L3: Knowledge Base (example)

Chapter 4

Outline

- Hidden knowledge (brother(X, Y))
- Represent negative knowledge
  - Example. Closed world assumption (new logical connectives)
- More problem modeling
  - Define orphan using father/2, mother/2, dead/1. Note closed world assumption.
  - Circuit problem
  - Hierarchical information

Hidden knowledge

- Define knowledge about brother.
  - For any person X and Y, X is a brother of Y if
    - X is a male,
    - X and Y have a same parent, and
    - X != Y.
  - Rule

Hidden knowledge

- Define knowledge about brother.
  - For any person X and Y, X is a brother of Y if
    - X is a male,
    - X and Y have a same parent, and
    - X != Y.
  - Rule

```
brother(X, Y) :- male(X),
      parent(Z, X), parent(Z, Y),
      X != Y.
```

Check which piece of condition you've missed when you defining the brother relation.

Represent negative knowledge

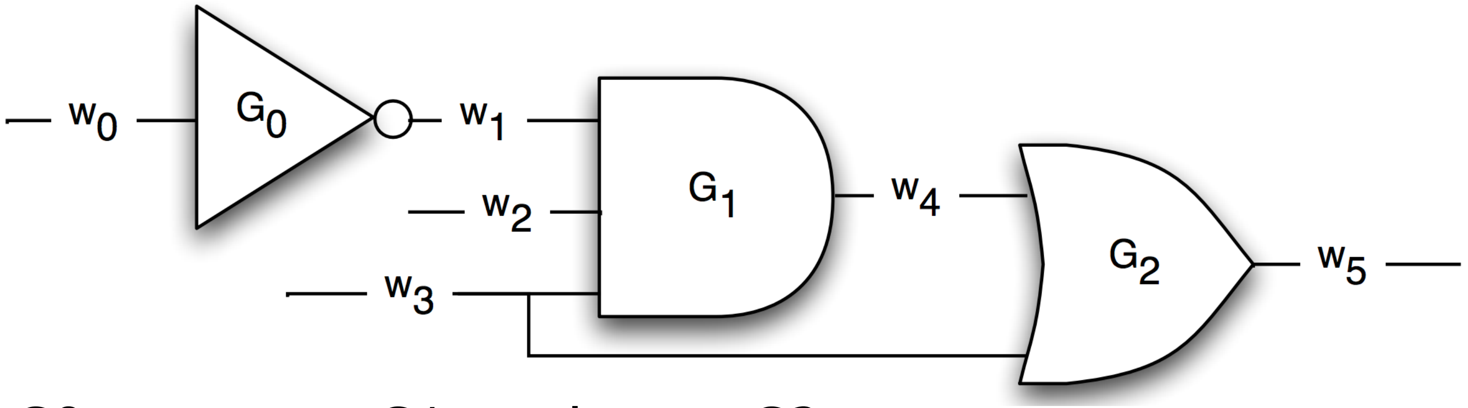
- Closed world assumption
  - Consider the family problem.
    - *Knowledge* : There is a family. John is the father and Joan is the mother. The children are Jim, Bill, and Sam.
    - *Questions.*
      - Is John the father of Bill? Who is Bill's mother?
      - Is John dad of Bill? Who is Bill's parent?
  - New question: is Jim the father of Bill?

- We assume all information is given in the problem description about then. father relation. So, we have the closed world assumption: “if we don’t know that X is the father of Y, X is not the father of Y.”
- rule

```
-father(X, Y) :- not father(X,Y).
```

- New logical connectives
  - `-`: **classical negation**
  - `not`: **default negation**

Circuit Problem



- G0: not gate, G1: and gate, G2: or gate
- Questions?
  - What are the input wires of gate G1? output wires of G1?
  - Given the value of w0 is 1, what is the value on wire w1?

- Identification
  - Objects:
    - wires: w0, …,
    - gates: g0, g1, g2, …
    - signal (value): 0, 1
    - type of gates: and, or, not
  - Relations:
    - input_wire(G, W): W is an input wire of G
    - output_wire(G, W): W is an output wire of G
    - val(W, V): the signal on wire W is V
    - type(G, T): type of gate G is T ((and, not, or).
  - Knolwedge
    - All facts about wires and gates in the diagram. e.g., w0 is an input wire of g0.
    - not gate: the output of a not gate is the opposite of its input.

- input_wire(G, W): W is an input wire of G
- output_wire(G, W): W is an output wire of G
- val(W, V): the signal on wire W is V
- type(G, T): type of gate G is T ((and, not, or).

- not gate: the output of a not gate is the opposite of its input.

```
val(Wo, V) :- type(G, not),
              output_wire(G, Wo),
              input_wire(G, Wi),
              val(Wi, Vi),
              opposite(V, Vi).
opposite(0, 1).
opposite(1, 0).
```

- value of output wire of and gate – method 1

```
%the output wire of an and gate is 0 if one of its
%input wire is 0. (note this define covers all cases when
%the output wire is 0, i.e., the knowledge is complete.)
val(W, 0) :- type(G, and),
             output_wire(G, W),
             input_wire(G, W1),
             val(W1, 0).
%the output wire of an and gate is 1 if we do not know (or
cannot derive from the program) that its is 0.
val(W, 1) :- type(G, and),
             output_wire(G, W),
             not val(W, 0).
```

- value of output wire of and gate – method 2:

% the value of the output wire of an and gate is 1
% if that of all input wires is 1.

```
val(W, 1) :- type(G, and),
             output_wire(G, W),
             allInputWis1(G).
```

% allInputWis1(G): all input wires of gate G is 1.

