Tufts
UNIVERSITY

PROPOSITIONAL LOGIC 2

Fabrizio Santini | COMP 131

VERSION 2.0

TODAY ON AI

- Automated reasoning
- Efficient satisfability
- Questions?

Automated reasoning

SECTION 01

Logical inference is used to create new sentences that logically follow from a given knowledge base.

- The most used inference rules:

RULE	PREMISE	CONCLUSION
Modus Ponens	$p, p \rightarrow q$	q
AND elimination	$p \wedge q$	p, q
Double negation	$\neg\neg p$	p
Unit resolution	$p \vee q, \neg q$	p
AND introduction	p, q	$p \wedge q$
Modus Tollens	$\neg q, p \rightarrow q$	$\neg p$

- There are two directions of search: **forward** and **backward** chaining.

KNOWLEDGE BASE

RULE	SYMBOL
1	$PersonInFrontOfCar \rightarrow Brake$
2	$((YellowLight \vee Policeman) \wedge \neg Slippery) \rightarrow Brake$
3	$Policecar \rightarrow Policeman$
4	$Snow \rightarrow Slippery$
5	$Slippery \rightarrow \neg Dry$
6	$RedLight \rightarrow Brake$
7	$Winter \rightarrow Snow$

FACTS

$YellowLight\ \neg RedLight \wedge \neg Snow \wedge Dry \wedge Policar \wedge \neg PersonInFrontOfCar$

Forward chaining: answer queries using a knowledge base to determine new facts until we find that our query is **true**, or until we've run out of new facts to generate.

QUERY

Do we need to $Brake$?

INFERENCE

KNOWN $\boldsymbol{Policecar}$
MP R3 $Policecar \rightarrow Policeman$
$\boldsymbol{Policeman}$

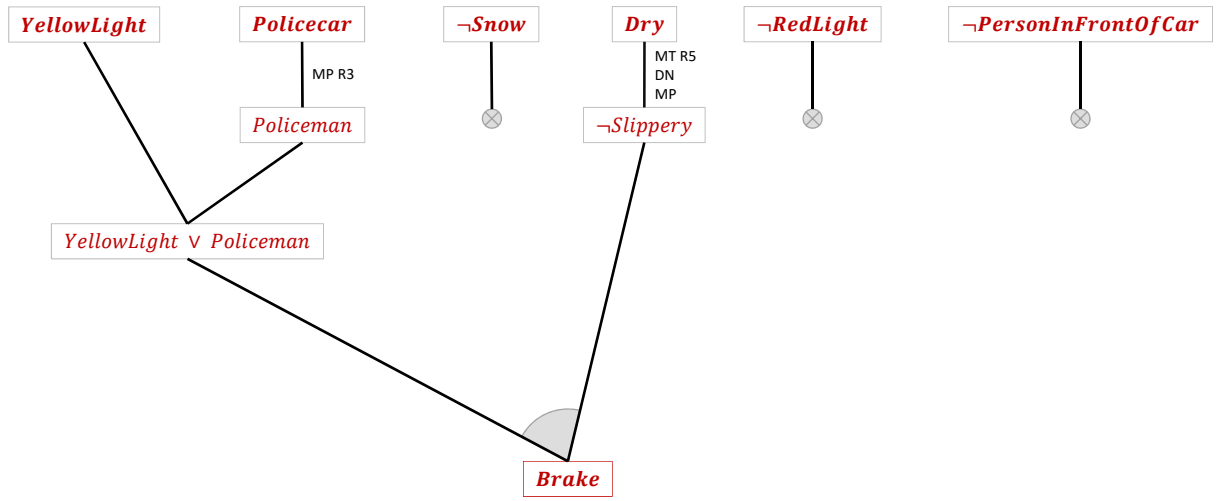
KNOWN $\boldsymbol{Dry}$
MT R5 $Slippery \rightarrow \neg Dry$
DN $\neg\neg Dry \rightarrow \neg Slippery$
MP $Dry \rightarrow \neg Slippery$
$\boldsymbol{\neg Slippery}$

