Logic and Knowledge Representation

Problem Types, and Problem solving methods

27 April 2018

Giovanni Sileno gsileno@enst.fr

Télécom ParisTech, Paris-Dauphine University

Problem solving

Well-defined problems

Problems are *well-defined* when there is a simple test to conclude whether a solution is a solution.

J. McCarthy (1956) The inversion of functions defined by Taring ma- chines. Automata Studies, Annals of Mathematical Studies, 34:177 – 181.

Well-defined problems & problem spaces

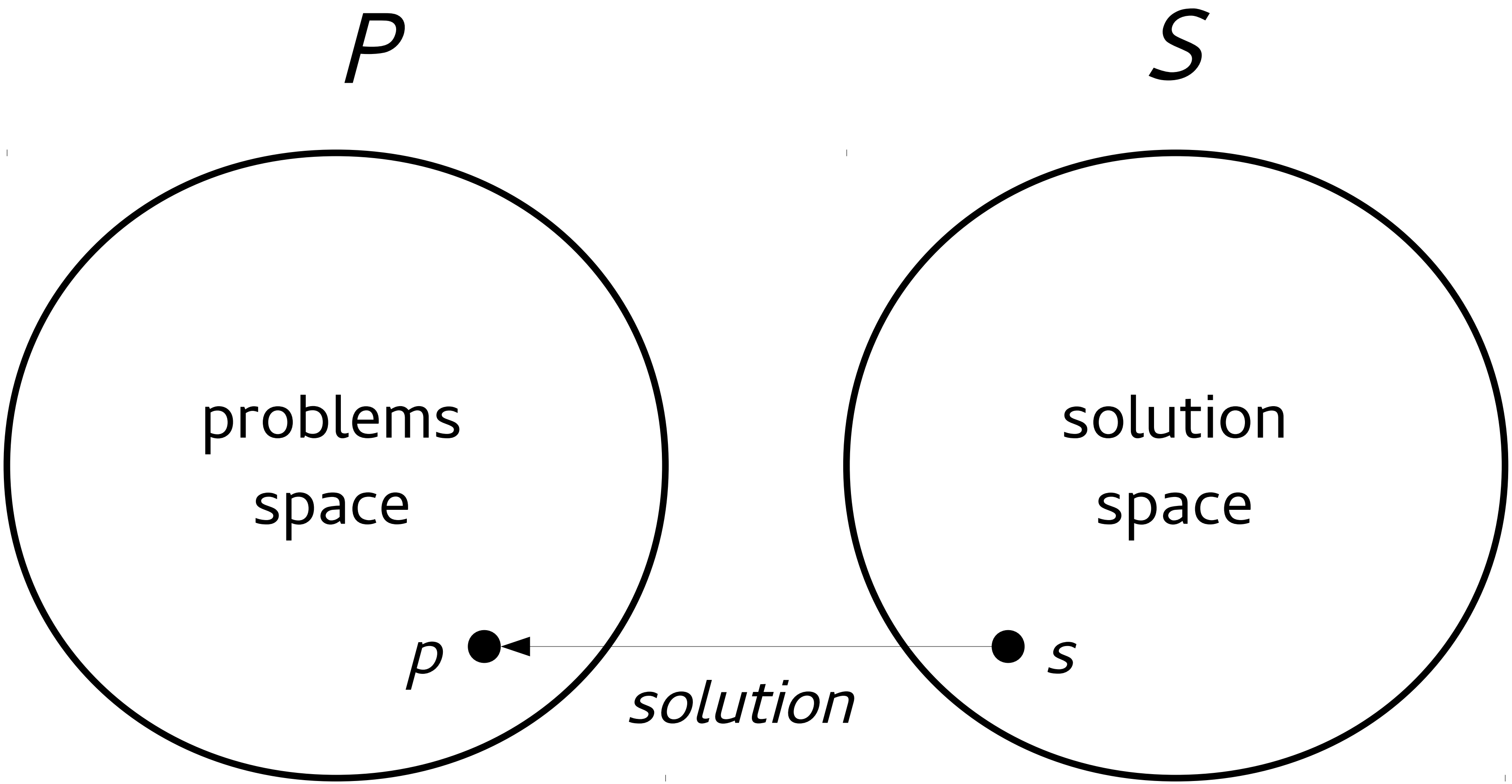
Problems are *well-defined* when there is a simple test to conclude whether a solution is a solution.

