Logic and Modelling

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 and then translated to propositional logic . . .

because propositional logic is usually not convenient for modelling but convenient for reasoning.

Logic and Modelling

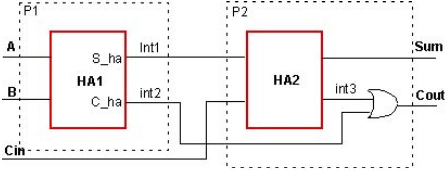
Satisfiability-checking in propositional logic has many applications.

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 and then translated to propositional logic . . .

because propositional logic is usually not convenient for modelling but convenient for reasoning.

Circuit Design

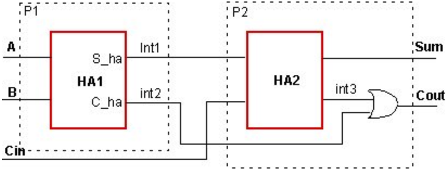


Circuit: propositional logic

```
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)

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.

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 characterized 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:

- A program state is defined by assigning values to all variables.
- A digital circuit state is defined by assigning values to the gates, clocks.
- Engineering devices: microwave ovens, vending machines, traffic light controllers.

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 characterized 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:

- A program state is defined by assigning values to all variables.
- A digital circuit state is defined by assigning values to the gates, clocks.
- Engineering devices: microwave ovens, vending machines, traffic light controllers.

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 characterized 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:

- A program state is defined by assigning values to all variables.
- A digital circuit state is defined by assigning values to the gates, clocks.
- Engineering devices: microwave ovens, vending machines, traffic light controllers.

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.

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.

Microwave Reasoning

variable	domain of values
mode	{*idle*, *cooking*, *defrost*}
door	{*open*, *closed*}
content	{*none*, *burger*, *pizza*, *cabbage*}
user	{*nobody*, *student*, *vegetarian*}

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 characterized 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.

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 characterized 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 of PLFD:

- 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.

  Example: $user = student$

- Other formulas are built from atomic formulas as in propositional logic, using the connectives $\top$, $\bot$, $\wedge$, $\vee$, $\neg$, $\rightarrow$, and $\leftrightarrow$.

  Example: $mode = cooking \rightarrow door = closed \wedge \neg user = nobody$

Syntax of PLFD

Formulas of PLFD:

- 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.

  Example: $user = student$

- Other formulas are built from atomic formulas as in propositional logic, using the connectives $\top$, $\bot$, $\wedge$, $\vee$, $\neg$, $\rightarrow$, and $\leftrightarrow$.

  Example: $mode = cooking \rightarrow door = closed \wedge \neg user = nobody$

Semantics

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

  Example: $I = \{mode \mapsto cooking, door \mapsto closed, content \mapsto pizza, user \mapsto student\}$

- Evaluate formula in an interpretation:
  - $I(x = v) = 1$ if and only if $I(x) = v$.
  - 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.

Semantics

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

  Example: $I = \{mode \mapsto cooking, door \mapsto closed,$
  $content \mapsto pizza, user \mapsto student\}$

- Evaluate formula in an interpretation:
  - $I(x = v) = 1$ if and only if $I(x) = v$.
  - 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.

Semantics

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

  Example: $I = \{mode \mapsto cooking, door \mapsto closed,$ $content \mapsto pizza, user \mapsto student\}$

- Evaluate formula in an interpretation:
  - $I(x = v) = 1$ if and only if $I(x) = v$.
  - 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: for I above we have:
  $I \models mode = cooking \rightarrow door = closed \wedge \neg user = nobody$

Example

Let a variable x range over the domain $\{0, 1, 2\}$, that is $dom(x) = \{0, 1, 2\}$. Then the following formula is valid:

$$\neg x = 0 \rightarrow x = 1 \vee x = 2.$$

But the following formula is not valid:

$$x = 0 \vee x = 1$$

Propositional Logic as PLFD

Now we translate propositional logic to PLFD.
The domain for each variable is $\{0, 1\}$.

Instead of atoms p use $p = 1$.

Example:
Propositional formula $\neg p \rightarrow q$.
PLFD formula $\neg p = 1 \rightarrow q = 1$.

Propositional Logic as PLFD

Now we translate propositional logic to PLFD.
The domain for each variable is $\{0, 1\}$.

Instead of atoms p use $p = 1$.

Example:
Propositional formula $\neg p \rightarrow q$.
PLFD formula $\neg p = 1 \rightarrow q = 1$.

Notation: When p is a Boolean variable, that is, $dom(p) = \{0, 1\}$, in PLFD we will write p instead of $p = 1$ and $\neg p$ instead of $p = 0$.

Example: In place of $(mode = idle \leftrightarrow \neg p = 1) \wedge p = 0$ we write $(mode = idle \leftrightarrow \neg p) \wedge \neg p$

Translation of PLFD into Propositional Logic

Consider a PLFD formula F.

- Introduce a propositional variable x_v for each variable x and value $v \in dom(x)$.
- Replace every atom $x = v$ by x_v in F obtaining a propositional formula F_{prop}.
- Domain axiom D_x for a 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 $dom(x) = \{v_1, \ldots, v_n\}$.
- Let D_{all} denote the conjunction of all domain axioms:

$$D_{all} = D_{x_1} \wedge \ldots \wedge D_{x_k}$$

- Then, PLFD formula F is satisfiable if and only if propositional formula $F_{prop} \wedge D_{all}$ is satisfiable.

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

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

Tableaux algorithm for PLFD

Signed Formulas

The tableau algorithm works on signed formulas:

- Signed formula: an expression $A = b$, where A is a formula and b a Boolean value.
- A signed formula $A = b$ is true in an interpretation I, denoted by $I \models A = b$, if $I(A) = b$.
- A formula A is satisfiable if and only if so is $A = 1$.

- Use new kind of atomic formulas $x \in \{v_1, \dots, v_n\}$, replace $x = a$ by $x \in \{a\}$.
- Abbreviations: instead of $(x \in D) = 1$ write $x \in D$, instead of $(x \in D) = 0$ write $x \notin D$.

Branch Expansion Rules

$$
\begin{array}{rcl}
(A_1 \wedge \ldots \wedge A_n) = 0 & \rightsquigarrow & A_1 = 0 \mid \ldots \mid A_n = 0 \\
(A_1 \wedge \ldots \wedge A_n) = 1 & \rightsquigarrow & A_1 = 1, \ldots, A_n = 1 \\
(A_1 \vee \ldots \vee A_n) = 0 & \rightsquigarrow & A_1 = 0, \ldots, A_n = 0 \\
(A_1 \vee \ldots \vee A_n) = 1 & \rightsquigarrow & A_1 = 1 \mid \ldots \mid A_n = 1 \\
(A_1 \rightarrow A_2) = 0 & \rightsquigarrow & A_1 = 1, A_2 = 0 \\
(A_1 \rightarrow A_2) = 1 & \rightsquigarrow & A_1 = 0 \mid A_2 = 1 \\
(\neg A_1) = 0 & \rightsquigarrow & A_1 = 1 \\
(\neg A_1) = 1 & \rightsquigarrow & A_1 = 0 \\
(A_1 \leftrightarrow A_2) = 0 & \rightsquigarrow & A_1 = 0, A_2 = 1 \mid A_1 = 1, A_2 = 0 \\
(A_1 \leftrightarrow A_2) = 1 & \rightsquigarrow & A_1 = 0, A_2 = 0 \mid A_1 = 1, A_2 = 1 \\
x \notin D & \rightsquigarrow & x \in dom(x) \setminus D \\
x \in D_1,\ x \in D_2 & \rightsquigarrow & x \in D_1 \cap D_2
\end{array}
$$

Branch Closure Rules

Building tableau:

- choose a branch and a formula in it
- expand tableau applying rules above to the selected formula

Branch closure:

- A branch is closed if it contains $x \in \{\}$.
- A branch is open if it is not closed.

Terminated tableau: if either

- All branches are closed.
  Then the signed formula is unsatisfiable.
- All rules have been applied on an open branch.
  Then the tableau is satisfiable.
- How to check validity?

Branch Closure Rules

Building tableau:

- choose a branch and a formula in it
- expand tableau applying rules above to the selected formula

Branch closure:

- A branch is closed if it contains $x \in \{\}$.
- A branch is open if it is not closed.

Terminated tableau: if either

- All branches are closed.
  Then the signed formula is unsatisfiable.