KNOWN $\boldsymbol{YellowLight}$
KNOWN $\boldsymbol{Policeman}$
KNOWN $\boldsymbol{\neg Slippery}$
MP R2 $((YellowLight \vee Policeman) \wedge \neg Slippery) \rightarrow Brake$

C $\boxed{\boldsymbol{Brake}}$

Automated reasoning

FORWARD CHAINING WITH AND-OR TREES



KNOWLEDGE BASE

RULE	SYMBOL
1	$PersonInFrontOfCar \rightarrow Brake$
2	$((YellowLight \vee Policeman) \wedge \neg Slippery) \rightarrow Brake$
3	$Policecar \rightarrow Policeman$
4	$Snow \rightarrow Slippery$
5	$Slippery \rightarrow \neg Dry$
6	$RedLight \rightarrow Brake$
7	$Winter \rightarrow Snow$

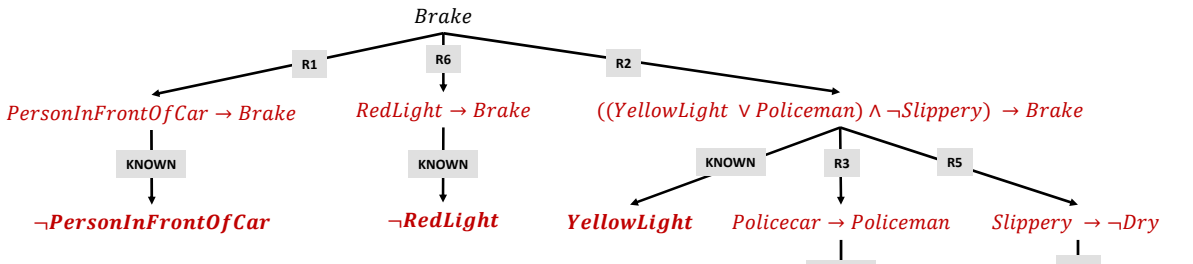
FACTS

$YellowLight\ \neg RedLight \wedge \neg Snow \wedge Dry \wedge Policar \wedge \neg PersonInFrontOfCar$

Backward chaining: an approach alternative to forward chaining in which the query is explicitly proven with the given knowledge, and work backward until all the facts are known.

QUERY

Do we need to $Brake$?



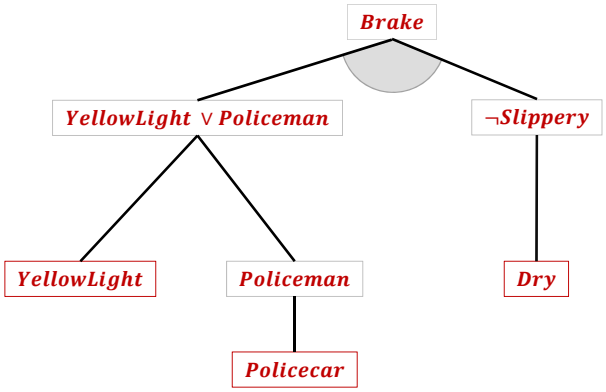
RULE	SYMBOL
~~1~~	~~$PersonInFrontOfCar \rightarrow Brake$~~
2	$((YellowLight \vee Policeman) \wedge \neg Slippery) \rightarrow Brake$
3	$Policecar \rightarrow Policeman$
4	$Snow \rightarrow Slippery$
5	$Slippery \rightarrow \neg Dry$
~~6~~	~~$RedLight \rightarrow Brake$~~
7	$Winter \rightarrow Snow$

QUERY

Do we need to $Brake$?