% someInputWis0(G): some input wire of G is 0.

```
allInpuitWis1(G) :-  not someInputWis0(G).
someInputWis0(G) :- val(W, 0),
                     input_wire(G, W).
```

- Translation
  - Knowledge to Rules: (not gate)
    - not gate: the output of a not gate is the opposite of its input.
      - refinements: (identify the objects/relations in the description above)
        - the signal on the output wire of a not gate is the opposite of the signal on the input wire of the gate.
        - For signal V, wire W, V is the signal on W if W is the output wire of a not gate, the input wire of the gate is W0 and V is the opposite of the signal V0 on wire W0. (you want also know the quantifier on "a gate". We also assume the input wire is unique here.)

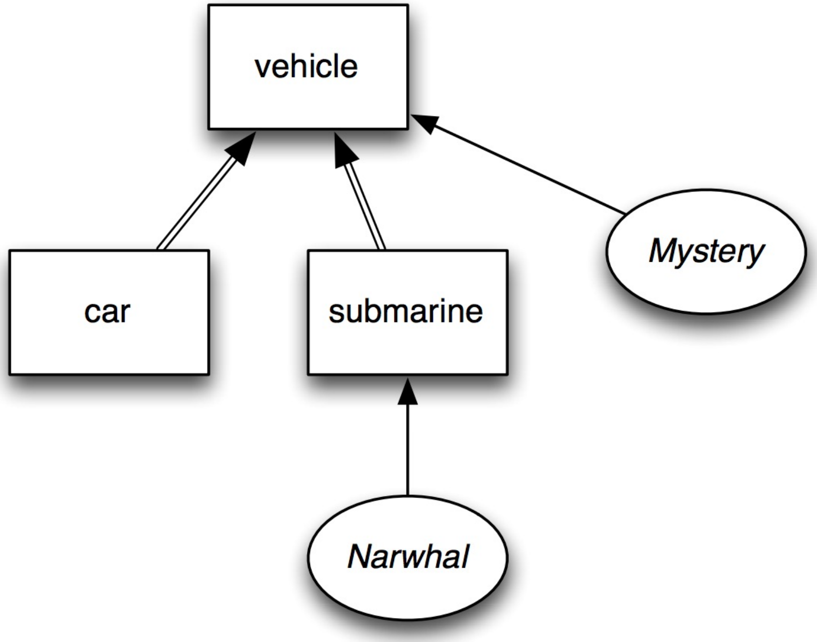
```
val(W1, V) :-
        output_wire(G, W1),
        type(G, not),
        input_wire(G, W0),
        val(W0, V0),
        opposite(V, V0).
```

  - Knowledge to rules: (and gate)
    - and gate: the output of an and gate is 0 if some of its input is 0.

Hierarchical Information and Inheritance

- Problem description
  - Knowledge:
    - The Narwhal is a submarine.
    - A submarine is a vehicle.
    - Submarines are black.
    - The Narwhal is a part of the U.S. Navy.
  - Questions:
    - What is the color of Narwhal?
    - Is Narwhale a vehicle?

- Identification
  - Objects:
    - classes: car, submarine, vehicle …
    - objects: narwhal
  - Relations:
    - directSubclass(X, Y): class X is a direct subclass of Y.
    - subclass(X, Y): class X is a subclass of Y.
    - member(X, Y): an object X is in class Y
    - is_a(X, Y): X is a direct member of class Y
    - color(X, Y): object X of color Y.
  - Knowledge …



- Translation
  - subclass: X is subclass of Y if X is a direct subclass of Y or X is subclass X1, …, Xn subclass of Y

```
subclass(X, Y) :-
    directSubclass(X, Y).
subclass(X, Y) :-
    directSubclass(X, Z),
    subclass(Z, Y).
```

  - member: X is a member of Y if X is a direct member of Y or X is a member of Z and Z is a subclass of Y.

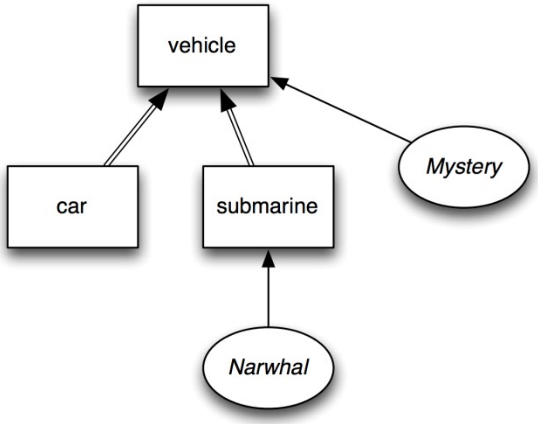
```
member(X, Y) :-
```

- Translation
  - Narwhal is a submrine

    is_a(narwhal, submarine).

  - submarine is a direct subclass of vehicle

    `directSubclass(submarine, vehicle).`



- Translation
  - Submarines are black, i.e., every submarine is black.
    - color(X, black) :- member(X, submarine).
  - An alternative knowledge: submarines are black unless they are not.
    - Refinements
      - A submarine is black unless its color is known (i.e., there is a color for it) and is not black.
      - We need two rules
        - The color of a submarine is known and is not black. We need a new relation: knownNonBlack(X).

```
knownNonBlack(X) :-
        color(X, C), C != black.
```

        - A submarine X is black if we don't believe knownNonBlack(X).

```
color(X, black) :-
        member(X, submarine),
        not knownNonBlack(X).
```

        - Compare this new approach with the one in the book.

Summary

- Represent negative knowledge.
- More examples on problem description and how to model a problem.
- Discuss more about how to write a rule
  - object, relations, and
  - continuous refinement of English description (i.e., our understanding) of a piece of knowledge until it can be easily translated into rule(s). New relations may be introduced in this refinement process. We also see a top down/problem decomposition here in representing a piece of knowledge: considering the important info first and then go the details of the info.
  - and recursive definition again.