- All rules have been applied on an open branch.
  Then the tableau is satisfiable.
- How to check validity?
  A is valid if and only if $A = 0$ is unsatisfiable

$\neg(\mathit{mode} \in \{\mathit{idle}\} \vee \neg\mathit{mode} \in \{\mathit{cooking}\} \rightarrow \mathit{mode} \in \{\mathit{idle}\})$ sat?

(a) $\neg(\mathit{mode} \in \{\mathit{idle}\} \vee \neg\mathit{mode} \in \{\mathit{cooking}\} \rightarrow \mathit{mode} \in \{\mathit{idle}\}) = 1$

$\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \to mode \in \{idle\})$ sat?

$$\text{(a)} \quad \neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \to mode \in \{idle\}) = 1$$

(a)

$$\text{(b)} \quad (mode \in \{idle\} \vee \neg mode \in \{cooking\} \to mode \in \{idle\}) = 0$$

$\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\})$ sat?

(a) $\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 1$

(a)

(b) $(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 0$

(b)

(c) $(mode \in \{idle\} \vee \neg mode \in \{cooking\}) = 1$

(d) $mode \notin \{idle\}$

$\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\})$ sat?

(a) $\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 1$

(a)

(b) $(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 0$

(b)

(c) $(mode \in \{idle\} \vee \neg mode \in \{cooking\}) = 1$

(d) $mode \notin \{idle\}$

(d)

(e) $mode \in \{cooking, defrost\}$

$\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\})$ sat?

(a) $\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 1$

(a)

(b) $(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 0$

(b)

(c) $(mode \in \{idle\} \vee \neg mode \in \{cooking\}) = 1$

(d) $mode \notin \{idle\}$

(d)

(e) $mode \in \{cooking, defrost\}$

(c) (c)

(f) $mode \in \{idle\}$

$mode \notin \{cooking\}$ (g)

$\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\})$ sat?

(a) $\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 1$

(a)

(b) $(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 0$

(b)

(c) $(mode \in \{idle\} \vee \neg mode \in \{cooking\}) = 1$

(d) $mode \notin \{idle\}$

(d)

(e) $mode \in \{cooking, defrost\}$

(c) (c)

(f) $mode \in \{idle\}$ $mode \notin \{cooking\}$ (g)

(e,f)

$mode \in \{\}$

closed

$\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\})$ sat?

(a) $\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 1$

(a)

(b) $(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 0$

(b)

(c) $(mode \in \{idle\} \vee \neg mode \in \{cooking\}) = 1$

(d) $mode \notin \{idle\}$

(d)

(e) $mode \in \{cooking, defrost\}$

(c)	(c)
(f) $mode \in \{idle\}$	$mode \notin \{cooking\}$ (g)
(e,f)	(g)
$mode \in \{\}$	$mode \in \{idle, defrost\}$ (h)
closed

$\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\})$ sat?

(a) $\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 1$

(a)

(b) $(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 0$

(b)

(c) $(mode \in \{idle\} \vee \neg mode \in \{cooking\}) = 1$

(d) $mode \notin \{idle\}$

(d)

(e) $mode \in \{cooking, defrost\}$

(c)	(c)
(f) $mode \in \{idle\}$	$mode \notin \{cooking\}$ (g)
(e,f)	(g)
$mode \in \{\}$ *closed*	$mode \in \{idle, defrost\}$ (h)
	(e,h)
	$mode \in \{defrost\}$ (i)

$\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\})$ sat?

(a) $\neg(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 1$

(a)

(b) $(mode \in \{idle\} \vee \neg mode \in \{cooking\} \rightarrow mode \in \{idle\}) = 0$

(b)

(c) $(mode \in \{idle\} \vee \neg mode \in \{cooking\}) = 1$

(d) $mode \notin \{idle\}$

(d)

(e) $mode \in \{cooking, defrost\}$

(c) (c)

(f) $mode \in \{idle\}$ $\qquad$ $mode \notin \{cooking\}$ (g)

(e,f) $\qquad$ (g)

$mode \in \{\}$ $\qquad$ $mode \in \{idle, defrost\}$ (h)

closed

(e,h)

$mode \in \{defrost\}$ (i)

The formula is satisfiable and the model is $I = \{mode \mapsto defrost\}$

Summary

Propositional logic of finite domains (PLFD) is useful for describing properties of states in state changing systems.

We have studied:

- translations of propositional logic to PLFD and PLFD to propositional logic.
- a tableau algorithm for checking satisfiability of PLFD formulas.