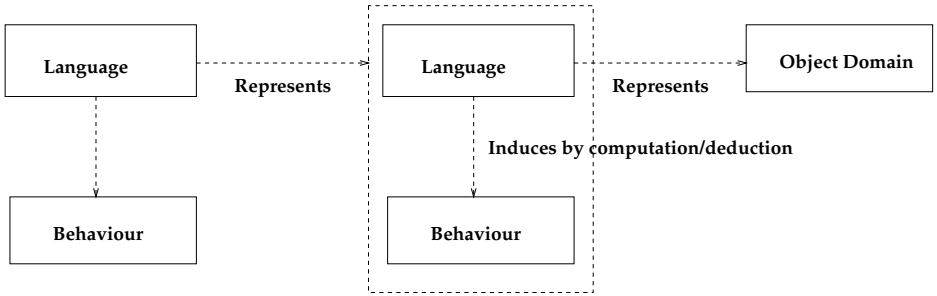
Today

- Meta-Language
- Meta-Interpreters

Recall

The Reflection Hypothesis:

give the system a representation of *itself* to manipulate.



Meta-language

Thus we get two languages, one describing the other. We say that the *meta-language* is used to talk about the *object language*.

Examples

English as meta-language, with French as object language:

> The word "poisson" is a masculine noun.

English as meta-language, with English as object-language:

> It is hard to understand "Everything I say is false".

Examples ctd

Prolog contains a mixture of object-level and meta-level statements.

`father(a,b).`	object-level
`functor(father(a,b),father,2).`	meta-level
`var(X).`	meta-level

It is better to keep these uses distinct.

Notice that `var/1` does not function according to Prolog's declarative semantics:

Compare:

```
| ?- var(X),X=2.

X = 2 ?

yes
| ?- X=2, var(X).

no
```

(remember, Prolog comma is just conjunction.)

Declarative Reading of Prolog

The clauses of the program correspond to universally quantified statements.

Prolog: `member(X,[H|T]) :- member(X,T).`
logic: $\forall x\, \forall h\, \forall t\, member(x,t) \rightarrow member(x,[h|t])$

A query corresponds to an existentially quantified statement:

Prolog: `?- member(X,[0,1]).`
logic: $\exists x\, member(x,[0,1])$

Success with `X=0` means that $member(0,[0,1])$ follows logically from the program.

Prolog in Prolog

Take the program:

```
father(a,b).

ancestor(X,Y) :- father(X,Y).
ancestor(X,Y) :- father(X,Z), ancestor(Z,Y).
```

We can write a description of Prolog programs in Prolog:

```
clause( father(a,b), true ).
clause( ancestor(X,Y), father(X,Y) ).
clause( ancestor(X,Y),
         (father(X,Z), ancestor(Z,Y)) ).
```


Meta-level Interpreter

We can also represent Prolog *inference* in Prolog.
When is a query solved?

```
solve( true ).
solve( (A,B) ) :- solve(A), solve(B).
solve( A ) :- clause(A,B), solve(B).
```

(This is known as the vanilla interpreter.)

To query, use

```
?- solve( ancestor(X, b) ).
```

As it stands, this mimics the Prolog interpreter, less efficiently. But we can use the idea to be more imaginative.

For example, we can describe other inference rules in Prolog.

Other inference procedures

We can also describe a different inference procedure:

```
% work with list of goals:  e.g., sort to
% look at clauses with fewer vars first.
% rule/2 is syntactic variant of clause/2.

solve_srt( Goal ) :-
        solve_srt_list( [ Goal ] ).
solve_srt_list( [] ) :- !.
solve_srt_list( Goals ) :-
        sort_goals( Goals, [First|Rest] ),
        rule( First, Body ),
        append( Body, Rest, NewGoals ),
        solve_srt_list( NewGoals ).

sort_goals( X, Y ) :- ....
```


A complete inference procedure

NB: Normal Prolog inference is incomplete.

```
% iterative deepening metainterpreter.
idsolve(Query) :- idsolve(Query,0).
idsolve(Query,N) :-
         solve1(Query/0,N),
         write('Solution found at depth '),
         write(N).
idsolve(Query,N) :-
         M is N+1,
         write('Searching at depth '),
         write(M),nl, idsolve(Query,M).

% solve1(+,+) unfolds clauses while checking
% that Max as second argument is not exceeded.
```



```
solve1(true/_, _ ) :- !.
solve1( (A/M, B ), Max ) :-
        solve1( A/M, Max ),
        solve1( B, Max ).
solve1( A/M, Max ) :-
        M =< Max, N is M+1,
        clause( A, B), postlist(B, N, BB),
        solve1( BB, Max ).
% postlist(+,+,?) distributes depth label to
%  elements of list
```

Control knowledge

This gives us a way to give control information in a more declarative way, in a meta-program.

- Specify the object-level knowledge (pure Prolog)
- Specify how to use the knowledge (meta-interpreter)

Given special characteristics, this can be a *more efficient* way of dealing with domain knowledge than using the standard interpreter.

Other meta-programs

anything that treats a program as data — compiler, debugger, . . .

Take care when combining object-level and meta-level statements in a single language.

It's very easy to get an inconsistent theory.

Example

Suppose that for every formula F in the language, there is a constant $\ulcorner F \urcorner$;
we have some predicate (call it $true$) such that

$$true(\ulcorner F \urcorner) \leftrightarrow F$$

for every formula F;
suppose also we have a diagonalisation property (this is a fairly weak condition.)

For any formula $G(x)$ with one free variable, there is a formula F such that

$$G(\ulcorner F \urcorner) \leftrightarrow F.$$

Then our logic is inconsistent!

We can find the inconsistency by diagonalising $\neg true(x)$.

There is a formula F such that

$$\neg true(\ulcorner F \urcorner) \leftrightarrow F.$$

By the definition of the truth predicate,

$$true(\ulcorner F \urcorner) \leftrightarrow F$$

so

$$\neg true(\ulcorner F \urcorner) \leftrightarrow true(\ulcorner F \urcorner)$$

— a contradiction.

A Choice

We can decide to use

1. Separate levels (a meta-logic and an object logic)

2. A single reasoning system (reflection)

In practice, we can have two levels, with connections between them.

Example

Object: pred calculus description (say for arithmetic)
Meta: says what object formulas are provable

So we get:

Object: $\vdash_O 0 \neq 1$
Meta: $\vdash_M Prov(\ulcorner 0 \neq 1 \urcorner)$

To go between, we need "bridging" rules:

- if $\vdash_O F$ then $\vdash_M Prov(\ulcorner F \urcorner)$
- if $\vdash_M Prov(\ulcorner F \urcorner)$ then $\vdash_O F$

What's the point?

In *Meta*, we can state not only $Prov(\ulcorner F \urcorner)$ but, eg,

$$\vdash_M \forall x\, \forall y\, Prov(Imp(x,y)) \rightarrow Prov(Neg(y)) \rightarrow Prov(Neg(x))$$

This extends the reasoning powers of the system – it's a derived inference rule:

$$\frac{P \rightarrow Q \qquad \neg Q}{\neg P}$$

Example

Take two arithmetic expressions that just use $+$ and vars, eg

$$(x + (y + z)) + \ldots = ((a + b) + c) + \ldots$$

The statement is true just if the lists

$$[x, y, z, \ldots], \quad [a, b, c, \ldots]$$

are permutations.

This is a *meta-level* statement.

We can implement the meta-level algorithm; to show it is correct, we need to use the bridge rules between the object theory and the meta-theory.

Summary

- Object and meta-language
- Meta-interpreters
- Two levels or combined
  - to specialise search
  - to extend reasoning ability