Resolution Refinements

Summary

- Refinement for Resolution
- Linear Resolution [Chang-Lee Ch. 7.2]
- Ordered Resolution
- Ordered Clause [Chang-Lee Ch. 7.4]
- Ordered Clause for Linear Resolution [Chang-Lee Ch. 7.4]
- Linear Deduction and Tree Searching [Chang-Lee 7.6]

Resolution so far

Need of refinement

- Resolution is refutationally complete
- Level Saturation generates all possible clauses
- Deletion strategies can be used to eliminate irrelevant and redundant clauses
- However, generating clauses and deleting them is not efficient
  - waste of time to generate them
  - waste of time and memory to check that they are irrelevant/redundant

Resolution Refinements

Refine resolution

- Want to avoid irrelevant and redundant clauses to be generated
- Many important refinements of resolution
- Main ones:
  1. Semantic Resolution
  2. Lock Resolution
  3. Linear Resolution

Semantic Resolution

Main features

- Choose one interpetation for S
- Divide clauses in two sets according to one interpretation
- Resolve clauses from different sets
- Semantic Resolution is complete
- Related methods:
  1. Hyperresolution: Semantic resolution where the interpretation is the negation of all atoms.
  2. Set-of-Support: individuates a set T such that $S - T$ is satisfiable.

Lock Resolution

Main features

- Arbitrarily assign an (integer) index to each literal in every clause
- Allow resolution only on literals of lowest index
- Indexes in resolvents are inherited from parent clauses
- Lock Resolution is complete

Linear Resolution

Basic Concepts

Chain application of Resolution steps

- Given a set of clauses S
- Choose a clause $C_0 \in S$
- Choose a second clause B_0 in S
- Apply resolution and obtain R_1
- Choose another clause B_1 from S or from previously generated resolvents
- Apply resolution to R_1 and B_1 generating R_2
- Repeat until $\square$ appears

Linear Resolution

Main benefits

- Extremely simple structure
- Refutationally Complete
- Can be used with other refinements (e.g., set-of-support)
- Many heuristics to make it very efficient

Linear Resolution: Definition

Definition (Definition)

Given a set S of clauses and a clause $C_0 \in S$, a linear deduction of C_n from S with top clause C_0 is a sequence of resolvents $R_1, \cdots, R_n$ where $R_n = C_n$ and:

- $R_0 = C_0$
- for $i = 0, \cdots, n-1$ R_{i+1} is a resolvent of C_i (called center clause) and B_i (called side clause)
- each B_i is either a clause in S or is a C_j for $j < i$

Linear Resolution: Example

Example (Linear Resolution)

Consider the set $S = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$, the following is a linear reduction of $\square$ from S with $C_0 = P \vee Q$:

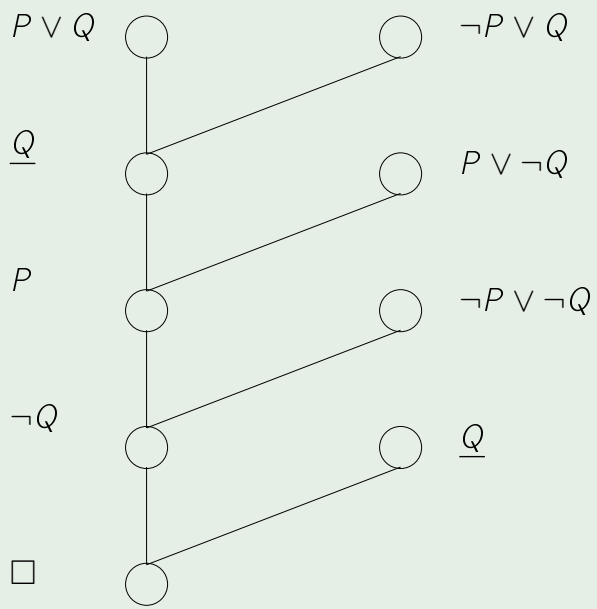

Figure: Linear deduction of $\square$ from S with $C_0 = P \vee Q$

Linear Resolution: Example II

Example (Linear Resolution)

Consider the set $S = \{P(x) \vee P(y), \neg P(u) \vee \neg P(v)\}$, give a linear reduction of $\square$ from S with $C_0 = P(x) \vee P(y)$:

Ordered Resolution: Basic Concepts

Pruning possible resolution steps

- Aim: reduce the possible application of resolution maintaining completeness
- Use order to constraint the resolution process
  - e.g., consider always the literal which is maximal with respect to the ordering
- Allow resolution only when the constraint is met

Different types of ordered resolution

types of order resolutions

1. Ordered resolution
   - Given a partial order on atoms
   - Apply resolution only when the literals resolved upon are maximal in both premises
2. Selection (also called ordered resolution)
   - Sort atoms of each clause into a fixed sequence
   - Apply resolution only when in at least one ofthe two premises a maximal atom is involved
   - Maximal in this case means leading (e.g. last of the sequence)

Ordered resolution: Example

Example (Ordered Resolution)

Consider the set $S = \{P \vee Q, Q \vee \neg P, \neg Q \vee P, \neg P \vee \neg Q\}$ And the order $P > Q$.

Selection: Example

Example (Selection)

Consider the set $S = \{P \vee Q, Q \vee \neg P, \neg Q \vee P, \neg P \vee \neg Q\}$
Order is given for each clause.

- If we perform resolution between maximal literals in both premises we obtain only $P \vee \neg P$ and $Q \vee \neg Q$
- If we allow resolution between a maximal literal of one premise and any literal of the other we can deduce $\square$

Ordered Resolution: Discussion

Discussion

- Both are refutationally complete
- Ordered Resolution requires orderings on atom set, many possibilities from Rewriting literature
- Selection can be extremely efficient when combined with linear resolution

Orderings for FOL

Orderings

- For FOL formulas ordering of atoms is not obvious:
  - Ordering on predicate symbol is all we need for propositional formulas
  - For FOL we need something more: $P(f(f(x))) >? P(f(x))$, $P(x) >? P(a)$
- We need a way to order atoms starting from ordering on predicate, function and costant symbols.

Orderings

Definition (strict order)

A strict ordering for a set P is an anti-reflexive transitive binary relation $>$

Orderings

- Anti-reflexive: $\forall x \in A \; x \not> x$
- Transitive: $\forall x, y, z \in A$ if $x > y$ and $y > z$ then $x > z$

Example (strict order)

Natural number set N with greather than $>$ relation define a strict order

Orderings for Atoms

Orderings for Atoms

- We are interested in strict orderings over the Atom set with specific properties:
  - Compatibility: $f \in \mathcal{F} \cup \mathcal{P}$ and $s_1 > s_2, s_i \in Term$ then $f(t_1, \cdots, s_1, \cdots, t_n) > f(t_1, \cdots, s_2, \cdots, t_n)\ \forall t_i \in Terms$
  - Stability: if $s_1 > s_2$ and σ is a substitution then $s_1\sigma > s_2\sigma$
- a strict ordering for which 1 and 2 hold is a rewriting ordering
- a semplification ordering is a rewriting ordering for which the following holds: $f(x_1, \cdots, x_n) > x_i\ \forall f \in \mathcal{F}$

Terminating Orderings

Termination

- A strict order $>$ is terminating iff there is no infinite chain $x_0 > x_1 > x_2 > \cdots$
- A strict order on a finite set is always terminating
- A rewriting order which is terminating is said a reduction order

Main Orderings

Orderings

- We assume a strict ordering $>_p$ on symbols of $\mathcal{F} \cup \mathcal{P}$. (we assume constants to be functions with arity $= 0$)
- Two widely used orders for theorem proving are:
  - Lexicographic Path Ordering (LPO)
  - Knuth Bendix Order (KBO)
- Both LPO and KBO are semplification orderings (and reduction orderings for finite languages)
- Both of them are total orders for ground atoms

LPO definition

Definition (LPO Definition)

Given a strict ordering $>_p$ on symbols of $\mathcal{F} \cup \mathcal{P}$, given two atoms s, t, $s >_{lpo} t$ if one of the following conditions hold:

- (LPO1) $s \equiv f(s_1, \cdots, s_n)$ and for some $i = 1, \cdots, n$ we have that $s_i >_{lpo} t$ or $s_i \equiv t$.
- (LPO2) $s \equiv f(s_1, \cdots, s_n), t \equiv g(t_1, \cdots, t_m)$, $f >_p g$ and $s >_{lpo} t_i$ for all $i = 1, \cdots, m$.
- (LPO3) $s \equiv f(s_1, \cdots, s_n), t \equiv f(t_1, \cdots, t_n)$ and for some $i = 1, \cdots, n$ we have that:
  $s_1 \equiv t_1, \cdots, s_i >_{lpo} t_i, s >_{lpo} t_{i+1}, \cdots, s >_{lpo} t_n$.

