Propositional Logic of Finite Domains
  Logic and Modelling
  State-changing systems
  PLFD
  PLFD and propositional logic
Logic and Modelling

Satisfiability-checking in propositional logic has many applications.
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
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;
## Scheduling

### All Second Year Timetable 2009-2010

<table>
<thead>
<tr>
<th>Printable</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>08:00</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>09:00</strong></td>
<td>MATH20701 CRAW TH.1 COMP20051 1.1</td>
<td>GCOMP20340(A) GCOMP20340(A)</td>
<td>GCOMP20411(A) GCOMP20010 G23</td>
<td>GCOMP20411(A) GCOMP20010 UNIX</td>
<td>GCOMP20340(B) UNIX</td>
</tr>
<tr>
<td></td>
<td>MATH20701 Minds Coop G20</td>
<td>GCOMP20340(A) IT407</td>
<td>GCOMP20010 UNIX IT407</td>
<td>GCOMP20010 UNIX IT407</td>
<td>GCOMP20340(B) UNIX</td>
</tr>
<tr>
<td><strong>10:00</strong></td>
<td>BMAN20870 SIMON B (B.41) COMP20340</td>
<td>BMAN21061 CRAW TH.1 COMP20010 G23</td>
<td>BMAN10621 GCOMP20411(A) GCOMP20010</td>
<td>BMAN10621 ROSCOE 1.007</td>
<td>BMAN20870 SIMON B (B.41) COMP20340</td>
</tr>
<tr>
<td></td>
<td>MATH20701 Minds Coop G20</td>
<td>BMAN21061 CRAW TH.1 COMP20010 G23</td>
<td>BMAN10621 GCOMP20411(A) GCOMP20010</td>
<td>BMAN10621 ROSCOE 1.007</td>
<td>MATH20701 RENO C016</td>
</tr>
<tr>
<td><strong>11:00</strong></td>
<td>BMAN20871 MBS EAST B8 MAT29631 SIMON 3</td>
<td>BMAN10621 CRAW TH.1 COMP20010 G23</td>
<td>COMP20081(A) GCOMP20010 G23</td>
<td>GCOMP20051(A) GCOMP20010 G23</td>
<td>BMAN20871 MBS EAST B8 MAT29631 SIMON 3</td>
</tr>
<tr>
<td></td>
<td>MATH20701 Minds Coop G20</td>
<td>BMAN21061 CRAW TH.1 COMP20010 G23</td>
<td>COMP20081(A) GCOMP20010 G23</td>
<td>GCOMP20051(A) GCOMP20010 G23</td>
<td>MATH20701 RENO C016</td>
</tr>
<tr>
<td><strong>12:00</strong></td>
<td>BMAN21061 ROSCOE 1.008 EEE20019 RENO C002</td>
<td>COMP-PASS LF15 Math20411 Turing G.107</td>
<td>COMP-PASS LF15 Math20411 Turing G.107</td>
<td>gCOMP20010 RENO C002 Math20111 Turing G.107</td>
<td>BMAN21061 ROSCOE 1.008 EEE20019 RENO C002</td>
</tr>
<tr>
<td></td>
<td>MATH20411 Turing G.107</td>
<td>MATH20411 Turing G.107</td>
<td>gCOMP20010 RENO C002 Math20111 Turing G.107</td>
<td>BMAN21061 ROSCOE 1.008 EEE20019 RENO C002</td>
<td>MATH20411 Turing G.107</td>
</tr>
<tr>
<td><strong>13:00</strong></td>
<td>COMP20340(A) IT407 COMP20340(B)</td>
<td>COMP20411 1.1 COMP20411</td>
<td>COMP20411 1.1 COMP20411</td>
<td>COMP20411 1.1 COMP20411</td>
<td>COMP20340(A) IT407 COMP20340(B)</td>
</tr>
<tr>
<td></td>
<td>COMP20081(A) G23 COMP20051(A)</td>