Automated reasoning

BACKWARD CHAINING WITH AND-OR TREES



Efficient satisfability

SECTION 02

- A sentence is in **conjunctive normal form** (or CNF) if it is a **disjunction of terms**
- Examples of CNF sentences: $(p \vee q \vee r \vee s)$, $(p \vee \neg q)$
- There is a way to convert a sentence into a clausal form:

SENTENCE	CLAUSAL FORM
$p \leftrightarrow q$	$p \rightarrow q$ $q \rightarrow p$
$p \rightarrow q$	$\neg p \vee q$
$\neg\neg p$	p
$\neg(p \vee q)$	$\neg p \wedge \neg q$
$\neg(p \wedge q)$	$\neg p \vee \neg q$

The **Davis-Putman-Logemann-Loveland** is a complete search algorithm for deciding if sentences are satisfiable.

- It uses Depth-First Search for backtracking
- DPLL requires that the knowledge base is represented in a CNF form
- It uses improvements to shorten the search:
  - **Early possible** termination
  - **Pure symbol** heuristic
  - **Unit clause** heuristic

DPLL(C, S, M):

1. If (every c $\in C$ is **T**) ∨ (C is empty),
   return **T**
2. If C contains an empty clause,
   return **F**
3. If there is a (t, polarity v) = **pure symbol**(C),
   return DPLL(C, $S - t$, $M \cup \{t = v\}$)
4. If there is a (u, polarity v) = **unit clause**(C),
   return DPLL(C, $S - u$, $M \cup \{u = v\}$)
5. P = **first**(S); R = **rest**(S);
6. Return
   DPLL(C, R, $M \cup \{P =$ **T**$\}$)
   ∨
   DPLL(C, R, $M \cup \{P =$ **F**$\}$)

DPLL(C, S, M):

1. If (every c $\in C$ is **T**) ∨ (C is empty),
   return **T**
2. If C contains an empty clause,
   return **F**
3. If there is a (t, polarity v) = **pure symbol**(C),
   return DPLL($C, S - t, M \cup \{t = v\}$)
4. If there is a (u, polarity v) = **unit clause**(C),
   return DPLL($C, S - u, M \cup \{u = v\}$)
5. P = **first**(S); R = **rest**(S); $P = \{s\}\ R = \{r, q, p\}$
6. Return
   DPLL($C, R, M \cup \{P =$ **T**$\}$) $M \cup \{s =$ **T** $\}$
   ∨
   DPLL(C, $R, M \cup \{P =$ **F**$\}$)

$S = \{s, r, q, p\}\ M = \{\}$

CLAUSES

$p \vee q \vee r \vee s$ ∧
$\neg p \vee q \vee \neg r$ ∧
$\neg q \vee \neg r \vee s$ ∧
$p \vee \neg q \vee r \vee s$ ∧
$q \vee \neg r \vee \neg s$ ∧
$\neg p \vee \neg s$ ∧
$p \vee \neg q$ ∧

{ }

DPLL(C, S, M):

1. If (every $c \in C$ is **T**) ∨ (C is empty), return **T**
2. If C contains an empty clause, return **F**
3. If there is a (t, polarity v) = **pure symbol**(C), return DPLL($C, S - t, M \cup \{t = v\}$)
4. If there is a (u, polarity v) = **unit clause**(C), return DPLL($C, S - u, M \cup \{u = v\}$)
5. P = **first**(S); R = **rest**(S);
6. Return
   DPLL($C, R, M \cup \{P =$ **T**$\}$)
   ∨
   DPLL(C, $R, M \cup \{P =$ **F**$\}$)

$S = \{r, q, p\}$ $M = \{s =$ **T**$\}$

(3) (r, **F**)

CLAUSES

$p \vee q \vee r \vee s$	∧	= T
$\neg p \vee q \vee \neg r$	∧	
$\neg q \vee \neg r \vee s$	∧	= T
$p \vee \neg q \vee r \vee s$	∧	= T
$q \vee \neg r \vee \neg s$	∧	
$\neg p \vee \neg s$	∧	
$p \vee \neg q$	∧	

{ }

{$s =$ T}

DPLL(C, S, M):