LPO: Example

Example (LPO)

Verify that $a(s(x), s(y)) >_{lpo} a(x, a(s(x), y))$ given the strict order $a >_p s >_p 0$

- use (LPO3) and verify that:
  1. $s(x) >_{lpo} x$; true because we can use LPO1 with $x \equiv x$
  2. $a(s(x), s(y)) >_{lpo} a(s(x), y)$
     1. use LPO3 again, and LPO1 verifying that $y \equiv y$

KBO Definition

KBO Requirements

- We need a strict ordering $>_p$ on symbols of $\mathcal{F} \cup \mathcal{P}$.
- An admissible weight function w that assigns to each function, predicate and variable symbol an integer

Weight function

Definition (admissible weight function)

Admissible weight function must meet the following requirements

1. There exists $d > 0$ such that $d = w(x)$ for all variable x. Moreover $w(c) \geq d$ for all constants
2. we can have $w(f) = 0$ but only for a single unary function or predicate symbol f. In that case, $f >_p g$ must hold for all other function, constant or predicate symbols.

Weight function is extended to all atoms as follow:

$$w(t) = \sum_{x \in \mathcal{V}} w(x)|t|_x + \sum_{f \in \mathcal{F} \cup \mathcal{P}} w(f)|t|_f$$

where t is an atom and $|t|_\alpha$ is the number of occurrence of α (variable, function or predicate symbol) in $|t|$

KBO definition

Definition (KBO Definition)

Given a strict ordering $>_p$ on symbols of $\mathcal{F} \cup \mathcal{P}$, a weight function w and two atoms s and t, $s >_{kbo} t$ iff for all variables x $|s|_x \geq |t|_x$ and one of the following conditions hold:

- (KBO1) $w(s) > w(t)$.
- (KBO2) $w(s) = w(t), s \equiv f^1(x)$ and $t \equiv x$, for some $f \in \mathcal{F}_1$ and for some x. Note that in KBO2 since $w(s) = w(t)$ then $w(f) = 0$.
- (KBO3) $w(s) = w(t), s \equiv f(s_1, \cdots, s_n), t \equiv g(t_1, \cdots, t_m)$ and $h >_p g$
- (KBO4) $w(s) = w(t), s \equiv f(s_1, \cdots, s_n), t \equiv f(s_1, \cdots, s_n)$ and for some $i = 1, \cdots, n$ we have that

$$s_1 \equiv t_1, \cdots, s_i >_{kbo} t_i$$

KBO Example

Example (KBO Example)

Show that $i(x * y) >_{kbo} i(x) * i(y)$ assuming that: $i >_p *$, $w(i) = 0, w(*) = w(v) = 1 \; \forall v \in \mathcal{V}$

- $|i(x * y)|_x = |i(x) * i(y)|_x$ and $|i(x * y)|_y = |i(x) * i(y)|_y$
- $w(i(x * y)) = w(i(x) * i(y)) = 3$
- since $i >_p *$ we can apply KBO3 and show that $i(x * y) >_{kbo} i(x) * i(y)$ holds.

Example for KBO and LPO

Example (Examples)

Consider the atoms $s = P(f(f(x)))$ and $t = P(f(x))$, and assume $P > f$.

- Decide whether $s >_{LPO} t$
- Assuming $w(x) = w(f) = w(P) = 1$, decide whether $s >_{KBO} t$

Ordered resolution using Orderings

Example

Consider the set $S = \{\neg P(x) \vee Q(f(x)), P(a), \neg Q(y))\}$ Show an ordered resolution, assuming $P >_p Q >_p f >_p a$, $w(y) = w(x) = w(f) = w(P) = w(Q) = w(a) = 1$ and using the KBO ordering and level saturation deleting redundant clauses.