<td>COMP20411 1.1 COMP20411</td>
<td>COMP20411 1.1 COMP20411</td>
<td>COMP20411 1.1 COMP20411</td>
<td>COMP20081(A) G23 COMP20051(A)</td>
</tr>
<tr>
<td></td>
<td>MATH20411 Turing G.107</td>
<td>COMP20411 1.1 COMP20411</td>
<td>COMP20411 1.1 COMP20411</td>
<td>COMP20411 1.1 COMP20411</td>
<td>MATH20411 Turing G.107</td>
</tr>
<tr>
<td><strong>14:00</strong></td>
<td>BMAN20880 SIMON 3 (3.40) EEE20019 RENO C009</td>
<td>EEE-LAB COMP20411 1.1</td>
<td>BMAN21061 CRAW TH.2 COMP20141 1.1</td>
<td>COMP20141 1.1 COMP20010</td>
<td>EEE20019 SSB A16 COMP20141 1.1</td>
</tr>
<tr>
<td></td>
<td>MATH20111 Turing G.207</td>
<td>COMP20411 1.1 COMP20411</td>
<td>BMAN21061 CRAW TH.2 COMP20141 1.1</td>
<td>COMP20141 1.1 COMP20010</td>
<td>EEE20019 SSB A16 COMP20141 1.1</td>
</tr>
<tr>
<td></td>
<td>COMP20340(A) IT407 COMP20340(B)</td>
<td>COMP20411 1.1 COMP20411</td>
<td>BMAN21061 CRAW TH.2 COMP20141 1.1</td>
<td>COMP20141 1.1 COMP20010</td>
<td>EEE20019 SSB A16 COMP20141 1.1</td>
</tr>
<tr>
<td></td>
<td>COMP20081(A) G23 COMP20051(A)</td>
<td>COMP20411 1.1 COMP20411</td>
<td>BMAN21061 CRAW TH.2 COMP20141 1.1</td>
<td>COMP20141 1.1 COMP20010</td>
<td>EEE20019 SSB A16 COMP20141 1.1</td>
</tr>
<tr>
<td></td>
<td>MATH20411 Turing G.107</td>
<td>COMP20411 1.1 COMP20411</td>
<td>BMAN21061 CRAW TH.2 COMP20141 1.1</td>
<td>COMP20141 1.1 COMP20010</td>
<td>EEE20019 SSB A16 COMP20141 1.1</td>
</tr>
<tr>
<td><strong>15:00</strong></td>
<td>COMP20051(A) G23 COMP20010</td>
<td>COMP20051(A) G23 COMP20010</td>
<td>COMP20051(A) G23 COMP20010</td>
<td>COMP20051(A) G23 COMP20010</td>
<td>COMP20051(A) G23 COMP20010</td>
</tr>
<tr>
<td></td>
<td>BMAN20880 SIMON 3 (3.40)</td>
<td>COMP20051(A) G23 COMP20010</td>
<td>COMP20051(A) G23 COMP20010</td>
<td>COMP20051(A) G23 COMP20010</td>
<td>COMP20051(A) G23 COMP20010</td>
</tr>
<tr>
<td></td>
<td>2nd Yr Tutorial</td>
<td>2nd Yr Tutorial</td>
<td>2nd Yr Tutorial</td>
<td>2nd Yr Tutorial</td>
<td>2nd Yr Tutorial</td>
</tr>
<tr>
<td><strong>16:00</strong></td>
<td>MATH20111 Turing G.207</td>
<td>MATH20411 Turing G.107</td>
<td>MATH20111 Turing G.207</td>
<td>MATH20111 Turing G.207</td>
<td>MATH20111 Turing G.207</td>
</tr>
<tr>
<td></td>
<td>MATH20411 Turing G.107</td>
<td>MATH20411 Turing G.107</td>
<td>MATH20411 Turing G.107</td>
<td>MATH20411 Turing G.107</td>
<td>MATH20411 Turing G.107</td>
</tr>
<tr>
<td></td>
<td>MATH20411 Turing G.107</td>
<td>MATH20411 Turing G.107</td>
<td>MATH20411 Turing G.107</td>
<td>MATH20411 Turing G.107</td>
<td>MATH20411 Turing G.107</td>
</tr>
<tr>
<td><strong>17:00</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Notes**

