Automated Reasoning

Searching for a Refutation

Jacques Fleuriot

Lecture VI

Example: Resolution Refutation

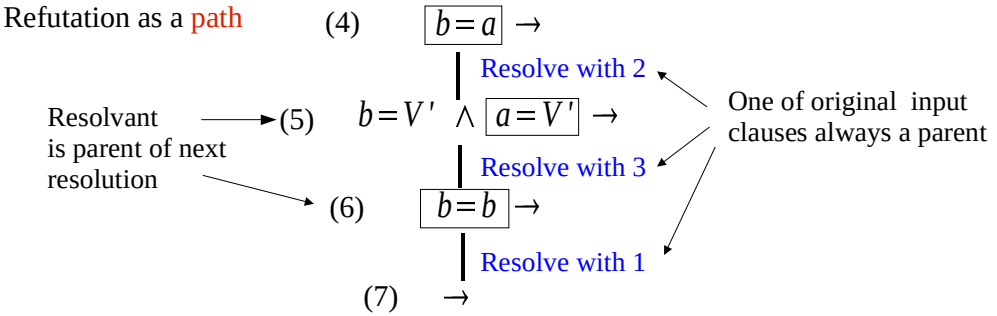
Input clauses in Kowalski Form:

1. $\rightarrow X = X$
2. $U = V \wedge W = V \rightarrow U = W$
3. $\rightarrow a = b$
4. $b = a \rightarrow$

Recall: $-ve \rightarrow +ve$

Negated conjecture $\forall X. \forall Y.\ X = Y \rightarrow Y = X$

Refutation as a path



Path gives an idea of shape of proof and also clauses that are used

Terminology: Linear Input Resolution Complete for Horn Clauses

The Story so Far

We have looked at

- Syntax and Semantics
- Rules of Inference + Proof
  - Resolution rule (refutation complete)
  - need for unification
  - Resolution proof : looks for refutation by deriving empty clause
- Matching and Unification
  - most general unifiers

Next issue?

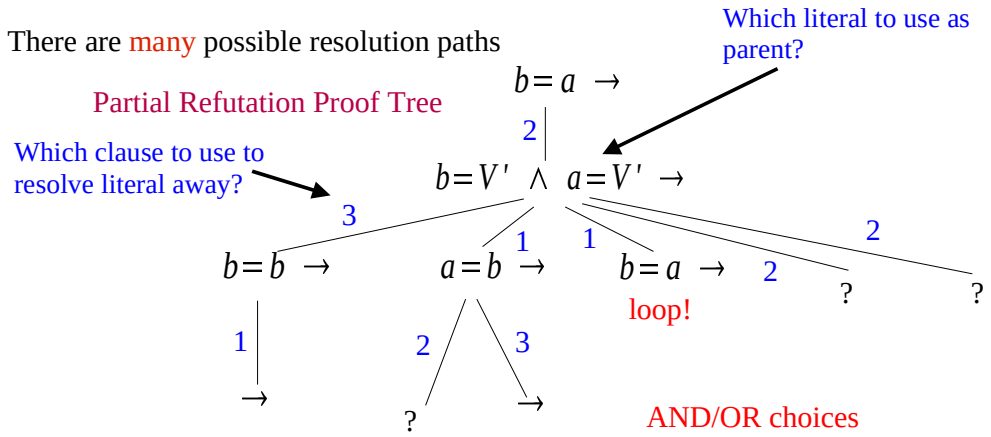
How do we actually find a proof using resolution?

Resolution Search Tree

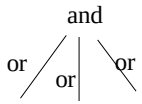
Input Clauses

1. $\rightarrow X = X$
2. $U = V \wedge W = V \rightarrow U = W$
3. $\rightarrow a = b$
4. $b = a \rightarrow$

There are many possible resolution paths



AND/OR Choices



AND Choice

- Case: Can resolve more than one literal in a clause
- All literals must eventually be resolved away to derive empty clause
  - must choose one AND then another
  - therefore, AND choices do not matter