Ordered Clauses

Basic Concepts

- Consider a clause as a sequence of literals (not a set)
- By doing this we specify the order of all literals in a clause
- $L_2 > L_1$ if L_2 follows L_1 (going from left to right)
- The largest literal in a clause is always the last literal

Ordered factor

Definition (Ordered Factor)

If two or more literals (with the same sign) of an ordered clause have an MGU σ, then the ordered clause obtained from the sequence $C\sigma$ by deleting any literals that is identical to a smaller literal in the sequence is called an ordered factor of C.

Generating an ordered factor

- find an MGU
- apply σ
- merge literals to the left

Ordered Factor: Example

Example (Ordered Factor)

Consider the ordered clause $C = P(x) \vee Q(x) \vee P(a)$, and the MGU $\sigma = \{a/x\}$ An ordered factor for this clause is $P(a) \vee Q(a)$

Notice that $Q(a) \vee P(a)$ is not an oredered factor for C, but would be a factor if the clause was not ordered

Ordered binary resolvent

Definition (Ordered Binary Resolvent)

- C_1 and C_2 ordered clauses
- L_1 and L_2 two literals in C_1 and C_2
- σ MGU for L_1 and $\neg L_2$

Obtain the ordered factor $C$by:

- concatenating $C_1\sigma$ and $C_2\sigma$
- removing $L_1\sigma$ and $L_2\sigma$
- merging left all identical literals

C is the ordered factor of C_1 against C_2, L_1 and L_2 are the literals resolved upon

Ordered Binary Resolvent: Example

Example (Ordered Binary Resolvent)

Consider the ordered clauses $C_1 = P(x) \vee Q(x) \vee R(x)$, and $C_2 = \neg P(a) \vee S(a) \vee Q(a)$. Choose $L_1 = P(x)$ and $L_2 = \neg P(a)$ the MGU $\sigma = \{a/x\}$.

- $C_1\sigma$ concatenated to $C_2\sigma$ is $P(a) \vee Q(a) \vee R(a) \vee \neg P(a) \vee S(a) \vee Q(a)$
- Removing $L_1\sigma$ and $L_2\sigma$ we have $Q(a) \vee R(a) \vee S(a) \vee Q(a)$
- Merging left $Q(a)$ we have $Q(a) \vee R(a) \vee S(a)$

$P(x)$ and $\neg P(a)$ are the literals resolved upon
Notice that by resolving C_2 against C_1 we have a different binary resolvent: $S(a) \vee Q(a) \vee R(a)$

Ordered resolvent

Definition (Ordered Resolvent)

Given two clauses C_1 and C_2 (parent clauses) an ordered resolvent of C_1 against C_2 is one of the following binary resolvents:

- an ordered binary resolvent of C_1 againts C_2
- an ordered binary resolvent of C_1 against an ordered factor of C_2
- an ordered binary resolvent of an ordered factor of C_1 against C_2
- an ordered binary resolvent of an ordered factor of C_1 against an ordered factor of C_2

Ordered Resolvent: Example

Example (Ordered Resolvent)

Consider the ordered clauses $C_1 = P(x) \vee Q(x) \vee R(x) \vee P(a)$, and $C_2 = \neg P(a) \vee Q(a)$.

- $C_1' = P(a) \vee Q(a) \vee R(a)$ is an ordered factor of C_1
- $C = Q(a) \vee R(a)$ is an ordered binary resolvent of C_1' against C_2
- Therefore C is an oredered resolvent of of C_1 against C_2

Resolution with oredered clauses

Discussion

- Without furher restriction resolution with ordered clauses is complete
- Ordered clauses can be used to restrict possible resolution application
  - For example allow resolution only with the greatest literal of one of the two premises
  - Still refutationally complete
- Extremely efficient when combined with linear resolution

Refinements for Linear Resolution

basic concepts

We aim to:

1. avoid storing intermediate clauses
   - by storing information on resolved literals
2. Further restrict possible applications of resolution
   - by using ordered clauses

Example

Example (plain linear resolution)

