8a. Reasoning with Horn Clauses
Review

• Lecture 1: What is KR&R
  – KR Hypothesis: Explicit representation of knowledge provides propositional account and causal explanation for intelligent behavior

• Lecture 2: Object-Oriented Representation
  – Frames provide a way to organize knowledge

• Lecture 3-5: Structured Descriptions
  – Adding structure to the definition of objects; sound, complete and efficient reasoning

• Lecture 6: Ontologies
  – Engineering discipline of deciding which class, function and relation symbols to use in representing a domain

• Lecture 7: Knowledge Representation in Social Context
  – KR&R concepts for the Web
Next Four Lectures

- Frames and structured descriptions provide useful subsets of FOL
  - Their expressive power, however, is limited
- In lectures 8 through 11, we will study more expressive representations
  - Reasoning with Horn Clauses
    - Foundation for logic programming family of languages
  - Procedural control of reasoning
    - Negation as Failure - a practical alternative to classical negation
  - Production Systems
    - Foundation of expert systems / rule-based systems
  - Advanced logics
    - Combining rules with object-oriented and structured representations, higher order logic, modal logic
  - Non Monotonic Reasoning
    - Representing default knowledge, answer set programming
Expressive Overlaps among KRs

First-Order Logic

Description Logic

Horn Logic Programs

Logic Programs

Non-Monotonic Reasoning (Procedural Attachments)

Description Logic Programs

Copyright 2009 by Vulcan Inc., Benjamin Grosof, Mike Dean, and Michael Kifer. All Rights Reserved.
Reasoning with Horn Clauses

- Definitions
- SLD Resolution
- Forward and Backward Chaining
- Efficiency of reasoning with Horn Clauses
- Horn FOL vs Horn LP
Definitions

• Term
• Formula
• Atomic Formula
• Sentence
• Literal
• Clause
Definitions

• Term
  – The set of terms of FOL is the least set satisfying these conditions:
    • every variable is a term
    • if t1 . . . . tn are terms, and f is a function symbol of arity n, then f(t1 . . . . tn) is a term

• Formula
  – The set of formulas of FOL is the least set satisfying these constraints:
    • if t1 . . . . tn are terms, and P is a predicate symbol of arity n, then P(t1 . . . . tn) is a formula;
    • if t1 and t2 are terms, then t1=t2 is a formula;
    • if α and β are formulas, and x is a variable, then ¬α, α ∨ β, α ∧ β, x α, and Exists α, are formulas.

• Atomic Formula
  – Formulas of first two types above

• Sentence
  – Any formula with no free variables

• Literal
  – Atomic formula or its negation

• Clause
  – A finite set of literals
Resolution

For the premises \((p \implies q)\) and \((q \implies r)\), we want to prove \((p \implies r)\)

1. \(\neg p, q\) Premise
2. \(\neg q, r\) Premise
3. \(p\) Negated Goal
4. \(\neg r\) Negated Goal
5. \(q\) 3, 1
6. \(4\) 5, 2
7. \(\{}\) 6, 4
Horn clauses

Clauses are used two ways:
- as disjunctions: \((\text{rain} \lor \text{sleet})\)
- as implications: \((\neg \text{child} \lor \neg \text{male} \lor \text{boy})\)

Here focus on 2nd use

Horn clause = at most one +ve literal in clause
- positive / definite clause = exactly one +ve literal
  \(e.g. [\neg p_1, \neg p_2, \ldots, \neg p_m, q]\)
- negative clause = no +ve literals (also, referred to as integrity constraints)
  \(e.g. [\neg p_1, \neg p_2, \ldots, \neg p_n]\) and also \([\ ]\)

Note: \([\neg p_1, \neg p_2, \ldots, \neg p_n, q]\) is a representation for
\((-p_1 \lor -p_2 \lor \ldots \lor -p_n \lor q)\) or \([(p_1 \land p_2 \land \ldots \land p_n) \Rightarrow q]\)

so can read as: If \(p_1\) and \(p_2\) and \(\ldots\) and \(p_n\) then \(q\)

and write as: \(p_1 \land p_2 \land \ldots \land p_n \Rightarrow q\) or \(q \Leftarrow p_1 \land p_2 \land \ldots \land p_n\)
Resolution with Horn clauses

