9.4.1

Logical Inference 1 introduction

Chapter 9

Some material adopted from notes by Andreas Geyer-Schulz,, Chuck Dyer, and Mary Getoor

Overview

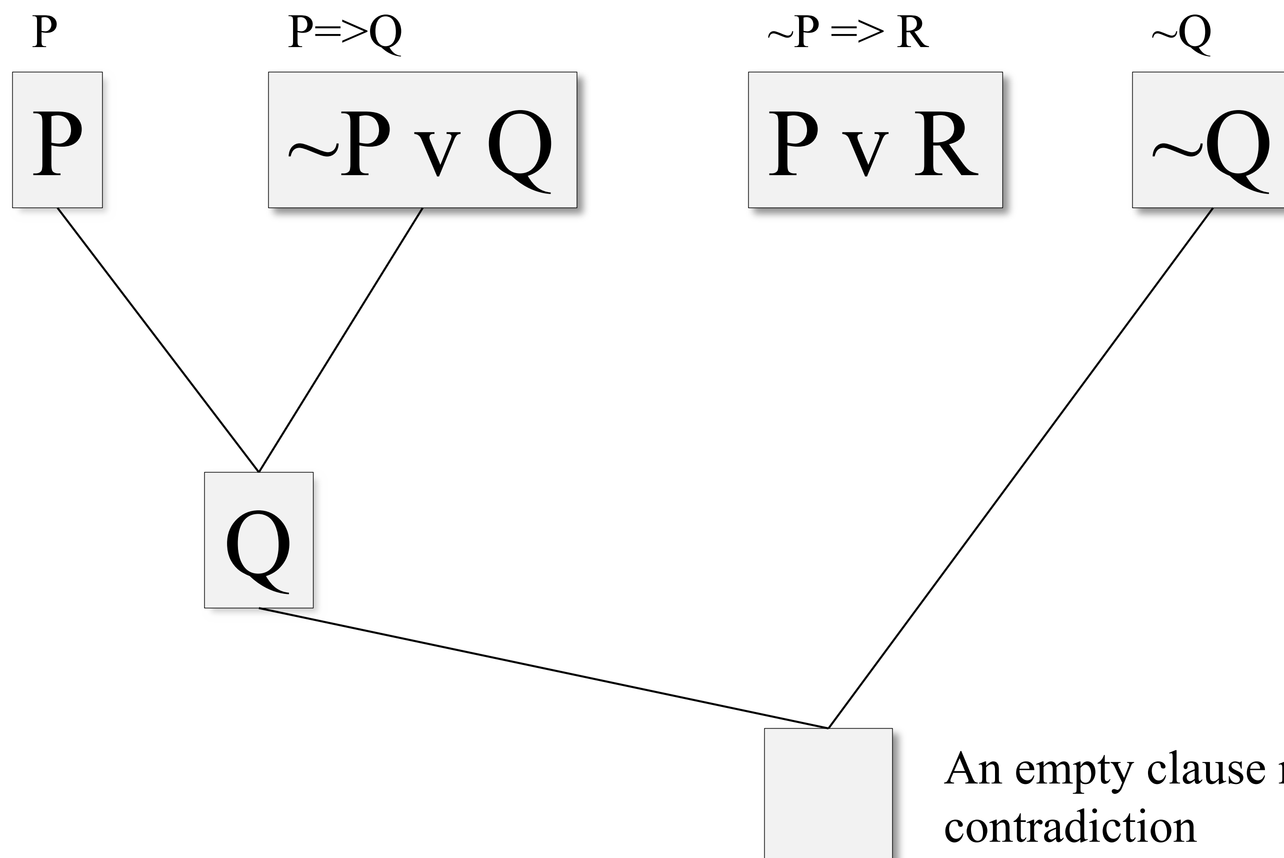
- A: Model checking for propositional logic
- Rule based reasoning in first-order logic
  - Inference rules and generalized modes ponens
  - Forward chaining
  - Backward chaining
- Resolution-based reasoning in first-order logic
  - Clausal form
  - Unification
  - Resolution as search
- Inference wrap up

From Satisfiability to Proof

- To see if a satisfiable KB entails sentence S, see if $\underline{KB \wedge \neg S}$ is satisfiable
  - If it is not, then the KB entails S
  - If it is, then the KB does not entail S
  - This is a refutation proof
- Consider the KB with (P, P=>Q, ~P=>R)
  - Does the KB it entail Q? R?