Consider the set of clauses

$S = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$

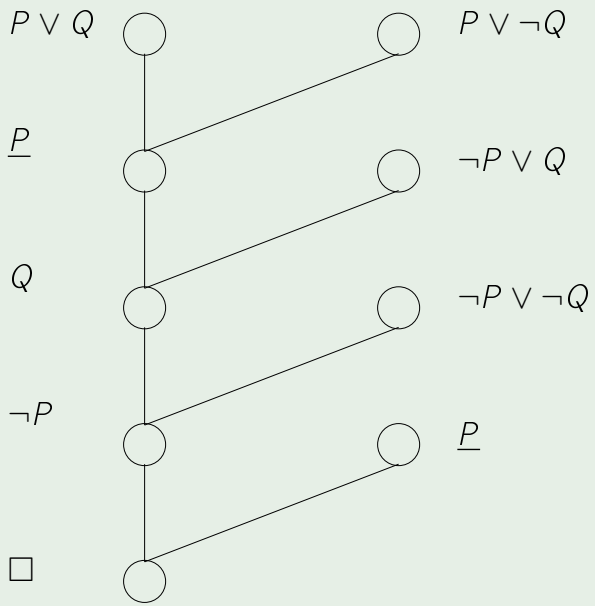

Figure: A possible linear resolution for S

Storing information on resolved literals

storing information

- We store literals resolved upon
- Keep one of the two literals in the clause in the position induced by the order
- Do not use it for further resolution
- Convention: we mark the used literals using underline
- When an underlined literal is the last one, we delete it

Example

Example (storing literals)

Consider the clauses $C_1 = P \vee Q$ and $C_2 = \neg Q \vee R$

- $C = P \vee R$ is an ordered resolvent of C_1 against C_2
- We store $P \vee \underline{Q} \vee R$

Notice that storing Q or $\neg Q$ is irrelevant, we just need one.
We always store the last literal of the center clause.

Ordered Linear Resolution

OL resolution

- ordered clause
- linear resolution
- storing of removed literals

Ordered Linear Resolution: Example

Example (OL-Resolution)

Consider the set of clauses

$S = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$

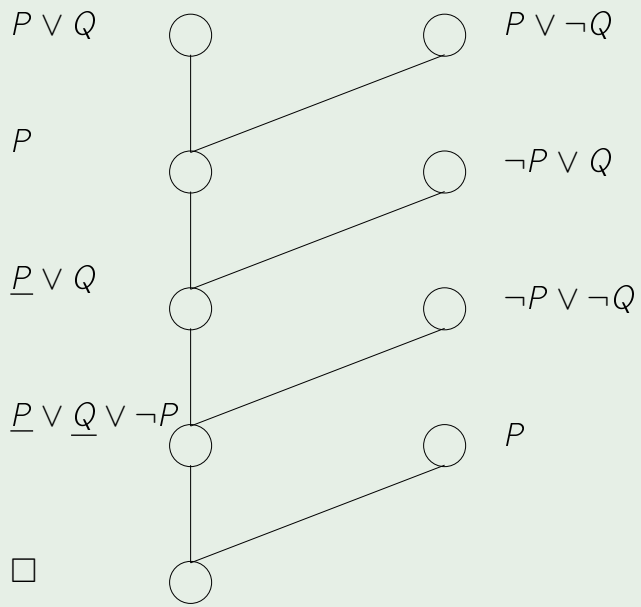

Figure: A possible OL resolution for S

Comments on the example

Comments

- We always resolve the last literal of center clauses
- We do not consider underlined literals in the resolution steps
- We delete underlined literals when they appear last
- In the last step we have a complementary pair between an underlined and a normal literal
- this happens when we need to use a (previous) center clause as a side clause
- We call this kind of clauses reducible clauses

Reducible clauses

Definition

Reducible clauses An ordered clause C is a reducible clause iff the last literal of C is unifiable with the negation of an underlined literal in C.

