Logical Inference and Rule-based reasoning

AI & SW History (non-scholarly)

Rule based reasoning has a long history in AI and SW

- 1970s: Prolog programming language based on rules and used for many AI systems in Europe
- 1980s: the expert systems paradigm mostly used rule-based reasoning; Japan embraced Prolog for its 5th generation computing systems push
- 2000s: semantic web promoted OWL, based on description logic KR rather than rules; SWRL rule system fails to become a W3C recommendation
- 2010s: various rule systems developed for RDF: RIF, RuleML, SPIN, Datalog
- 2022: SWRL is still commonly supported and used

Automated inference for FOL

- Automated inference for FOL is harder than Propositional Logic
  - Variables can take on an infinite number of possible values from their domains
  - Hence there are potentially an infinite number of ways to apply the Universal Elimination rule
- Godel's Completeness Theorem says FOL entailment is only **semi-decidable**
  - If a sentence is true given a set of axioms, there is a procedure that will eventually determine this
  - If a sentence is false, there's no guarantee a procedure will ever discover this — it may never halt

Generalized Modus Ponens (GMP)

- Modus Ponens: **P, P=>Q |= Q**
- Generalized Modus Ponens extends this to rules in FOL
- Combines And-Introduction, Universal-Elimination, and Modus Ponens, e.g.
  - given P(c) , Q(c) , $\forall$x P(x)$\wedge$Q(x) $\rightarrow$ R(x)
  - derive R(c)
- Must deal with
  - more than one condition on rule's left side
  - variables

Often rules restricted to Horn clauses

- A Horn clause is a sentence of the form:

  $P_1(\ldots) \wedge P_2(\ldots) \wedge \ldots \wedge P_n(\ldots) \rightarrow Q(\ldots)$

  where
  - ≥ 0 P_is and 0 or 1 Q
  - P_is and Q are positive (i.e., non-negated) literals
  - The … are a sequence of variables or literals
- Prolog and most rule-based systems are limited to Horn clauses
- Horn clauses are a subset of all FOL sentences

Horn clauses 2

- Special cases
  - Typical rule: $P_1 \wedge P_2 \wedge \ldots P_n \rightarrow Q$
  - Constraint: $P_1 \wedge P_2 \wedge \ldots P_n \rightarrow$ false
  - A fact: $\rightarrow Q$
  - A goal: $Q \rightarrow$
- Examples
  - parent(P1,P2) $\wedge$ parent(P2,P3) $\rightarrow$ grandparent(P1,P3)
  - male(X) $\wedge$ female(X) $\rightarrow$ false
  - $\rightarrow$ male(john)
  - female(mary) $\rightarrow$

Horn clauses 3

- These are not Horn clauses:
  - married(x, y) $\rightarrow$ loves(x, y) $\vee$ hates(x, y)
  - $\neg$ likes(john, mary)
  - $\neg$ likes(x, y) $\rightarrow$ hates(x, y)
- Can't assert/conclude disjunctions (i.e., "or")
- Can't have "true" negation
  - Though some systems, like Prolog, allow a negation operator that means "can't prove"
- No wonder Horn clause reasoning is easier

Horn clauses 3

- Where are the quantifiers?
  - Variables in conclusion **universally quantified**
  - Variables appearing only in premises **existentially quantified**
- Examples:
  - parentOf(P,C) $\rightarrow$ childOf(C,P)
    $\forall$P $\forall$C parentOf(P,C) $\rightarrow$ childOf(C,P)
  - parentOf(P,X) $\rightarrow$ isParent(P)
    $\forall$P $\exists$X parent(P,X) $\rightarrow$ isParent(P)
  - parent(P1, X) $\wedge$ parent(X, P2) $\rightarrow$ grandParent(P1, P2)
    $\forall$P1,P2 $\exists$X parent(P1,X) $\wedge$ parent(X, P2) $\rightarrow$ grandParent(P1, P2)