OR Choice

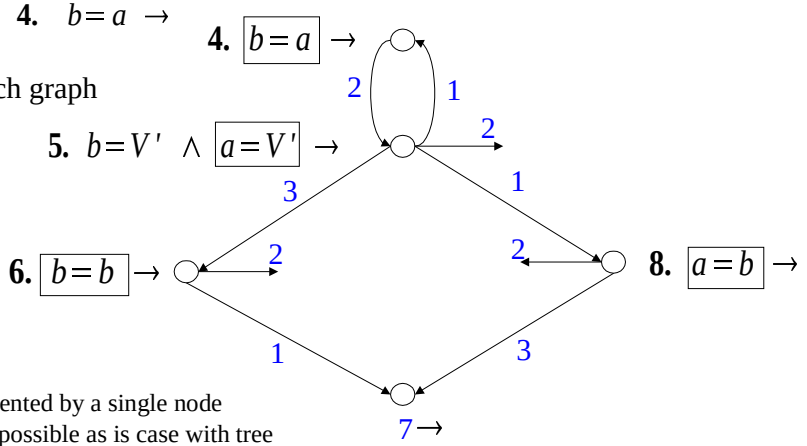
- Case: Can resolve with more than one clause
- different clauses may not always yield a proof
- some paths may not lead to empty clause →
  - must choose one OR the other
  - therefore, OR choices do matter

Resolution Search Graph

Input Clauses

1. $\rightarrow\ X = X$
2. $U = V \wedge W = V\ \rightarrow\ U = W$
3. $\rightarrow\ a = b$
4. $b = a\ \rightarrow$

Refutation search graph



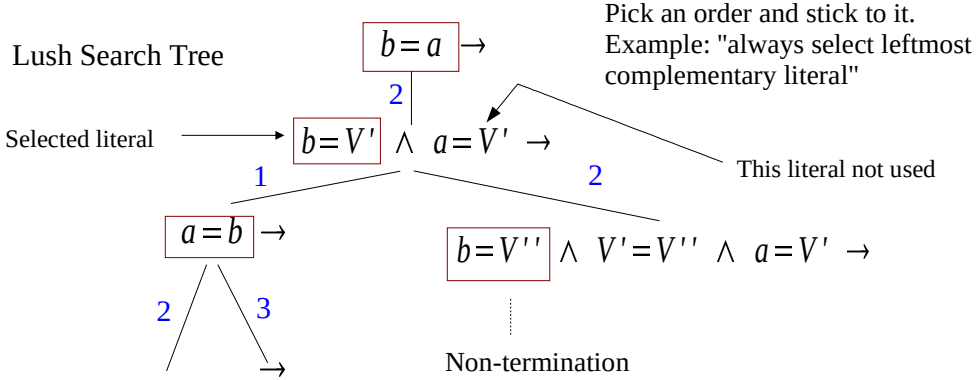
- Each clause represented by a single node
  i.e no duplication possible as is case with tree
- Loops are clearly depicted

Lush Resolution

Input Clauses

1. $\rightarrow\ X = X$
2. $U = V \wedge W = V\ \rightarrow\ U = W$
3. $\rightarrow\ a = b$
4. $b = a\ \rightarrow$

Lush Search Tree



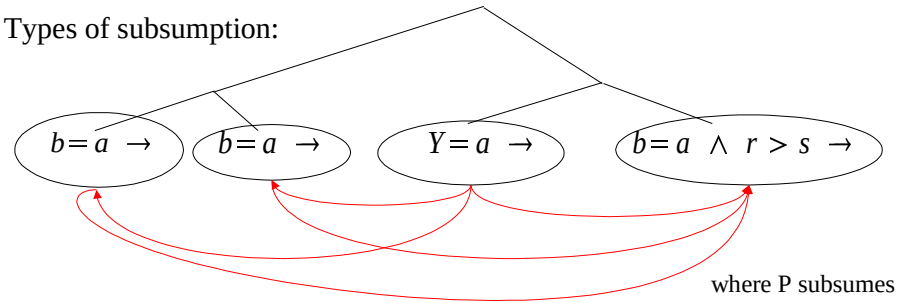
Pick an order and stick to it.
Example: "always select leftmost complementary literal"

This literal not used

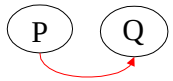
- Lush: **L**inear input resolution with **U**nrestricted **S**election function for **H**orn Clauses
- Lush resolution is complete for Horn Clauses

