Remembering Resolution

If our program represents all sentences about the world in **clause form**:

$$p_1 \textbf{ or } p_2 \ldots \textbf{ or } p_n$$

then it can use **resolution**:

$$\begin{array}{ll} p & \textbf{or } \text{disjunction}_1 \\ \underline{[\ \textbf{not } p\ ]} & \underline{\textbf{or } \text{disjunction}_2} \\ \text{disjunction}_1 & \textbf{or } \text{disjunction}_2 \end{array}$$

to infer new facts about the world or to verify some fact.

Inferring new facts is as simple as using resolution to derive new sentences. Generally, shorter sentences are more useful.

It can verify some fact by constructing a **proof by contradiction**, showing that if the fact were wrong a contradiction would result.

A Simple Domain?

Here are some facts about one little paddock of the world:

1. Horses and cows are mammals.

```
(not horse(x)) or mammal(x)
(not cow(x))   or mammal(x)
```

2. An offspring of a horse is a horse.

```
(not offspring(x, y)) or
       (not horse(y)) or horse(x)
```

3. Bluebeard is a horse.

```
horse(Bluebeard)
```

4. Bluebeard is Charley’ s parent.

```
parent(Bluebeard, Charley)
```

5. Offspring and parent are inverse relations.

```
(not offspring(x, y)) or parent(y, x)
(not parent(x, y)) or offspring(y, x)
```

6. Every mammal has a parent.

```
(not mammal(x)) or parent( ParentOf(x), x)
```

Complication #1

Stating #6 requires us to talk about the *existence* of some individual, not all individuals. We can use a function to compute that object:

```
(not mammal(x)) or parent( ParentOf(x), x)
```

`ParentOf` is called a *Skolem function*. We create and use Skolem functions as intermediate helpers when dealing with existence claims.

Complication #2

Suppose that an agent using resolution wants to decide whether there is a horse. In the world.

We also need a Skolem function to state this assertion — but not to do the proof!

The natural way to express this assertion is:

exists horse(x)

But that sentence is not in clause form.
However, it is equivalent to:

not not exists x**.**horse(x)
not (not exists x**.**horse(x)**)**
not (forall x **not** horse(x)**)**

The inverse of this claim is...

not not (forall x **not** horse(x)**)**
forall x **not** horse(x)
not horse(x)

A Quick Proof

Show that:

i. Charley is a horse. Contradict: **not** horse(Charley)

1. **(not** horse(x)**) or** mammal(x)
2. **(not** cow(x)**) or** mammal(x)
3. **(not** offspring(x, y)**) or (not** horse(y)**) or** horse(x)
4. horse(Bluebeard)
5. parent(Bluebeard, Charley)
6. **(not** offspring(x, y)**) or** parent(y, x)
7. **(not** parent(x, y)**) or** offspring(y, x)
8. **(not** mammal(x)**) or** parent(ParentOf(x), x)

9. **not** horse(Charley) <assumed>

10. **(not** offspring(Charley, y)**) or (not** horse(y)**)** < 9, 3>
11. **(not** parent(y, Charley)**) or (not** horse(y)**)** <10, 7>
12. **(not** horse(Bluebeard)**)** <11, 5>
13. **FALSE** <12, 4>

not horse(Charley) is false,
so horse(Charley) must be true

Another Quick Proof

Show that:

j. There is a horse. Contradict: **not** horse(x)

1. **(not** horse(x)**) or** mammal(x)
2. **(not** cow(x)**) or** mammal(x)
3. **(not** offspring(x, y)**) or (not** horse(y)**) or** horse(x)
4. horse(Bluebeard)
5. parent(Bluebeard, Charley)
6. **(not** offspring(x, y)**) or** parent(y, x)
7. **(not** parent(x, y)**) or** offspring(y, x)
8. **(not** mammal(x)**) or** parent(ParentOf(x), x)

9. **not** horse(x) <assumed>

10. **FALSE** < 9, 4>

not horse(x) is false,
so **exists** x.horse(x) must be true.

Difficulties in Automating Proofs: Unifying Variables

Our simple query **Is Charley a horse?** demonstrates the concept of unifying the variables and constants that occur in two sentences:

```
 7. (not parent(x, y)) or
    offspring(y, x)
10. (not offspring(Charley, y)) or
    (not horse(y))
  .
  .
11. (not parent(y, Charley)) or
    (not horse(y))
```

Programs are much better at handling these sorts of issues. A solution that works most of the time is to **standardize the variables apart**, so that each sentence uses different variables. Then you can create a clear binding that makes the sentences work.

```
 7. (not parent(a, b)) or offspring(b, a)
10. (not offspring(Charley, y)) or
    (not horse(y))
  .
  .     [ b = Charley   a = y ]
  .
11. (not parent(y, Charley)) or
    (not horse(y))
```

Difficulties in Automating Proofs: Conflict Resolution

The query **Is there a horse?** seems trivial. Of course, there is. Sentence 4 says so.