Definite Clauses

- A **definite clause** is a horn clause with a conclusion
- What's not allowed is a horn clause w/o a conclusion, e.g.
  - male(x), female(x) $\rightarrow$
  - i.e., ~male(x) $\vee$ ~female(x)
- Most rule-based reasoning systems, like Prolog, allow only definite clauses in the KB

Limitations

- Most rule-based reasoning systems use only definite horn clauses
  - Limited ability to reason about **negation** and **disjunction**
  - Cannot have try negation in a rule
  - Cannot have rules whose conclusion is an or
- These are required for some problems (e.g., playing Clue)
- Benefit is **decidability** and **efficiency**
- Some limitations can be overcome by
  - Adding procedural components
  - Augmenting with other reasoners

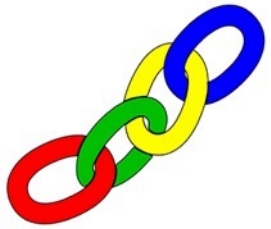
Some non-limitations

- We can easily rewrite some non Horn clause rules to as several simpler Horn clauses
- Example, rewrite 1 as 2
  1. $married(x, y) \rightarrow loves(x, y) \wedge knows(x, y)$
  2. $married(x, y) \rightarrow loves(x, y)$
     $married(x, y) \rightarrow knows(x, y)$
- Example, rewrite 1 as 2
  1. $loves(x, y) \vee likes(x,y) \rightarrow knows(x, y)$
  2. $loves(x, y) \rightarrow knows(x, y)$
     $likes(x, y) \rightarrow knows(x, y)$
- Additional rewriting rules are available

Forward & Backward Reasoning

- We often talk about two reasoning strategies:
  - Forward chaining and
  - Backward chaining
- Both are equally powerful, but optimized for different use cases
- You can also have a mixed strategy
  - Each rule specifies how it's to be used

Forward chaining



- Proofs start with given axioms/premises in KB, deriving new sentences using GMP until the goal/query sentence is derived
  - Process follows a chain of rules and facts going from the KB to the conclusion
- This defines a forward-chaining inference procedure because it moves "forward" from the KB to the goal [eventually]
- Inference using GMP is sound and complete for KBs containing only Horn clauses

Forward chaining example

- KB:
  1. allergies(X) $\rightarrow$ sneeze(X)
  2. cat(Y) $\wedge$ allergicToCats(X) $\rightarrow$ allergies(X)
  3. cat(felix)
  4. allergicToCats(mary)
- Goal:
  - sneeze(mary)

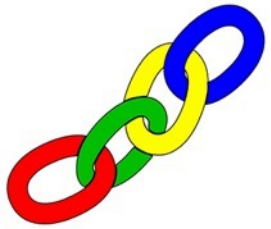
Forward chaining example

- KB:
  1. allergies(X) $\rightarrow$ sneeze(X)
  2. cat(Y) $\wedge$ allergicToCats(X) $\rightarrow$ allergies(X)
  3. cat(felix)
  4. allergicToCats(mary)
  5. #3 , #4 $\rightarrow$ allergies(mary)
  6. allergies(mary)
  7. #6 , #1 $\rightarrow$ sneeze(mary)
  8. sneeze(mary)
- Goal:
  - sneeze(mary)

In practice…

- Most forward chaining systems compute and add to the KB **all** inferred facts
  - This is generally OK for Horn clause systems, as there will be a finite number of them
  - But could be a problem if the KB is large
- Another problem: suppose you want to remove a fact or rule?
  - You could delete all of the inferred facts and start over, or
  - Use a truth maintenance system to only remove inferred facts that depended on the deleted items

Backward chaining



- Backward-chaining deduction using GMP is also complete for KBs containing only Horn clauses
- Proofs start with the goal query, find rules with that conclusion, and then tries to prove each of the antecedents in the rule
- Keep going until you reach premises
- Avoid loops by checking if new subgoal is already on the goal stack
- Avoid repeated work: use a cache to check if new subgoal already proved true or failed

Backward chaining example

- KB:
  - allergies(X) $\rightarrow$ sneeze(X)
  - cat(Y) $\wedge$ allergicToCats(X) $\rightarrow$ allergies(X)
  - cat(felix)
  - allergicToCats(mary)
- Goal:
  - sneeze(mary)

