Today

- Examinable material
- higher-order logic programming for Knowledge Representation

Examinable material

You should be able to explain the following

- Knowledge Representation Hyp, Reflection Hyp
- Declarative representation
- Deductive Reasoning as symbolic manipulation
- Syntax and semantics for first-order logic (connectives, quantifiers)
- Logical consequence
- Inference system (exact rules not expected), soundness, completeness
- Inference procedure/strategy, completeness, decision procedure

Examinable material ctd

- Proof search, using rules top down, bottom up.
- modal logics (how extend non-modal logic, non-modal treatment)
- counter-models, reading counter model from failed sequent proof
- reified time; discrete, unbounded, linear order
- meta-language, simple meta-interpreters for different search, debugging, explanation . . .
- non-monotonic logic, closed world assumption, reasoning with defaults
- reasoning maintenance, de Kleer's ATMS, update procedure

Examinable material ctd

- belief revision, epistemic entrenchment
- Amphion, deductive synthesis
- Higher-order logic programming in outline:
  quantification over predicates and functions,
  predicates as arguments to (higher-order) predicates,
  lambda-terms (locally bound variables).

Recall

:

Extend declarative language by adding:

- implication;
- higher-order variables;
- quantification

and associate search procedures with the extended language.

Higher-Order Logic for KR

Let's sketch how this language can be used to describe a standard KR approach. Suppose there are types `i`,`bool` for individuals and booleans. The KR language uses the following syntax (recall that `o` is the type of λProlog statements):

```
% First the types of the constants that
% encode the connectives.
type    ==>     bool -> bool -> bool.
type    &       bool -> bool -> bool.
type    all     (i -> bool) -> bool.
type    some    (i -> bool) -> bool.

type prim_concept (i -> bool) -> o.
type prim_role    (i -> i -> bool) -> o.
type prim_subsume (i -> bool) -> (i -> bool) -> o
type    fact       bool -> o.
```


Reasoning with concepts and subsumption

Now our reasoning engine uses the following notions:

```
% Knowledge base predicates
type intrp       bool -> o.
type subsume      (i -> bool) -> (i -> bool) -> o.
type concept      (i -> bool) -> o.
type role         (i -> i -> bool) -> o.
```

We imagine a KB with statements of type `bool`;
a concept then categorises individual entities (eg `rich` in `rich fred`);
subsumption is a relation between concepts, when one entails the other.
Finally, a role relates two individuals.

This is a form of *taxonomy*.

Concepts, Roles

Now say what the concepts and roles look like:

```
% How to make complex concepts from simple ones.

concept C :- prim_concept C.
concept (X\ (C1 X & C2 X)) :- concept C1, concept C2.
concept (X\ (all Y\ (R X Y ==> C1 Y))) :-
                              concept C1, role R.
```

So form new concepts by

- conjunction (both concepts hold;
- given conecpt holds for all related objects (under some Role) *eg* all X's children are girls

Roles

```
% How to make complex roles from simple ones.

role R :- prim_role R.
role (X\Y\ (some Z\ (R1 X Z & R2 Z Y))) :-
                              role R1, role R2.
role (X\Y\ (some A\ (C A & R1 A X & R2 A Y))) :-
                         role R1, role R2, concept C.
```

New roles from:

- composition (get "cousin" from "grandparent" and "grandchild"
- related to common entity
  (*eg* X studied at a university where Y taught)

Subsumption

Subsumption – one concept is a generalisation of another;

```
%  Subsumption of concepts
subsumes C1 C2 :- prim_subsume C1 C2.
subsumes (Z\ (A Z & B Z)) C :- subsumes A C,
                               subsumes B C.
subsumes A (Z\ (B Z & C Z)) :- subsumes A B.
subsumes A (Z\ (B Z & C Z)) :- subsumes A C.
subsumes (Z\ (all (Y\ (R Z Y ==> A Y))))
         (Z\ (all (Y\ (R Z Y ==> B Y)))) :-
                               subsumes A B.
subsumes C C.

% eg
% prim_subsumes mammal person.
```


Using these ideas

```
%  A simple little theorem prover which uses
%  the knowledge representation database.

intrp A :- fact A.
intrp (A & B)  :- intrp A, intrp B.
intrp (C X) :-   concept C, subsumes C D,
                 intrp (D X).
intrp (C U) :-
   concept C,
   subsumes (X\ (all Y\ (R X Y ==> C Y))) D,
   intrp (R V U), intrp (D V).
```

for the last clause:
suppose `D V` and `subsumes (X\ (all Y\ (R X Y ==> C Y))) D`; then
`all Y\ (R V Y ==> C Y)`; so if we have `R V U` then we can conclude `C U`.

Example

Now, we can use these tools to reason over a simple knowledge base, e.g.:

```
%  membership in concepts (is_a statements)
fact (message m1).
fact (person kirk).
fact (crew scotty).

%  relations among individuals
fact (recipient m1 scotty).
fact (body m1 send-help).
fact (sender kirk m1).
```

```
%   some primitive concepts
prim_concept message.
prim_concept person.
prim_concept crew.
prim_concept commander.
prim_concept important_message.

%   some primitive roles
prim_role recipient.
prim_role body.
prim_role sender.
prim_role senddate.
```



```
%%  Some subsumption relations

%       for messages
prim_subsume thing   message.
prim_subsume (X\ (all Y\ (sender X Y ==> person Y)))
             message.
prim_subsume (X\ (all Y\ (body X Y ==> text Y)))
             message.
prim_subsume (X\ (all Y\ (recipient X Y ==> crew Y)))
             message.
prim_subsume message important_message.
prim_subsume (X\ (all Y\(sender X Y ==> commander Y))) important_message.


%       for crew
prim_subsume person crew.
```


Querying

Now this machinery can infer, eg:

```
interp (commander kirk)
interp (person scotty)
```


Summary

- Examinable material
- Higher-order logic programming
- KR by concepts and roles, with subsumption