But we had to know to use Sentence 4 to make the proof obvious. How would our program, which does not have our horse sense, do the same?

The problem is that there is a **third** sentence we could resolve <9> with: **<3>**.

```
3. (not offspring(x, y)) or
   (not horse(y)) or horse(x)
4. horse(Bluebeard)
9. horse(Charley)

10. not horse( x )
```

This is the problem of **conflict resolution**:

Which of many possibilities do we consider first?

Techniques include:

- syntactic solutions (**try the sentences in order**)
- semantic solutions, which use knowledge to decide.

Conflict resolution calls for the same trade-off between brute force and knowledge we encountered in search...

An Exercise

Okay, so Charley is a horse.

Suppose that our resolution-based agent just told my daughter Ellen so.

As Ellen often does, she says, “Why?”

What should the agent say?

Your answer should include:

- the answer the agent should say, and
- how the agent would decide what to say.

Remember: the agent isn’t you. It knows only what is in its program and its database...

For Better or For Worse

Earlier today, someone asked me,

"Why did 'Joe' skip 053 class on Tuesday?"

I had two possible answers at my disposable:

Answer 1: He was working on some other project.
Answer 2: We had an exam in 053 the previous session.

Which is the better answer? Why?

Write a list of at least three criteria we can use to compare competing explanations such as these. Each criterion might be of a form similar to:

"If A is more ????? than B,
then A is the better explanation."

Good Explanations

Good explanations...

- ... tell the right story at the right time.
- ... contain neither too many details nor too few.

The context of the question is critical. I would give different answers to the question about Joe if it were asked by:

- my wife
- a classmate of Joe's
- another CS prof
- my five-year-old daughter

How do we make these ideas concrete enough that we can implement them in a computer program?

Creating Good Explanations

Early attempts in explanation came out of the logic- and rule-based systems research...

Explanations of this sort:

> "Why do you think that the patient is anemic?"
>
> > "Because the patient's blood sample shows high Fe+ content."
>
> "Why did you ask for the patient's temperature?"
>
> > "Because a temperature of 100 degrees or more indicates anemia."

...were generated from a trace of the rules used to derive the fact being asked about:

> IF BLOOD SAMPLE shows high Fe+ content
> THEN ANEMIA is likely.
>
> IF TEMPERATURE = high
> THEN ANEMIA is likely.

A Rule-Based Example

Rules

If ?x is a lawyer, then ?x is rich.

If ?x is rich and ?x owns a house ?h, then ?h is big.

If ?h is big, then the owner of ?h has to do a lot of yard work.

Data Base

John is a lawyer.

John owns a house.

Conclusion: John has to do a lot of yard work.

Why?

What Makes an Explanation Good?

Good explanations ground out in facts.

Good explanations give the questioner:

- facts she doesn't know.
- rules she doesn't know.

In order for a program to give good explanations,
it needs to know:

- which facts are part of the derivation?
- which of these are likely unknown
  to the questioner?

Is there really only one derivation?
What problems do multiple derivations pose?

Could you tell the wrong story? (e.g., "John is greedy.")

Adding an Explanation Facility to a Logical Reasoner

1. Identify initial facts in the database as one of:

    - commonsense
    - assumptions

2. Modify the inference engine to record along with each new sentence that it infers the sentence's "set of support" — the sentences used to make the inference.

3. When asked for an explanation of sentence S, the inference engine answers with the assumptions in the set of support for S.

How Good is such a Facility?

This approach gives a minimal answer, from which the questioner can fill in the blanks.

Under what circumstances will this approach not be good enough?

- The answerer knows that the questioner is unskilled at “filling in the blanks”.

This approach assumes that each sentence has only one set of support. Can a sentence have two sets of support?

What problems do multiple derivations pose?

Uses of an Explanation Facility

Explanations can actually be used to solve problems.

- "Construct a travel itinerary that puts me in Phoenix on February 25."

  becomes

  Explain p = "There exists an itinerary that puts me in Phoenix on February 25."

- "Tell me what is wrong with this broken machine."

  becomes

  a. assume that all rules are correct
  b. explain a fault by finding items in an explanation for correct behavior that could be wrong

  This is not typically how humans diagnose faults, but...