Does the KB entail Q?

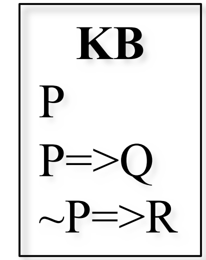
KB
P
P=>Q
~P=>R

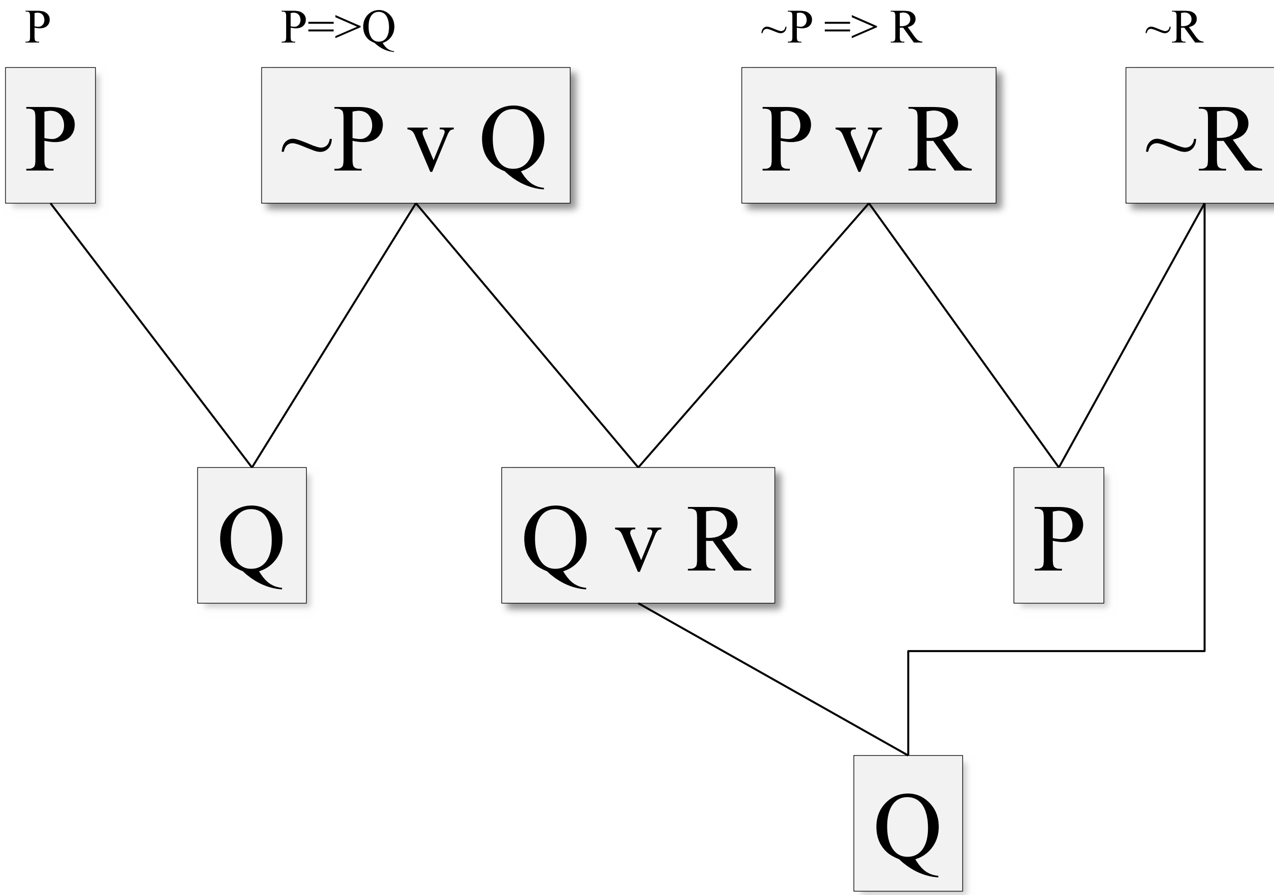


An empty clause represents a contradiction

We assume that every sentence in the KB is true. Adding ~Q to the KB yields a contradiction, so ~Q must be false, so Q must be true.

Does the KB entail R?





Adding ~R to KB does not produce a contradiction after drawing all possible conclusions, so it could be False, so KB doesn't entail R.

Propositional logic model checking

- Given KB, does a sentence S hold?
  - All the variables in S must be in the KB
  - A candidate model is just an assignment of T|F to every variable in the KB
- Basically it is generate and test:
  - Consider candidate models M for the KB
  - If $\forall$M S is true, then S is **provably true**
  - If $\forall$M $\neg$S, then S is **provably false**
  - Otherwise ($\exists$M1 S $\wedge$ $\exists$M2 $\neg$S): S is **satisfiable** but neither provably true or provably false

Efficient PL model checking (1)

Davis-Putnam algorithm (DPLL) is generate-and-test model checking with several optimizations:

- *Early termination:* short-circuiting of disjunction or conjunction sentences
- *Pure symbol heuristic*: symbols appearing only negated or un-negated must be FALSE/TRUE respectively

  e.g., in [(A∨¬B), (¬B∨¬C), (C∨A)] A & B are pure, C impure. Make pure symbol literal true: if there's a model for S, making pure symbol true is also a model
- *Unit clause heuristic*: Symbols in a clause by itself can immediately be set to TRUE or FALSE

Using the AIMA Code