Reduction

When a reducible clause appears:

- we do not need to retrieve a center clause from memory
- we simply delete the last literal in the clause

Reduction

Definition

Reduction

- Let C be a reducible clause.
- Let L be a unifiable literal with the negation of an underlined literal L'
- Let σ be the MGU

The reduced ordered clause of C is obtained from $C\sigma$ by:

- deleting $L\sigma$
- deleting every subsequent underlined literals not followed by a normal literal

Example: Reduction

Example (Reduction)

Consider $C = \underline{P} \vee \underline{Q} \vee \neg P$

- we delete the last literal P
- we are left with underlined literals not followed by normal literals
- we delete both underlined literals and obtain $\square$

Ordered Clauses and information storing

underlined literals

Underlined literals are for information storing only:

- ordered factor with underlined literals:
  - Same as ordered factors,
  - We delete underlined literals not followed by other literals
- ordered binary resolvent with underlined literals
  - Same as ordered binary resolvent
  - We underline the literal resolved upon in the first caluse
  - We delete underlined literals not followed by other literals
- ordered resolvent is exactly the same.

Example: Ordered factor

Example (Ordered factor)

Consider the ordered clause $C = P(x) \vee \underline{Q(x)} \vee P(a)$, and the MGU $\sigma = \{a/x\}$

- we generate the ordered factor $P(a) \vee \underline{Q(a)}$
- we delete the last underlined literal $\underline{Q(a)}$

An ordered factor for this clause is $P(a)$

Example: ordered binary resolvent

Example (Ordered Binary Resolvent)

Consider the ordered clauses $C_1 = \underline{\neg Q(x)} \vee R(x) \vee P(x)$, and $C_2 = \neg Q(a) \vee \underline{S(a)} \vee \neg P(a)$. Choose $L_1 = P(x)$ and $L_2 = \neg P(a)$ the MGU $\sigma = \{a/x\}$.

- $C_1\sigma$ concatenated to $C_2\sigma$ is $\underline{\neg Q(a)} \vee R(a) \vee P(a) \vee \neg Q(a) \vee \underline{S(a)} \vee \neg P(a)$
- Removing $L_2\sigma$ and underlining $L_1\sigma$ we have $\underline{\neg Q(a)} \vee R(a) \vee \underline{P(a)} \vee \neg Q(a) \vee \underline{S(a)}$
- Removing $\underline{S(a)}$ which is not followed by any other literals we have $\underline{\neg Q(a)} \vee R(a) \vee \underline{P(a)} \vee Q(a)$

$P(x)$ and $\neg P(a)$ are the literals resolved upon

Notice that we do not merge left $\neg Q(a)$ with $\underline{\neg Q(a)}$

Ordered Linear Resolution

Definition (OL Resolution)

Given S set of ordered clauses and $C_0 \in S$, an OL-deduction of C_n from S with top clause C_0 is a linear deduction of C_n for which the following conditions hold:

1. For $i = 0, \cdots, n-1$ C_{i+1} is an ordered resolvent of C_i (center ordered clause) against B_i (side ordered clause). The literal resolved upon in C_i (or an ordered factor of C_i) is always the last literal.
2. Each B_i is either an ordered clause in S or an instance of some ordered center clause C_j with $j < i$.
   - B_i is an instance of some ordered center clause C_j with $j < i$ iff C_i is a reducible ordered clause, in this case C_{i+1} is the reduced ordered clause of C_i.
3. No tautology is in the deduction.

Property of ordered reducible clauses

Lemma

In an OL-deduction, if C_i is a reducible ordered clause, then there exist a center ordered clause C_j with $j < i$, such that the reduced ordered clause C_{i+1} of C_i, is an ordered resolvent of C_i against and instance of C_j

Basic ideas

- $C_i = C_i' \vee \underline{L_1} \vee B_i' \vee L_2$
- $L_1\sigma = \neg L_2\sigma$
- $C_{i+1} = C_i'\sigma \vee \underline{L_1\sigma} \vee B_i'\sigma$
- Since L_1 is underlined it means that we resolved upont that literal before.
- $C_j = D_j \vee L_j$, L_1 is an instance of L_j and C_i' is an instance of D_j.
- C_{i+1} is an ordered resolv. of C_i against C_j with σ MGU

Re-stating OL-deduction conditions

new conditions

1. For $i = 0, \cdots, n-1$ If C_i is a reducible ordered clause then C_{i+1} is the reduced ordered clause of C_i. Otherwise, C_{i+1} is an ordered resolvent of C_i (center ordered clause) against B_i (side ordered clause). The literal resolved upon in C_i (or an ordered factor of C_i) is always the last literal.
2. Each B_i is an ordered clause in S.
3. No tautology is in the deduction.