- BMAN20880 weeks 8,9 & 10
### Constraints on Solutions

#### 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)

---

**Rooms should have a sufficient number of seats.**

**A teacher cannot teach two courses at the same time.**

**Andrei cannot teach at 9am.**
Our main interest from now on is modelling state-changing systems.

<table>
<thead>
<tr>
<th>Informally</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>At each time moment, the system is in a particular state.</td>
<td></td>
</tr>
<tr>
<td>The system state is changing in time. There are actions (controlled or not) that change the state.</td>
<td></td>
</tr>
</tbody>
</table>
Our main interest from now on is modelling state-changing systems.

<table>
<thead>
<tr>
<th>Informally</th>
<th>Formally</th>
</tr>
</thead>
<tbody>
<tr>
<td>At each time moment, the system is in a particular state.</td>
<td>This state can be characterised by values of some variables, called the state variables.</td>
</tr>
<tr>
<td>The system state is changing in time. There are actions (controlled or not) that change the state.</td>
<td>Actions change values of some state variables.</td>
</tr>
</tbody>
</table>
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.
Our first step to modelling state-changing systems is to introduce a logic in which we can *express values of variables* in state.
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 $v \in \text{dom}(x)$ 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 $v \in \text{dom}(x)$ 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 $\top$, $\bot$, $\land$, $\lor$, $\neg$, $\rightarrow$, 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 $I(x) \in \text{dom}(x)$.

- 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.
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 $I(x) \in \text{dom}(x)$.
- 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.
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 $I(x) \in \text{dom}(x)$.

- 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 $\text{dom}(x) = \{a, b, c\}$. Then the following formula is valid:

$$\neg x = a \rightarrow x = b \lor x = c.$$
Example

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

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

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

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

\[
p \land q \rightarrow \neg r
\]

are exactly the models of

\[
p = 1 \land q = 1 \rightarrow r = 0.
\]
We say that $p$ is a boolean variable if $\text{dom}(p) = \{0, 1\}$.

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 $v \in \text{dom}(x)$.
- Replace every atom $x = v$ by $x_v$;
- Add domain axiom for each variable $x$:

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

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

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

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

we have to check satisfiability of the set of formulas

$$(x_a \lor x_b \lor x_c) \land (\neg x_a \lor \neg x_b) \land (\neg x_a \lor \neg x_c) \land (\neg x_b \lor \neg x_c) \land \neg (x_b \lor x_c).$$
Preservation of models

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

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

**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
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.

\[ \text{has}_W \text{MD} \rightarrow \text{is}_\text{guilty} \]
Formalisation in propositional logic

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

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

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

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

\[ \text{can\_start\_war} \]

\[ \neg\text{is\_guilty} \]
Formalisation in propositional logic

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

\[ \text{has}_\text{WMD} \rightarrow \text{is}_\text{guilty} \]

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

\[ \neg \text{has}_\text{WMD} \rightarrow \neg \text{can}_\text{start}_\text{war} \]

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

\[ \text{can}_\text{start}_\text{war} \rightarrow \neg \text{is}_\text{guilty} \]

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.

\[ \text{has\_WMD} = \text{yes} \rightarrow \text{is\_guilty} \]
Formalisation in propositional logic of finite domains

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

\[ \text{has}_{\text{WMD}} = \text{yes} \rightarrow \text{is}_{\text{guilty}} \]

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

\[ \text{has}_{\text{WMD}} = \text{no} \rightarrow \neg \text{can}_{\text{start}_{\text{war}}} \]
Formalisation in propositional logic of finite domains

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

\[ \text{has\_WMD} = \text{yes} \rightarrow \text{is\_guilty} \]

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

\[ \text{has\_WMD} = \text{no} \rightarrow \neg \text{can\_start\_war} \]