J. McCarthy (1956) The inversion of functions defined by Taring ma- chines. Automata Studies, Annals of Mathematical Studies, 34:177 – 181.

People solve problems by *searching* through a problem space, consisting of the *initial state*, the *goal state*, and *all possible states in between*.

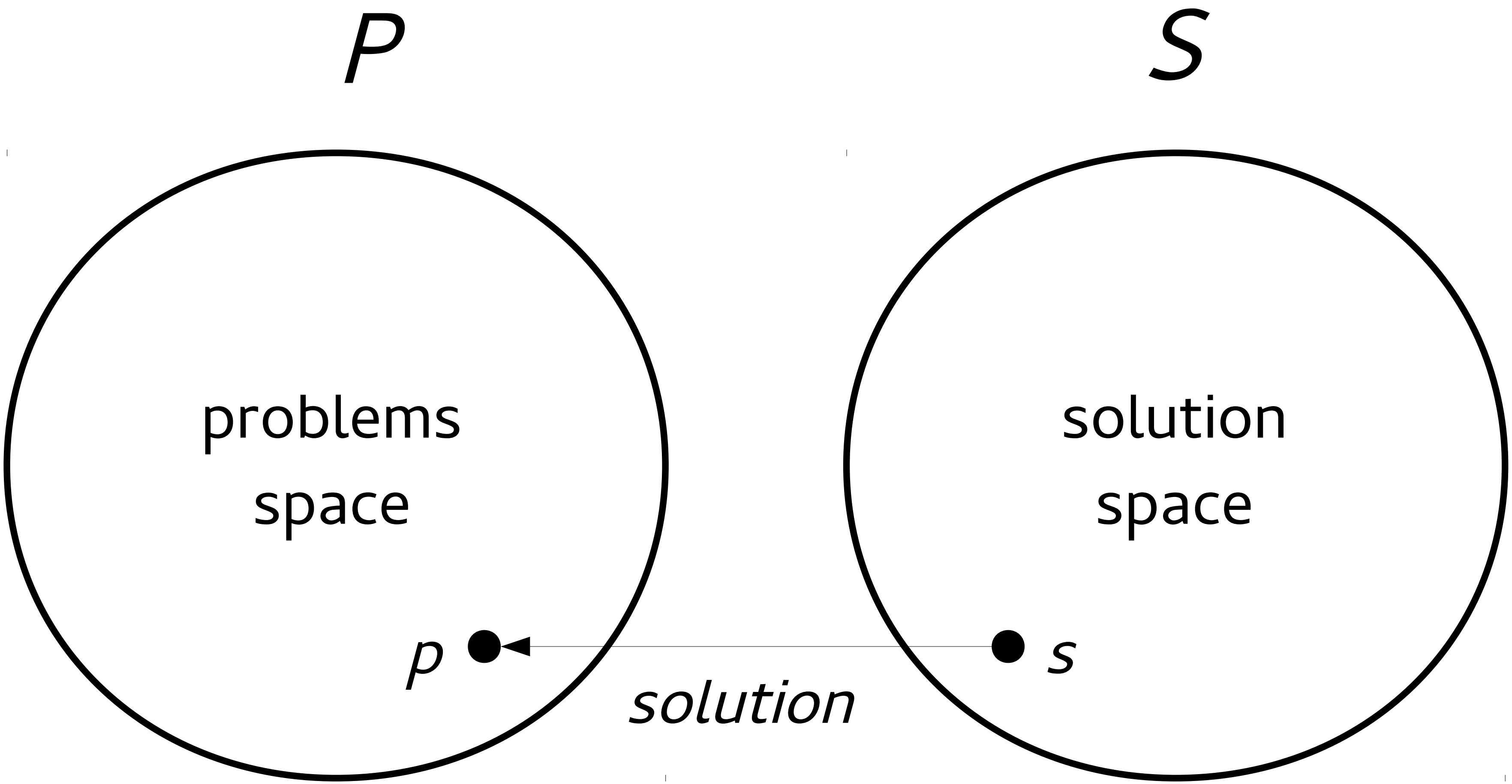
Newell, A., & Simon, H. A. (1972). Human problem solving.