OL-refutation: Example I

Example (OL-Refutation I)

Consider a set of ordered clauses $S = \{P \vee Q, \neg Q \vee R, \neg Q \vee \neg R, R \vee \neg P, \neg P \vee \neg R\}$ Give a OL-refutation from S with top clause $P \vee Q$.

OL-refutation: Example II

Example (OL-Refutation II)

Consider a set of ordered clauses $S = \{\neg Q(x) \vee P(x), \neg P(a), P(a) \vee Q(x)\}$ Give a OL-refutation from S with top clause $P(a) \vee Q(x)$.

Completeness of OL-refutation

Lemma (OL-refutation ground completeness)

If C is a ground ordered clause in an unsatisfiable set S of ground ordered clauses, and if $S - \{C\}$ is satisfiable, then there exists an OL-refutation from S with top ordered clause C.

Completeness

- OL-refutation ground completeness + lifting lemma
- following theorem

Theorem

If C is an ordered clause in an unsatisfiable set S of ordered clauses, and if $S - \{C\}$ is satisfiable, then there exists an OL-refutation from S with top ordered clause C.

OL-refutation: Example III

Example (OL-Refutation III)

Consider a set of ordered clauses
$S = \{\neg R(x) \vee \neg P(x), \neg Q(x) \vee P(x), \neg P(a), P(a) \vee Q(x)\}$
Give a OL-refutation from S with top clause $\neg R(x) \vee \neg P(x)$.

Linear Deduction and Tree Searching

OL-refutation as tree searching

algorithm

Given S and C_0

- Try to resolve C_0 with every $B_i \in S$
- Obtain $R_1, \cdots, R_m$, every R_i for $i = 1, \cdots, m$ is a possible center clause
- If any R_i is $\square$ then we are done
- Otherwise for each R_i find all possible side clauses that give a resolvent and continue

OL-refutation and tree searching

search tree

- OL-resolution can be seen as a search problem on a tree
- Benefit many efficient search techniques on trees
- Top clause C_0: root
- Side clauses B_i: operators, used to generate successor nodes
- Center clause C: nodes

Searching the tree

searching

- Operators applied to nodes generate immediate successors
- When all immediate successors have been generated a node is expanded
- Search ends when:
  - all possible nodes have been expanded
  - when $\square$ is found

Tree Expansion Algorithm

search algorithm for tree

1. Initialise LIST with the root
2. If list is empty terminate
3. Otherwise pop firt element from the list
4. generate all successors expanding the element (if $\square$ found terminate)
5. insert generated successors in the list (using some order)
6. Go to step 2

Expansion techniques

expansion techniques

- Breadth first
  - always insert successors at the end
  - LIST $\Rightarrow$ Queue
- Depth First
  - always insert successors at the front
  - LIST $\Rightarrow$ Stack
- Heuristics
  - use an ordered List
  - the order specifies the heuristic

Breadth-First Method

algorithm

1. CLIST $= \{C_0\}$
2. If CLIST is empty terminate
3. Otherwise pop the first element C
4. Find all the ordered clause in S that can be side clauses for C. If no such clause exists go to step 2. Otherwise, resolve C with all these side clauses and let $R_1, \cdots, R_m$ denote the ordered resolvents. Let R_i^* be the reduced clause obtained from R_i. If R_i is not reducible then $R_i^* = R_i$.
5. If some R_q^* is $\square$ terminate with a proof. Otherwise put $R_1^*, \cdots, R_m^*$ in an arbitrary order at the end of CLIST
6. Go to step 2

OL-refutation: Example

Example (OL-refutation as tree searching)

Consider the following set $S = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$ of ordered clauses. Give an OL-refutation from S with $C_0 = P \vee Q$ using Breadth First.

algorithm

- Reduce reducible clauses as soon as possible
- Leave tautology for now

Completeness and prof minimality

Definition (minimal proof)

A minimal proof from S with top clause C_0 is an OL-refutation (from S with top clause C_0) that involves the smallest number of resolutions.

discussion

- Breadth First is complete
- Breadth First generates many redundant clauses: not efficient
- Depth First is generally more efficient than Breadth First but not complete
- Depth First with limited bound

OL-refutation: Example

Example (OL-refutation as tree searching)

Consider the following set $S = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$ of ordered clauses. Give an OL-refutation from S with $C_0 = P \vee Q$ using Depth First.