Looping and Subsumption

Types of subsumption:



where P subsumes Q:



Definition of subsumption

$$C \text{ subsumes } D \text{ iff } \exists D'.\ \exists \phi\ \ D \equiv (C \circ \phi) \vee D'$$

May be"f"

May be empty

Example: Subsumption

$Y = a \rightarrow$ subsumes $b = a \wedge s > t \rightarrow$

since

$b = a \wedge s > t \rightarrow \equiv (Y = a) \circ \phi \wedge s > t \rightarrow$

where $\phi = \{b/Y\}$

i.e

$\neg(Y = a)$ subsumes $\neg(b = a) \vee \neg(s > t)$

since

$\neg(b = a) \vee \neg(s > t) \equiv (\neg(Y = a) \circ \phi) \vee \neg(s > t)$

where $\phi = \{b/Y\}$

Non-Horn Clauses

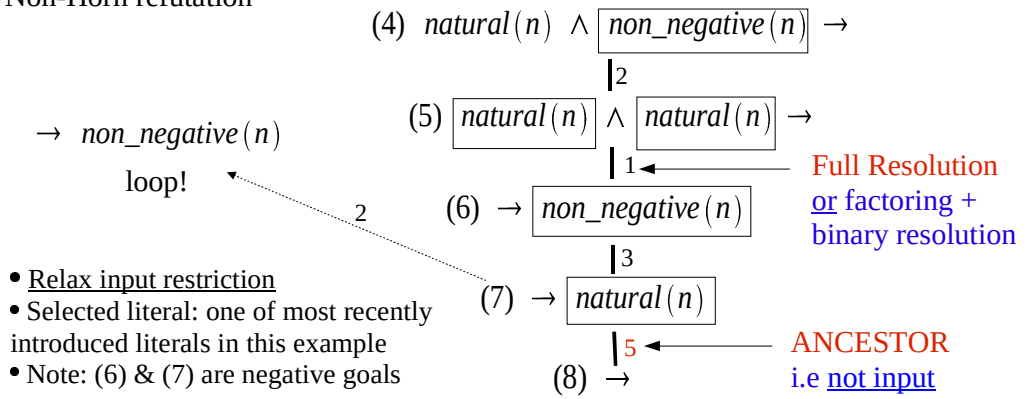
Input clause

1. $\rightarrow natural(n) \vee non\_negative(n)$
2. $natural(n) \rightarrow non\_negative(n)$
3. $non\_negative(n) \rightarrow natural(n)$

Top Clause:

4. $natural(n) \wedge non\_negative(n) \rightarrow$

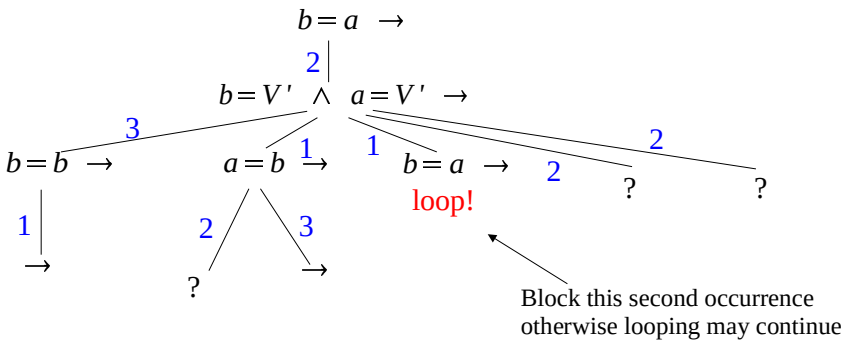
Non-Horn refutation



- Relax input restriction
- Selected literal: one of most recently introduced literals in this example
- Note: (6) & (7) are negative goals

Subsumption Checking

- If a clause is subsumed, we can consider that branch to be blocked
- Subsumption checking stops further development of subsumed clauses: it adds a restriction to resolution
- Subsumption preserves completeness (with some conditions)



Search Strategies

- Full resolution is complete
- Lush resolution is complete for Horn clauses

Yes, assuming the *search strategy* is complete

Completeness of the *inference system*: "there *exists* a proof using (say) Lush resolution"

Question: How do we find (search for) the proof?