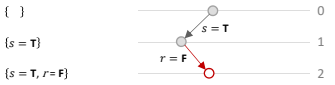
1. If (every $c \in C$ is **T**) ∨ (C is empty), return **T**
2. If C contains an empty clause, return **F**
3. If there is a (t, polarity v) = **pure symbol**(C), return DPLL($C, S - t, M \cup \{t = v\}$)
4. If there is a (u, polarity v) = **unit clause**(C), return DPLL($C, S - u, M \cup \{u = v\}$)
5. P = **first**(S); R = **rest**(S);
6. Return
   DPLL($C, R, M \cup \{P =$ **T**$\}$)
   ∨
   DPLL(C, $R, M \cup \{P =$ **F**$\}$)

$S = \{q, p\}$ $M = \{s =$ **T**, $r =$ **F**$\}$

(q, **F**)

CLAUSES

$p \vee q \vee r \vee s$	∧	= **T**
$\neg p \vee q \vee \neg r$	∧	= **T**
$\neg q \vee \neg r \vee s$	∧	= **T**
$p \vee \neg q \vee r \vee s$	∧	= **T**
$q \vee \neg r$	∧	= **T**
$\neg p$	∧	
$p \vee \neg q$	∧	



DPLL(C, S, M):

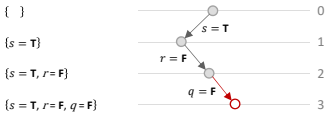
1. If (every $c \in C$ is **T**) ∨ (C is empty), return **T**
2. If C contains an empty clause, return **F**
3. If there is a (t, polarity v) = **pure symbol**(C), return DPLL($C, S - t, M \cup \{t = v\}$)
4. If there is a (u, polarity v) = **unit clause**(C), return DPLL($C, S - u, M \cup \{u = v\}$)
5. P = **first**(S); R = **rest**(S);
6. Return
   DPLL($C, R, M \cup \{P =$ **T**$\}$)
   ∨
   DPLL(C, $R, M \cup \{P =$ **F**$\}$)

$S = \{p\}$ $M = \{s =$ **T**, $r =$ **F**, $q =$ **F**$\}$

(p, **F**)

CLAUSES

$p \vee q \vee r \vee s$	∧	= **T**
$\neg p \vee q \vee \neg r$	∧	= **T**
$\neg q \vee \neg r \vee s$	∧	= **T**
$p \vee \neg q \vee r \vee s$	∧	= **T**
$q \vee \neg r$	∧	= **T**
$\neg p$	∧	
$p \vee \neg q$	∧	= **T**



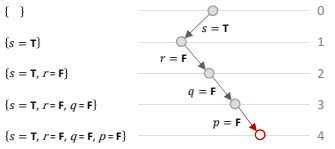
DPLL(C, S, M):

1. If (every $c \in C$ is **T**) ∨ (C is empty), return **T**
2. If C contains an empty clause, return **F**
3. If there is a (t, polarity v) = **pure symbol**(C), return DPLL($C, S - t, M \cup \{t = v\}$)
4. If there is a (u, polarity v) = **unit clause**(C), return DPLL($C, S - u, M \cup \{u = v\}$)
5. P = **first**(S); R = **rest**(S);
6. Return
   DPLL($C, R, M \cup \{P =$ **T**$\}$)
   ∨
   DPLL(C, $R, M \cup \{P =$ **F**$\}$)

$S = \{\}$ $M = \{s =$ **T**, $r =$ **F**, $q =$ **F**, $p =$ **F**$\}$

CLAUSES

$p \vee q \vee r \vee s$	∧	= T
$\neg p \vee q \vee \neg r$	∧	= T
$\neg q \vee \neg r \vee s$	∧	= T
$p \vee \neg q \vee r \vee s$	∧	= T
$q \vee \neg r$	∧	= T
$\neg p$	∧	= T
$p \vee \neg q$	∧	= T



QUESTIONS ?