Depth First with limited bound

discussion

- Depth bound: threshold on depth of clauses to expand d^*
- Depth of a clause:
  - $d(C_0) = 0$
  - If R_i is a resolvent of some center clause C then $d(R_i) = d(C) + 1$
- The length of a proof is the depth of $\square$

Depth-First Method (with depth bound)

Depth Bound d^*

1. CLIST $= \{C_0\}$
2. If CLIST is empty terminate
3. Otherwise pop the first element C. If $d(C) > d^*$ go to step 2. Otherwise continue.
4. Find all the ordered clause in S that can be side clauses for C. If no such clause exists go to step 2. Otherwise, resolve C with all these side clauses and let $R_1, \cdots, R_m$ denote the ordered resolvents. Let R_i^* be the reduced clause obtained from R_i. If R_i is not reducible then $R_i^* = R_i$.
5. If some R_q^* is $\square$ terminate with a proof. Otherwise put $R_1^*, \cdots, R_m^*$ in an arbitrary order at the beginning of CLIST
6. Go to step 2

Depth First Example

Example (Depth First)

Consider the following set $S = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$ of ordered clauses. Give a Depth First OL-refutation from S with $C_0 = P \vee Q$ with $d^* = 2$.

Modified Depth-First

Discussion

- Depth first generate all possible successors for each node
- More efficient to expand one successor only
- Modified Depth First: generates only one successor at time

Modified Depth-First

modified depth first

- Find al ordered clause in S that can be side clauses of C_0. If no such clause exists, terminate without a proof. Otherwise, let $B_0^1, \cdots, B_0^r$ such side clauses. Let $CLIST = (C_0, B_0^1), \cdots, (C_0, B_0^r)$.
- If CLIST is empty terminate.
- Otherwise pop the first element (C, B). If $d(C) > d^*$ go to step 2. Otherwise continue.

Modified Depth-First

modified depth first

- Resolve C with B. Let $R_1, \cdots, R_m$ denote the ordered resolvents of C against B. Let R_i^* be the reduced clause obtained from R_i. If R_i is not reducible then $R_i^* = R_i$.
- If some R_q^* is $\square$ terminate with a proof. Otherwise, for each $i = 1, 2, \cdots, m$ find al ordered clause in S that can be side clauses of R_i^*. If no such clause exists, delete R_i^*. Otherwise, let $B_{i_1}^1, \cdots, B_{i_{s_i}}^r$ be such side clauses. Put $(R_i^*, B_{i_1}^1), \cdots, (R_i^*, B_{i_{s_i}}^r)$ in an arbitrary order at the beginning of CLIST
- Go to step 2

Modified Depth First Example

Example (Modified depth first)

Consider the following set
$S = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$ of ordered clauses.
Give a Modified Depth First OL-refutation from S with
$C_0 = P \vee Q$ with $d^* = 2$.

Depth First Characteristics

Discussion

- In general the depth first method searches a smaller tree
- It is not complete
- Depth bound d^*: complete only if the proof is shorter than d^*

Tautology Deletion

Deleting tautology

- Ordered clause C_1 subsumes another ordered clause C_2 iff the ordered clause C_1' obtained removing the underlined literals from C_1 subsumes the ordered clause C_2' obtained removing the underlined literals from C_2.
- An ordered clause C_1 is a tautology iff the clause C_1' obtained removing the underlined literals from C_1 is a tautology (i.e., it contains a complementary pair of literals).
- We can remove tautology from the search by inserting resolvents in *CLIST* only if they are not subsumed by other clause in *CLIST* or if they are not tautologies.

Exercise: OL Breadth First Resolution

Exercise

Find an OL refutation using the breadth first method for the following set of unsat. clauses:

1. $\neg P(x, y) \vee \neg L(x) \vee C(y)$
2. $\neg L(x) \vee P(f(x), x)$
3. $\neg L(x) \vee L(f(x))$
4. $\neg C(x) \vee V(x)$
5. $L(a)$
6. $\neg V(a)$

. Set $C_0 = \neg V(a)$

Would the Depth First method be complete if applied on S ?

If not can we set a d^* such that the Depth First Method is complete on S ?