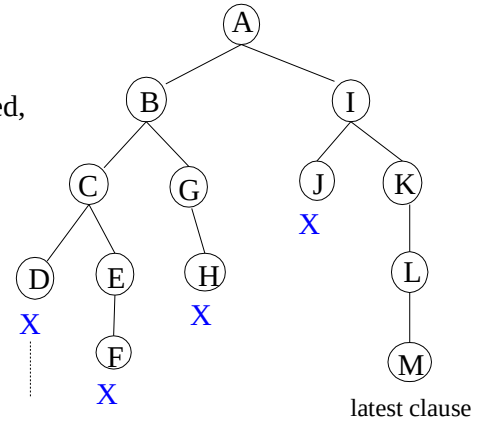
i.e. in terms of tree: how to make OR choices?

Will briefly look at four strategies:

- Depth-first search
- Breadth-first search
- iterative deepening
- Best-first search

Depth First Search

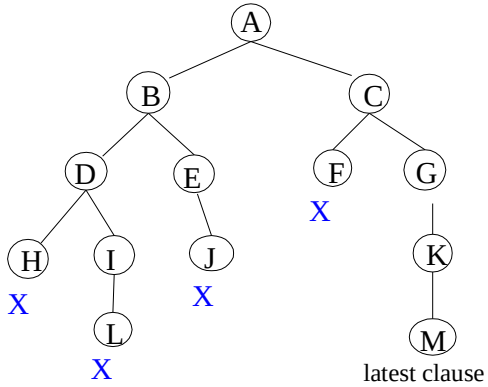
- Pursue current leftmost path until blocked,
- then back up to last choice
- incomplete search
- not guaranteed to find shortest proof



Prepend to agenda:

$$A \rightarrow \boxed{\begin{matrix}B\\I\end{matrix}} \rightarrow \boxed{\begin{matrix}C\\G\end{matrix}} \rightarrow \boxed{\begin{matrix}D\\E\end{matrix}} \rightarrow \begin{matrix}E\\G\\I\end{matrix} \rightarrow \begin{matrix}F\\G\\I\end{matrix} \rightarrow \begin{matrix}G\\I\end{matrix} \rightarrow \begin{matrix}H\\I\end{matrix} \rightarrow \begin{matrix}I\\\\\end{matrix} \rightarrow \begin{matrix}J\\K\end{matrix} \rightarrow \ldots$$

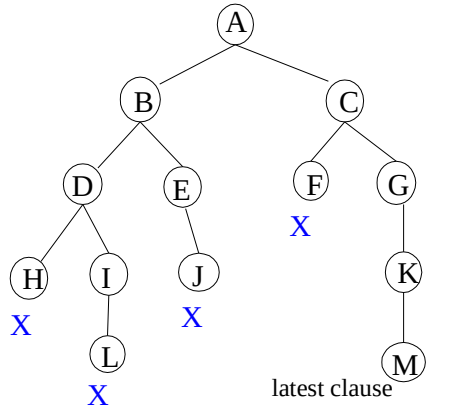
Iterative Deepening



- Compromise between breadth-first and depth-first
- Explore depth first but at most to depth 1
  then explore depth-first to depth 2
  then explore depth-first to depth 3, etc.
- Complete search; shortest proof first
- no more storage required than depth-first
- explores nodes high in search tree several times

Breadth First Search

- Explore each level in turn
- Complete search
- always finds shortest proof
- expensive in terms of storage



$$A \rightarrow \boxed{\begin{matrix}B\\C\end{matrix}} \rightarrow \boxed{\begin{matrix}C\\D\\E\end{matrix}} \rightarrow \boxed{\begin{matrix}D\\E\\F\\G\end{matrix}} \rightarrow \begin{matrix}E\\F\\G\\H\\I\end{matrix} \rightarrow \begin{matrix}F\\G\\H\\I\\J\end{matrix} \rightarrow \begin{matrix}G\\H\\I\\J\end{matrix} \rightarrow \begin{matrix}H\\I\\J\\K\end{matrix} \rightarrow \begin{matrix}I\\J\\K\\L\end{matrix} \rightarrow \begin{matrix}J\\K\\L\end{matrix} \rightarrow \begin{matrix}K\\L\\M\end{matrix} \rightarrow \begin{matrix}L\\M\end{matrix} \rightarrow \begin{matrix}M\\\\\end{matrix}$$