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

\[ \neg \text{can\_start\_war} \rightarrow \neg \text{is\_guilty} \]
Translation to Propositional Logic

has\_WMD_{yes} \rightarrow \text{is\_guilty}
has\_WMD_{no} \rightarrow \neg \text{can\_start\_war}
can\_start\_war
\neg \text{is\_guilty}
has\_WMD_{yes} \lor has\_WMD_{no} \\
\lor has\_WMD_{suspected}
\neg has\_WMD_{yes} \lor \neg has\_WMD_{no}
\neg has\_WMD_{yes} \lor \neg has\_WMD_{suspected}
\neg has\_WMD_{no} \lor \neg has\_WMD_{suspected}

This set is satisfiable. Satisfiability can be established by unit propagation.
Translation to Propositional Logic

\[\begin{align*}
\text{has}_W\text{MD}_{\text{yes}} & \rightarrow \text{is}_\text{guilty} \\
\text{has}_W\text{MD}_{\text{no}} & \rightarrow \neg\text{can}_\text{start}_\text{war} \\
\neg\text{is}_\text{guilty} \\
\text{has}_W\text{MD}_{\text{yes}} & \lor \text{has}_W\text{MD}_{\text{no}} \\
& \lor \text{has}_W\text{MD}_{\text{suspected}} \\
\neg\text{has}_W\text{MD}_{\text{yes}} & \lor \neg\text{has}_W\text{MD}_{\text{no}} \\
\neg\text{has}_W\text{MD}_{\text{yes}} & \lor \neg\text{has}_W\text{MD}_{\text{suspected}} \\
\neg\text{has}_W\text{MD}_{\text{no}} & \lor \neg\text{has}_W\text{MD}_{\text{suspected}}
\end{align*}\]

This set is satisfiable. Satisfiability can be established by unit propagation.
Translation to Propositional Logic

\[
\begin{align*}
\text{has}_\text{WMD}_{yes} & \rightarrow \text{is}_\text{guilty} \\
\text{has}_\text{WMD}_{no} & \rightarrow \neg\text{can}_\text{start}_\text{war} \\
\text{can}_\text{start}_\text{war} & \\
\neg\text{is}_\text{guilty} & \\
\neg\text{has}_\text{WMD}_{yes} & \lor \text{has}_\text{WMD}_{no} \\
\text{has}_\text{WMD}_{suspected} & \\
\neg\text{has}_\text{WMD}_{yes} & \lor \neg\text{has}_\text{WMD}_{no} \\
\neg\text{has}_\text{WMD}_{suspected} & \\
\neg\text{has}_\text{WMD}_{no} & \lor \neg\text{has}_\text{WMD}_{suspected}
\end{align*}
\]

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

\[\begin{align*}
\text{has	extunderscore WMD}_{yes} & \rightarrow \text{is	extunderscore guilty} \\
\text{has	extunderscore WMD}_{no} & \rightarrow \neg \text{can	extunderscore start	extunderscore war} \\
\neg \text{can	extunderscore start	extunderscore war} & \\
\neg \text{is	extunderscore guilty} \\
\text{has	extunderscore WMD}_{yes} & \vee \text{has	extunderscore WMD}_{no} \\
& \quad \vee \text{has	extunderscore WMD}_{suspected} \\
\neg \text{has	extunderscore WMD}_{yes} & \vee \neg \text{has	extunderscore WMD}_{no} \\
\neg \text{has	extunderscore WMD}_{yes} & \vee \neg \text{has	extunderscore WMD}_{suspected} \\
\neg \text{has	extunderscore WMD}_{no} & \vee \neg \text{has	extunderscore WMD}_{suspected}
\end{align*}\]

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

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

\{can\_start\_war \rightarrow 1, \\
is\_guilty \rightarrow 0, \\
\text{has\_WMD} \rightarrow \text{suspected}\}
Translation to Propositional Logic

{can_start_war \rightarrow 1,
 is_guilty \rightarrow 0,
 has_WMD \rightarrow \textit{suspected}}