COMP4418: Knowledge Representation and Reasoning
Procedural Control

Maurice Pagnucco
School of Computer Science and Engineering

COMP4418, Week 3
Declarative / procedural

Theorem proving (like resolution) is a general domain-independent method of reasoning
Does not require the user to know how knowledge will be used
  • will try all logically permissible uses
Sometimes have ideas about how to use knowledge, how to search for derivations
  • do not want to use arbitrary or stupid order
Want to communicate to ATP procedure guidance based on properties of domain
  • perhaps specific method to us
  • perhaps merely method to avoid
Example: directional connectives
In general: control of reasoning

B&L (2005)
DB + rules

Can often separate (Horn) clauses into two components:

- database of facts
  - basic facts of the domain
  - usually ground atomic wffs

- collection of rules
  - extend vocabulary in terms of basic facts
  - usually universally quantified conditionals

Both retrieved by unification matching

Example:

MotherOf(jane,billy)
FatherOf(john,billy)
FatherOf(sam,john)
...
ParentOf(x,y) ← MotherOf(x,y)
ParentOf(x,y) ← FatherOf(x,y)
ChildOf(x,y) ← ParentOf(y,x)
...

Control Issue: how to use rules
Rule formulation

Consider AncestorOf in terms of ParentOf
Three logically equivalent versions:

1. \( \text{AncestorOf}(x,y) \iff \text{ParentOf}(x,y) \)
   \( \text{AncestorOf}(x,y) \iff \text{ParentOf}(x,z) \land \text{AncestorOf}(z,y) \)

2. \( \text{AncestorOf}(x,y) \iff \text{ParentOf}(x,y) \)
   \( \text{AncestorOf}(x,y) \iff \text{ParentOf}(z,y) \land \text{AncestorOf}(x,z) \)

3. \( \text{AncestorOf}(x,y) \iff \text{ParentOf}(x,y) \)
   \( \text{AncestorOf}(x,y) \iff \text{AncestorOf}(x,z) \land \text{AncestorOf}(z,y) \)

Back-chaining goal of AncestorOf(sam,sue) will ultimately reduce to set of ParentOf(−,−) goals

1. get \( \text{ParentOf}(\text{sam},z) \): find child of Sam
   searches downward from Sam

2. get \( \text{ParentOf}(z,\text{sue}) \): find parent of Sue
   searches upward from Sue

3. get \( \text{ParentOf}(\text{−},\text{−}) \): find parent relations
   searches in both directions

Search strategies are not equivalent
if more than 2 children per parent, (2) is best
Algorithm design

Example: Fibonacci numbers

\[1, 1, 2, 3, 5, 8, 13, 21, \ldots\]

Version 1:

\[
\text{Fibo}(0, 1) \\
\text{Fibo}(1, 1) \\
\text{Fibo}(s(s(n)), x) \iff \text{Fibo}(n, y) \land \text{Fibo}(s(n), z) \land \text{Plus}(y, z, x)
\]

Requires *exponential* number of Plus subgoals

Version 2:

\[
\text{Fibo}(n, x) \iff F(n, 1, 0, x) \\
F(0, c, p, c) \\
F(s(n), c, p, x) \iff \text{Plus}(p, c, s) \land F(n, s, c, x)
\]

Requires only *linear* number of Plus subgoals

B&L (2005)
Ordering goals

Example:

\[ \text{AmericanCousinOf}(x,y) \iff \text{American}(x) \land \text{CousinOf}(x,y) \]

In back-chaining, can try to solve either subgoal first

Not much difference for

\[ \text{AmericanCousinOf}(\text{fred}, \text{sally}) \]

Big difference for

\[ \text{AmericanCousinOf}(x, \text{sally}) \]

1. find an American and then check to see if she is a cousin of Sally
2. find a cousin of Sally and then check to see if she is an American

So want to be able to order goals

better to *generate* cousins and test for American

In Prolog: order clauses, and literals in them

- Notation:  \( G :\neg G_1, G_2, \ldots, G_n \) stands for  \( G \iff G_1 \land G_2 \land \ldots \land G_n \)
- but goals are attempted in presented order
Commit