Only two possibilities:

Neg  Pos
   Neg

Pos  Pos
   Pos

It is possible to rearrange derivations of negative clauses so that all new derived clauses are negative

\[ [-a, \neg q, p] [-b, q] \]
\[ [-c, \neg p] [p, \neg a, \neg b] \]
\[ [-a, \neg b, \neg c] \]  \[ [-c, \neg p] [-a, \neg q, p] \]
\[ [-a, \neg c, \neg q] [-b, q] \]
\[ [-a, \neg b, \neg c] \]
derived positive clause to eliminate
Further restricting resolution

Can also change derivations such that each derived clause is a resolvent of the previous derived one (negative) and some positive clause in the original set of clauses

- Since each derived clause is negative, one parent must be positive (and so from original set) and one parent must be negative.
- Chain backwards from the final negative clause until both parents are from the original set of clauses
- Eliminate all other clauses not on this direct path
Example 1

KB

FirstGrade
FirstGrade ⊨ Child
Child ∧ Male ⊨ Boy
Kindergarten ⊨ Child
Child ∧ Female ⊨ Girl
Female

Show that KB ⊨ Girl

[FirstGrade]
[¬Child, ¬Male, Boy]
[¬FirstGrade, Child]
[Kindergarten, Child]
[¬Child, ¬Female, Girl]
[Female]
[¬Girl, ¬Female]
[Girl]
[¬Girl]

Derivation has 9 clauses, 4 new
SLD version of Example 1

KB

FirstGrade
FirstGrade ⊃ Child
Child ∧ Male ⊃ Boy
Kindergarten ⊃ Child
Child ∧ Female ⊃ Girl
Female

Show that KB |= Girl

[¬Child, ¬Female, Girl]

¬Girl

[Female]

[¬Child, ¬Female]

[¬FirstGrade, Child]

[¬Child]

[FirstGrade]

[¬FirstGrade]

[ ]
An SLD-derivation of a clause $c$ from a set of clauses $S$ is a sequence of clause $c_1, c_2, \ldots, c_n$ such that $c_n = c$, and

1. $c_1 \in S$
2. $c_{i+1}$ is a resolvent of $c_i$ and a clause in $S$

Write: $S \xrightarrow{\text{SLD}} c$

Note: SLD derivation is just a special form of derivation and where we leave out the elements of $S$ (except $c_1$)

In general, cannot restrict ourselves to just using SLD-Resolution

Proof: $S = \{[p, q], [p, \neg q], [\neg p, q], [\neg p, \neg q]\}$. Then $S \rightarrow []$.

Need to resolve some $[\rho]$ and $[\neg \rho]$ to get $[]$.

But $S$ does not contain any unit clauses.

So will need to derive both $[\rho]$ and $[\neg \rho]$ and then resolve them together.
Completeness of SLD

However, for Horn clauses, we can restrict ourselves to SLD-Resolution

**Theorem:** SLD-Resolution is refutation complete for Horn clauses: \( H \rightarrow [] \) iff \( H \overset{\text{SLD}}{\rightarrow} [] \)

So: \( H \) is unsatisfiable iff \( H \overset{\text{SLD}}{\rightarrow} [] \)

This will considerably simplify the search for derivations

**Note:** in Horn version of SLD-Resolution, each clause in the
\( c_1, c_2, \ldots, c_n \), will be negative

So clauses \( H \) must contain at least one negative clause, \( c_1 \)
and this will be the only negative clause of \( H \) used.

Typical case:

- KB is a collection of positive Horn clauses
- Negation of query is the negative clause
Example 1 (again)

KB

FirstGrade
FirstGrade ⊨ Child
Child ∧ Male ⊨ Boy
Kindergarten ⊨ Child
Child ∧ Female ⊨ Girl
Female

SLD derivation

[¬Girl]
[¬Child, ¬Female]
[¬Child]
[¬FirstGrade]
[]

Alternate representation

A goal tree whose nodes are atoms, whose root is the atom to prove, and whose leaves are in the KB.

