From Satisfiability to Proof

• To see if a satisfiable KB entails sentence $S$, see if $KB \land \neg S$ 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 $(P, P \rightarrow Q, \neg P \rightarrow R)$
  – Does the KB it entail $Q$? $R$?
Does the KB entail Q?

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 \( \sim R \) 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 of the variables in S must be in the KB

• Basically generate and test:
  – Consider models M in which every sentence in the KB is TRUE
  – If $\forall M \ S$, then S is provably true
  – If $\forall M \neg S$, then S is provably false
  – Otherwise ($\exists M_1 \ S \land \exists M_2 \neg S$): 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 \([(A \lor \neg B), (\neg B \lor \neg C), (C \lor A)]\) 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
expr parses a string, and returns a logical expression
```
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...2018): identify new challenging *benchmarks* & to promote new *solvers* for Boolean SAT”
>>> kbl = PropKB()
>>> kbl.clauses
[]
>>> kbl.tell(expr('P==>Q & ~P==>R'))
>>> kbl.clauses
[(Q | ~P), (R | P)]
>>> kbl.ask(expr('Q'))
False
>>> kbl.tell(expr('P'))
>>> kbl.clauses
[(Q | ~P), (R | P), P]
>>> kbl.ask(expr('Q'))
{}
>>> kbl.retract(expr('P'))
>>> kbl.clauses
[(Q | ~P), (R | P)]
>>> kbl.ask(expr('Q'))
False

AIMA KB Class

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