Need to allow for backtracking in goals

\[
\text{AmericanCousinOf}(x,y) :- \text{CousinOf}(x,y), \text{American}(x)
\]

For goal AmericanCousinOf(x,sally), may need to try American(x) for various values of \(x\)

But sometimes, given clause of the form

\[
G :- T, S
\]

goal \(T\) is needed only as a *test* for the applicability of subgoal \(S\)

In other words: if \(T\) succeeds, commit to \(S\) as the *only* way of achieving goal \(G\).

so if \(S\) fails, then \(G\) is considered to have failed

- do not look for other ways of solving \(T\)
- do not look for other clauses with \(G\) as head

In Prolog: use of cut symbol

Notation: \(G :- T_1, T_2, \ldots, T_m,!, G_1, G_2, \ldots, G_n\)

attempt goals in order, but if all \(T_i\) succeed, then commit to \(G_i\)
If-then-else

Sometimes inconvenient to separate clauses in terms of unification, as in
\[ G(\text{zero}, -) : \text{- method 1} \]
\[ G(\text{succ}(n), -) : \text{- method 2} \]
For example, might not have distinct cases:
\[ \text{NumberOfParentsOf}(\text{adam}, 0) \]
\[ \text{NumberOfParentsOf}(\text{eve}, 0) \]
\[ \text{NumberOfParentsOf}(x, 2) \]
want: 2 for everyone except Adam and Eve
Or cases may split based on computed property:
\[ \text{Expt}(a, n, x) : \text{- Even}(n), (\text{what to do when } n \text{ is even}) \]
\[ \text{Expt}(a, n, x) : \text{- Even}(s(n)), (\text{what to do when } n \text{ is odd}) \]
want: check for even numbers only once
Solution: use ! to do if-then-else
\[ G : \text{- } P, !, Q. \]
\[ G : \text{- } R. \]
To achieve \( G \): if \( P \) then use \( Q \) else use \( R \).
\[ \text{Expt}(a, n, x) : \text{- Even}(n), !, (\text{for even } n) \]
\[ \text{Expt}(a, n, x) : (\text{for odd } n) \]
\[ \text{NumberOfParentsOf}(\text{adam}, 0) : \text{- } ! \]
\[ \text{NumberOfParentsOf}(\text{eve}, 0) : \text{- } ! \]
\[ \text{NumberOfParentsOf}(x, 2) \]
Controlling backtracking

Consider a goal

So goal should be:
   \text{AncestorOf}(jane, billy), \neg, \text{Male}(jane)

Similarly:

\text{Member}(x, l) \iff \text{FirstElement}(x, l)
\text{Member}(x, l) \iff \text{Rest}(l, l') \land \text{Member}(x, l')

If only to be used for testing, want
   \text{Member}(x, l) \colon\!- \text{FirstElement}(x, l), \neg

On failure, do not try to find another $x$ later in rest of list
Negation as failure

Procedurally: can distinguish between

- can solve goal \( \neg G \)
- cannot solve \( G \)

Use \( \text{not}(G) \) to mean goal that succeeds if \( G \) fails, and fails if \( G \) succeeds

roughly:

\[
\text{not}(G) :- G, !, \text{fail} /* \text{fail if } G \text{ succeeds */}\]
\[
\text{not}(G) /* \text{otherwise succeed */}\]

Only terminates when failure is \( \text{finite} \)

no more resolvents vs. infinite branch

Useful when DB + rules is complete

\[
\text{NoParents}(x) :- \text{not(ParentOf}(z,x))\]

or when method already exists for complement

\[
\text{Composite}(n) :- \text{not(PrimeNum}(n))\]

Declaratively: same reading as \( \neg \), but complications with \textit{new} variables in \( G \)

\[
[\text{not(ParentOf}(z,x)) \rightarrow \text{NoParents}(x)]
\]

vs. \( \neg \text{ParentOf}(z,x) \rightarrow \text{NoParents}(x)\]