Show KB ∪ {¬Girl} unsatisfiable
Back-chaining procedure

\[
\text{Solve}[q_1, q_2, \ldots, q_n] = \quad \text{/* to establish conjunction of } q_i \text{ */}
\]

If \( n=0 \) then return \textbf{YES}; \quad \text{/* empty clause detected */}

For each \( d \in \text{KB} \) do

\[
\begin{align*}
\text{If} \quad d & = [q_1, \neg p_1, \neg p_2, \ldots, \neg p_m] \quad \text{/* match first } q_i \text{ */} \\
\text{and} & \quad \text{/* replace } q \text{ by -ve lits */} \\
\text{Solve}[p_1, p_2, \ldots, p_m, q_2, \ldots, q_n] & \quad \text{/* recursively */}
\end{align*}
\]

then return \textbf{YES}

end for;

end if; \quad \text{/* can't find a clause to eliminate } q \text{ */}

Return \textbf{NO}

Depth-first, left-right, back-chaining

- depth-first because attempt \( p_i \) before trying \( q_i \)
- left-right because try \( q_i \) in order, 1, 2, 3, …
- back-chaining because search from goal \( q \) to facts in KB \( p \)

This is the execution strategy of Prolog

First-order case requires unification etc.
Problems with back-chaining

Can go into infinite loop

- tautologous clause: \([p, \neg p]\) (corresponds to Prolog program with \(p : - p\)).

Previous back-chaining algorithm is inefficient

Example: Consider 2n atoms, \(p_0, ..., p_{n-1}, q_0, ..., q_{n-1}\) and 4n-4 clauses

\((p_{i-1} \Rightarrow p_i), (q_{i-1} \Rightarrow p_i), (p_{i-1} \Rightarrow q_i), (q_{i-1} \Rightarrow q_i)\).

With goal \(p_k\) the execution tree is like this

Is this problem inherent in Horn clauses?
Forward-chaining

Simple procedure to determine if Horn KB $\models q$.

**main idea:** mark atoms as solved

1. If $q$ is marked as solved, then return **YES**
2. Is there a $\{p_1, \neg p_2, \ldots, \neg p_n\} \in$ KB such that $p_2, \ldots, p_n$ are marked as solved, but the positive lit $p_1$ is not marked as solved?
   - no: return **NO**
   - yes: mark $p_1$ as solved, and go to 1.

**FirstGrade example:**

Marks: FirstGrade, Child, Female, Girl then done!

**Observe:**

- only letters in KB can be marked, so at most a linear number of iterations
- not goal-directed, so not always desirable
- a similar procedure with better data structures will run in *linear* time overall

**Note:** FirstGrade gets marked since all the negative atoms in the clause (none) are marked
First-order undecidability

Even with just Horn clauses, in the first-order case we still have the possibility of generating an infinite branch of resolvents.

KB:

\[ \text{LessThan}(\text{succ}(x), y) \implies \text{LessThan}(x, y) \]

Query:

\[ \text{LessThan}(\text{zero}, \text{zero}) \]

As with full Resolution, there is no way to detect when this will happen

There is no procedure that will test for the satisfiability of first-order Horn clauses
the question is undecidable

As with non-Horn clauses, the best that we can do is to give control of the deduction to the user

to some extent this is what is done in Prolog, but we will see more in “Procedural Control”
Horn FOL vs Horn LP

- In Horn LP, the conclusions are limited to ground atomic formulas. For example:
  - Suppose, we have:
    - DangerousTo(?x,?y) ← PredatorAnimal(?x) ∧ Human(?y);
    - PredatorAnimal(?x) ← Lion(?x)
    - Lion(Simba)
    - Human(Joey)

  -- In Horn LP, we can derive:
    - I1 = {Lion(Simba), Human(Joey)}
    - I2 = {PredatorAnimal(Simba), Lion(Simba), Human(Joey)}
    - I3 = {DangerousTo(Simba,Joey), PredatorAnimal(Simba), Lion(Simba), Human(Joey)}

  -- In Horn FOL, we will also derive:
    - DangerousTo(Simba,?y) ← Human(?y)
    - ¬Human(?y) ← ¬DangerousTo(Simba,?y).

- Horn LP is the foundation of logic programming and Prolog

1. Example adapted from Grosof, Kifer & Dean
Recommended Reading

- Chapter 5 of Brachman & Levesque textbook