Backward chaining example

- KB:
  1. allergies(X) $\rightarrow$ sneeze(X)
  2. cat(Y) $\wedge$ allergicToCats(X) $\rightarrow$ allergies(X)
  3. cat(felix)
  4. allergicToCats(mary)
- Goal:
  - sneeze(mary) if allergies(mary)
  - allergies(mary) if cat(Y) $\wedge$ allergicToCats(mary)
  - allergies(mary) if cat(Y) $\wedge$ cat(felix)
  - allergies(mary)
  - sneeze(mary)

In practice…

- Backward chaining is very common and used in Prolog and many other rule-based systems
- It does not have a problem if the facts of rules change
- It can be less efficient if you want to prove the same conclusion multiple times

Forward vs. backward chaining

- Forward chaining is data-driven
  - Automatic, unconscious processing, e.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
  - Efficient when you want to compute **all conclusions**
- Backward chaining is goal-driven, better for problem-solving and query answering
  - Where are my keys? How do I get to my next class?
  - Complexity can be much less than linear wrt KB size
  - Efficient when you want **one or a few conclusions**
  - Good where the underlying facts are changing

Mixed strategy

- Many practical reasoning systems do both forward and backward chaining
- How you encode rule determines how it's used:

```
spouse(X,Y) => spouse(Y,X)              % forward chaining
father(X,Y) <= parent(X,Y), male(X)     % backward chaining
```

- Forward chaining rules useful if you want to draw conclusions and **materialize them as facts**
  - This also easily avoid loops, e.g., don't trigger forward chaining if the fact already exists
- Given a model of your rules and the kind of reasoning needed, you can decide which to encode as FC and which as BC rules

Completeness of GMP

- GMP (using forward or backward chaining) is complete for KBs containing only Horn clauses
- not complete for simple KBs with non-Horn clauses
- What is entailed by the following sentences?

1. $(\forall x)\ P(x) \rightarrow Q(x)$
2. $(\forall x)\ \neg P(x) \rightarrow R(x)$
3. $(\forall x)\ Q(x) \rightarrow S(x)$
4. $(\forall x)\ R(x) \rightarrow S(x)$

Completeness of GMP

- The following entail that S(A) is true:

  1.$(\forall x)\ P(x) \rightarrow Q(x)$

  2.$(\forall x)\ \neg P(x) \rightarrow R(x)$

  3.$(\forall x)\ Q(x) \rightarrow S(x)$

  4.$(\forall x)\ R(x) \rightarrow S(x)$

- If we want to conclude S(A), with GMP we can't, since the second one isn't a Horn clause
- It is equivalent to $P(x) \vee R(x)$

How about Prolog?

- Prolog syntax is a bit different, putting the rule's conclusion first

  hasMother(?x, ?m) :- hasParent(?x, ?m), female(?m) .

  head = conclusion body = conjunction of conditions

- A fact is a rule w/o a body (i.e., no conditions)

  hasParent(john, tom).

  hasParent(john, mary).

  female(mary).

- Prolog 'proves' queries by matching a fact, or a rule's conclusion and then proving each condition in the rule's body

Can we do this in Prolog?

Try encoding this in Prolog

1. q(X) :- p(X).
2. r(X) :- neg(p(X)).
3. s(X) :- q(X).
4. s(X) :- r(X).

1. $(\forall x)\ P(x) \rightarrow Q(x)$
2. $(\forall x)\ \neg P(x) \rightarrow R(x)$
3. $(\forall x)\ Q(x) \rightarrow S(x)$
4. $(\forall x)\ R(x) \rightarrow S(x)$

- We should not use **\+** or **not** (in SWI) for negation since it means "**negation as failure**"
- i.e., selective use of the closed world assumption or "if I can't prove it, it must be false"
- Prolog explores possible proofs independently
- It can't take a larger view and realize that one branch must be true since $p(x) \vee \sim p(x)$ is always true

Fin