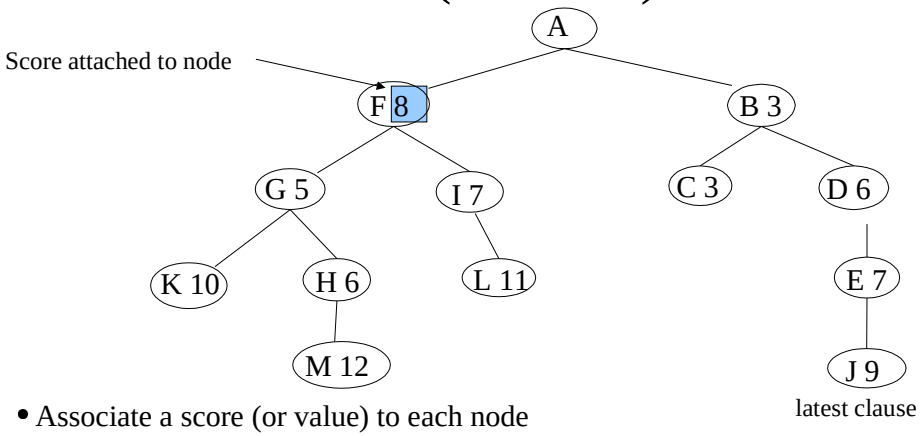
Order of Nodes in Iterative Deepening

In what order are the nodes of the tree on the previous slide searched?

- (depth 1) A B C
- (depth 2) A B D E C F G
- (depth 3) A B D H I E J C F G K

At each level, search looks like breadth-first

Best First (heuristic) Search



- Associate a score (or value) to each node
- Need some evaluation function to evaluate score of node
- Choice made on basis of evaluation
- Evaluation function (hence score) is usually heuristic, so no guarantees:
  - may be complete
  - may be shortest first

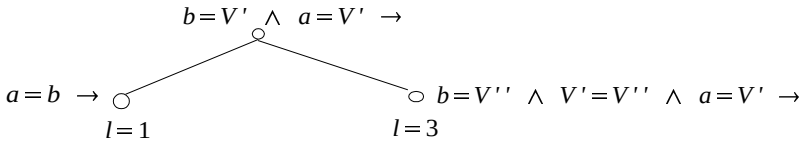
Perils of Evaluation Functions!

- Best-first search can be useful, but for some search problems it may be inappropriate
- How much time do we spend fiddling with the evaluation function?

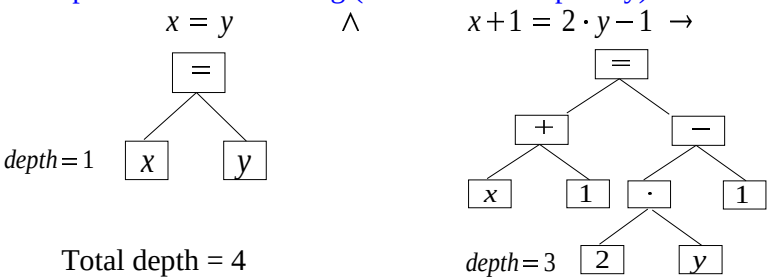
  Do change it over time/with more experience
- Can we explain success or lack of success?
- Can we learn general lessons for AR
- How about cheating?

Evaluation Functions

- Good evaluation functions are hard to define
- Good heuristic scores usually depend on the problem
- Example of best-first scoring: functions are from clauses to numbers
  - e.g. Length of clause (the shorter the better)



  - Depth of function nesting (measure of complexity)



Summary

- Search trees and graphs
- subsumption as a restriction
- Lush resolution
- Ancestor resolution for non-Horn clauses
- Search Strategies:
  - depth-first, breadth-first, best-first search
  - need for evaluation function
  - incomplete and complete search strategies
- Bundy Chapter 6