrow \text{is\_guilty}$
$\text{has\_WMD}_{no} \rightarrow \neg\text{can\_start\_war}$
can_start_war
$\neg\text{is\_guilty}$
$\text{has\_WMD}_{yes} \lor \text{has\_WMD}_{no} \lor \text{has\_WMD}_{suspected}$
$\neg\text{has\_WMD}_{yes} \lor \neg\text{has\_WMD}_{no}$
$\neg\text{has\_WMD}_{yes} \lor \neg\text{has\_WMD}_{suspected}$
$\neg\text{has\_WMD}_{no} \lor \neg\text{has\_WMD}_{suspected}$

This set is satisfiable. Satisfiability can be established by unit propagation.

Translating the propositional model to a model of the original problem gives

Translation to Propositional Logic

$$\begin{aligned}
&\text{has_WMD}_{yes} \rightarrow \text{is_guilty}\\
&\text{has_WMD}_{no} \rightarrow \neg\text{can_start_war}\\
&\text{can_start_war}\\
&\neg\text{is_guilty}\\
&\text{has_WMD}_{yes} \vee \text{has_WMD}_{no}\\
&\qquad\qquad \vee\text{has_WMD}_{suspected}\\
&\neg\text{has_WMD}_{yes} \vee \neg\text{has_WMD}_{no}\\
&\neg\text{has_WMD}_{yes} \vee \neg\text{has_WMD}_{suspected}\\
&\neg\text{has_WMD}_{no} \vee \neg\text{has_WMD}_{suspected}
\end{aligned}$$

This set is satisfiable. Satisfiability can be established by unit propagation.

Translating the propositional model to a model of the original problem gives

$$\begin{aligned}
\{&\text{can_start_war} \mapsto 1,\\
&\text{is_guilty} \mapsto 0,\\
&\text{has_WMD} \mapsto \textit{suspected}\}
\end{aligned}$$

Translation to Propositional Logic

These Weapons of Mass Destruction cannot be displayed

The weapons you are looking for are currently unavailable. The country might be experiencing technical difficulties, or you may need to adjust your weapons inspectors mandate.

Please try the following:

- Click the Regime change button, or try again later.
- If you are George Bush and typed the country's name in the address bar, make sure that it is spelled correctly. (IRAQ).
- To check your weapons inspector settings, click the **UN** menu, and then click **Weapons Inspector Options**. On the **Security Council** tab, click **Consensus**. The settings should match those provided by your government or NATO.
- If the Security Council has enabled it, The United States of America can examine your country and automatically discover Weapons of Mass Destruction.
  If you would like to use the CIA to try and discover them, click Detect weapons
- Some countries require 128 thousand troops to liberate them. Click the **Panic** menu and then click **About US foreign policy** to determine what regime they will install.
- If you are an Old European Country trying to protect your interests, make sure your options are left wide open as long as possible. Click the **Tools** menu, and then click on **League of Nations**. On the Advanced tab, scroll to the Head in the Sand section and check settings for your exports to Iraq.
- Click the Bomb button if you are Donald Rumsfeld.

$$\{\text{can\_start\_war} \mapsto 1,\ \text{is\_guilty} \mapsto 0,\ \text{has\_WMD} \mapsto \textit{suspected}\}$$