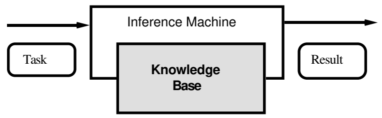
Expert Systems Classic

Expert System = Inference Machine + Knowledge Base

Knowledge Base is the most important part

Expert System Structure



Metainterprets Classic

Metainterpret = interpret that is written in the interpreted language.

Why are the metainterprets used? or
Why are the metainterprets so popular?

metainterpret is a gate to the interpret mechanism

simple changes of that mechanism

user can simple control everything he wants

Metainterprets in PROLOG

PROLOG is a very good programming language for writing metainterprets
(program and data have the same structure)

A famous PROLOG metainterpret is Vanilla metainterpret

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

New Approach to Metainterprets

Let's concentrate on the possibility of metainterpret to control an interpret mechanism.

Metainterpret = Kernel + Complement

Kernel = a general description (common part) of "all" interprets

Complement = a part that describes a concrete interpret

writing new (meta)interpret = describing interpret

= writing a complement

simplier because of the procedural part is hidden in the Metainterpret Kernel

programming a new interpret is more descriptive (like programming in PROLOG)

procedural $\rightarrow$ declarative

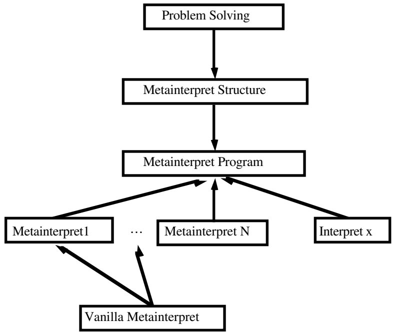
Two Ways to General Metainterpret

Top-Down Method

From Problem Solving to Metainterpret Structure

Bottom-Up Method

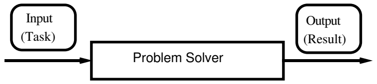
From Metainterprets to Metainterpret Kernel



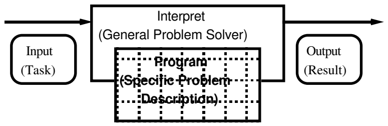
Top-Down Method (Problem Solving and Interprets)

From Problem Solver Structure to Interpret Structure

Problem Solver Structure

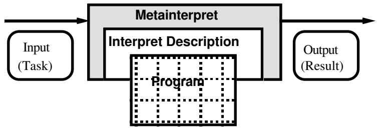


Interpret Structure



Use the similar transfer to Interpret Structure

Metainterpret Structure



General Structure of Metainterpret

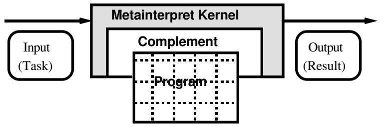
Renaming titles from previous picture (to be consistent with metainterpret definition)

Metainterpret $\rightarrow$ Metainterpret **Kernel**

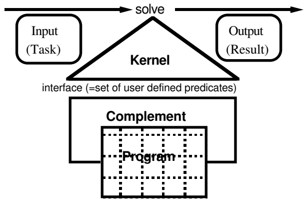
Interpret Description $\rightarrow$ Metainterpret **Complement**

Metainterpret = Kernel + Complement

General Structure of Metainterpret



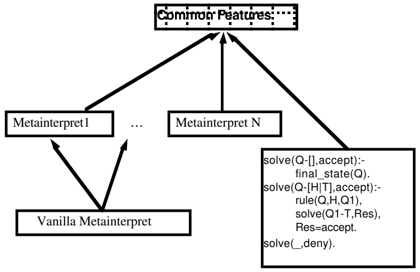
Hiearchy of General Metainterpret **(PROLOG program)**



Bottom-Up Method

From Structure to Program

Many well-known metainterprets (and interprets) are used to finding common features.



Features Found

- some tasks have simple solution
- some (all) tasks have unsuccessful solution (fail)
- simple subtask is selected from complex task
- this subtask is transformed to another subtask
- new task is made using transformed subtask
- this new task is solved (using the same mechanism)
- solution of new task is transformed to solution of original task

Metainterpret Kernel Program

realisation of common features using PROLOG program

```
solve(Task,Solution):-
      extract_solution(Task,Solution).
solve(Task,Solution):-
      select_goal(Task,Goal,Frontier),
      expand_goal(Goal,NewGoal,Rule),
      make_task(Frontier,NewGoal,NewTask),
      solve(NewTask,SubSolution),
      customize_solution(Frontier,Rule,SubSolution,Solution).
solve(Task,Solution):-
      rest_solution(Task,Solution).
```

user defined predicates of first level:

extract_solution

select_goal

expand_goal

make_task

customize_solution

rest_solution

Example of Complement

Metainterpret Kernel

```
solve(Task,Solution):-
      extract_solution(Task,Solution).
solve(Task,Solution):-
      select_goal(Task,Goal,Frontier),
      expand_goal(Goal,NewGoal,Rule),
      make_task(Frontier,NewGoal,NewTask),
      solve(NewTask,SubSolution),
      customize_solution(Frontier,Rule,SubSolution,Solution).
solve(Task,Solution):-
      rest_solution(Task,Solution).
```

Complement for Vanilla

```
extract_solution(true,yes).
rest_solution(_,no).
customize_solution(F,R,yes,yes).

select_goal((A,B),G,[B|T]):-  select_goal(A,G,T).
select_goal(G,G,[]):-   G\=(_,_),  G\=true.

expand_goal(A,B,A-B):- clause(A,B).

make_task([B|T],Goal,Task):-
      make_task(T,Goal,A),   and(A,B,Task).
make_task([],Goal,Goal).

and(true,B,B).
and(A,B,(A,B)):-  A\=true.
```

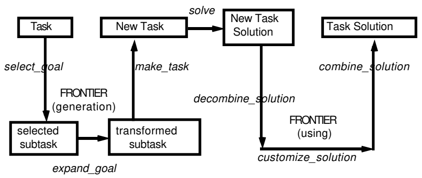
Second Level of Kernel

select_goal, make_task and *customize_solution* can be specified (they have procedural character) and new user defined predicates (of second level) will appear

Frontier Structure

Frontier = subtask selection process information + making new task process information

Second Level Structure



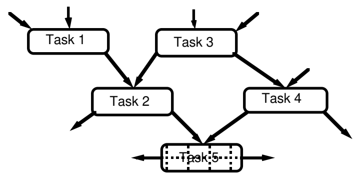
Second Level Program

```
select_goal(Task,Goal,[(S,Ch)|T]):-
      meta_task(Task),
      goal_selection(Task,SubTask,S),
      select_goal(SubTask,Goal,T).
select_goal(Task,Goal,[(S,Ch)]):-
      simple_task(Task),  /* not  meta_task(Task)  */
      custom_goal_selection(Task,Goal,S).

make_task([(S,Ch)|T],Goal,Task):-
      make_task(T,Goal,SubTask),
      combine_tasks(S,SubTask,Task,Ch).
make_task([],Goal,Goal,[]).

customize_solution([F|T],Rule,SubSol,Sol):-
      decombine_solution(F,SubSol,Sol1,Sol2),
      customize_solution(T,Rule,Sol1,SSol1),
      combine_solution(F,SSol1,Sol2,Sol).
customize_solution([],Rule,SubSol,Sol):-
      combine_rule_solution(Rule,SubSol,Sol).
```

Metainterpret Kernel vs. Search

A. Newell: "Search is a key part of intelligence"

Metainterpret Kernel has a structure of search algorithms



edge in graph = using rule (and selecting subtask)

Task5 - task with simple solution