```
python> python
Python ...
>>> from logic import *
>>> expr('P & P==>Q & ~P==>R')
((P & (P >> Q)) & (~P >> R))

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R'))
{R: True, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~R'))
{R: False, P: True, Q: True}

>>> dpll_satisfiable(expr('P & P==>Q & ~P==>R & ~Q'))
False

>>>
```

expr parses a string, and returns a logical expression

dpll_satisfiable returns a model if satisfiable else False

The KB entails Q but does not entail R

Efficient PL model checking (2)

- WalkSAT: a local search for satisfiability: Pick a symbol to flip (toggle TRUE/FALSE), either using min-conflicts *or* choosing randomly
- …or use *any* local or global search algorithm
- Many model checking algorithms & systems:
  - E.g.: MiniSat: minimalistic, open-source SAT solver developed to help researchers & developers use SAT"
  - E.g.: International SAT Competition (2002…2020): identify new challenging **benchmarks** to promote new **solvers** for Boolean SAT"

AIMA KB Class

```
>>> kb1 = PropKB()
>>> kb1.clauses
[]
>>> kb1.tell(expr('P==>Q & ~P==>R'))
>>> kb1.clauses
[(Q | ~P), (R | P)]
>>> kb1.ask(expr('Q'))
False
>>> kb1.tell(expr('P'))
>>> kb1.clauses
[(Q | ~P), (R | P), P]
>>> kb1.ask(expr('Q'))
{}
>>> kb1.retract(expr('P'))
>>> kb1.clauses
[(Q | ~P), (R | P)]
>>> kb1.ask(expr('Q'))
False
```

PropKB is a subclass

A sentence is converted to CNF and the clauses added

The KB does not entail Q

After adding P the KB does entail Q

Retracting P removes it and the KB no longer entails Q

Fin