Problem and solution spaces



`solution(s, p)` can be interpreted as s *satisfies* p

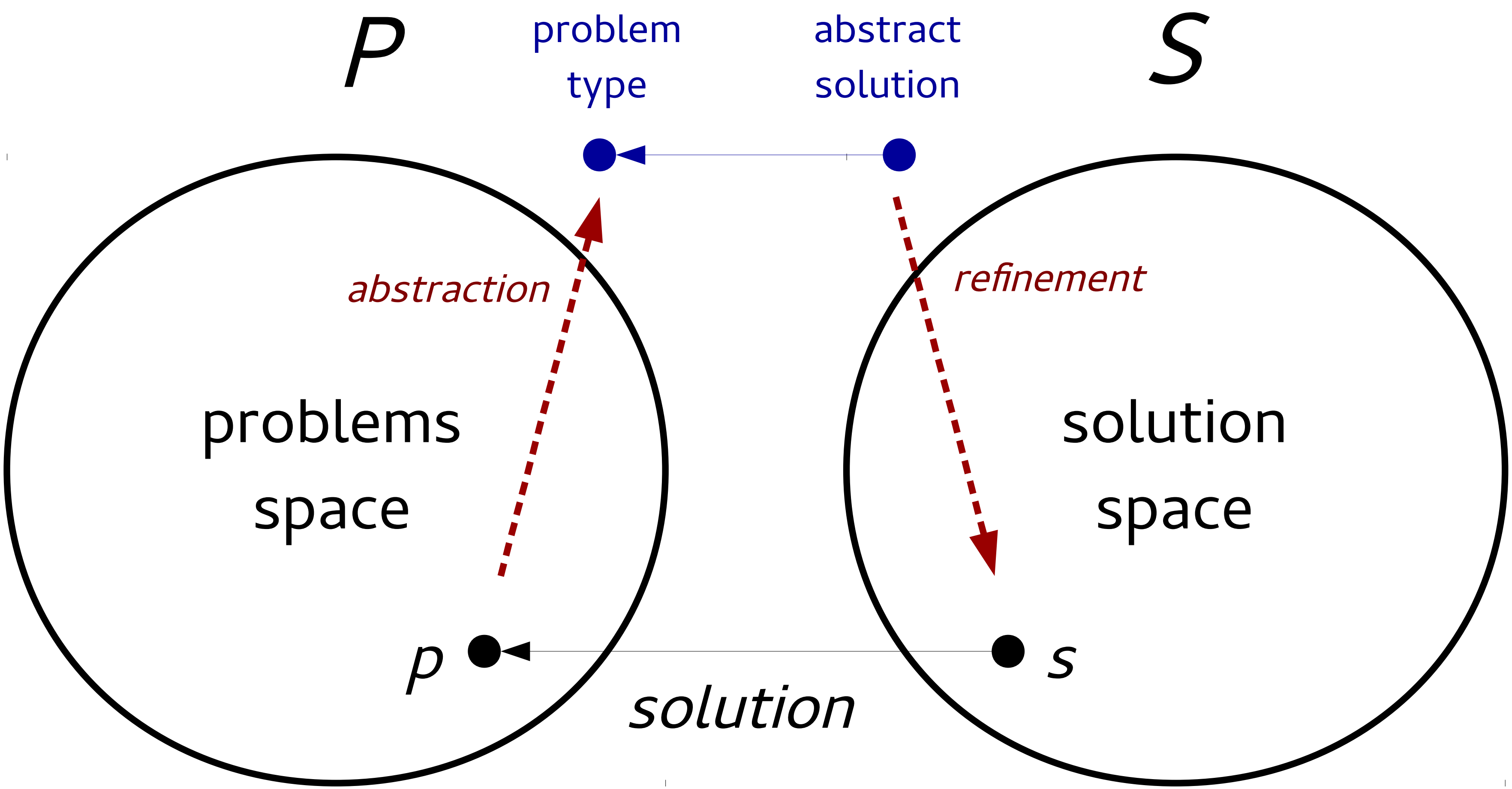
Problem and solution spaces



`solution(s, p)` can be interpreted as s *satisfies* p

problem: how to generate solutions?

Problem and solution spaces



`solution(s, p)` can be interpreted as s *satisfies* p

problem: how to generate solutions?

Defining the problem...

An old lady wants to visit her friend in a neighbouring village. She takes her car, but halfway the engine stops after some hesitations. On the side of the road she tries to restart the engine, but to no avail.

Breuker, J. (1994). Components of problem solving and types of problems. A Future for Knowledge Acquisition, 867, 118–136.

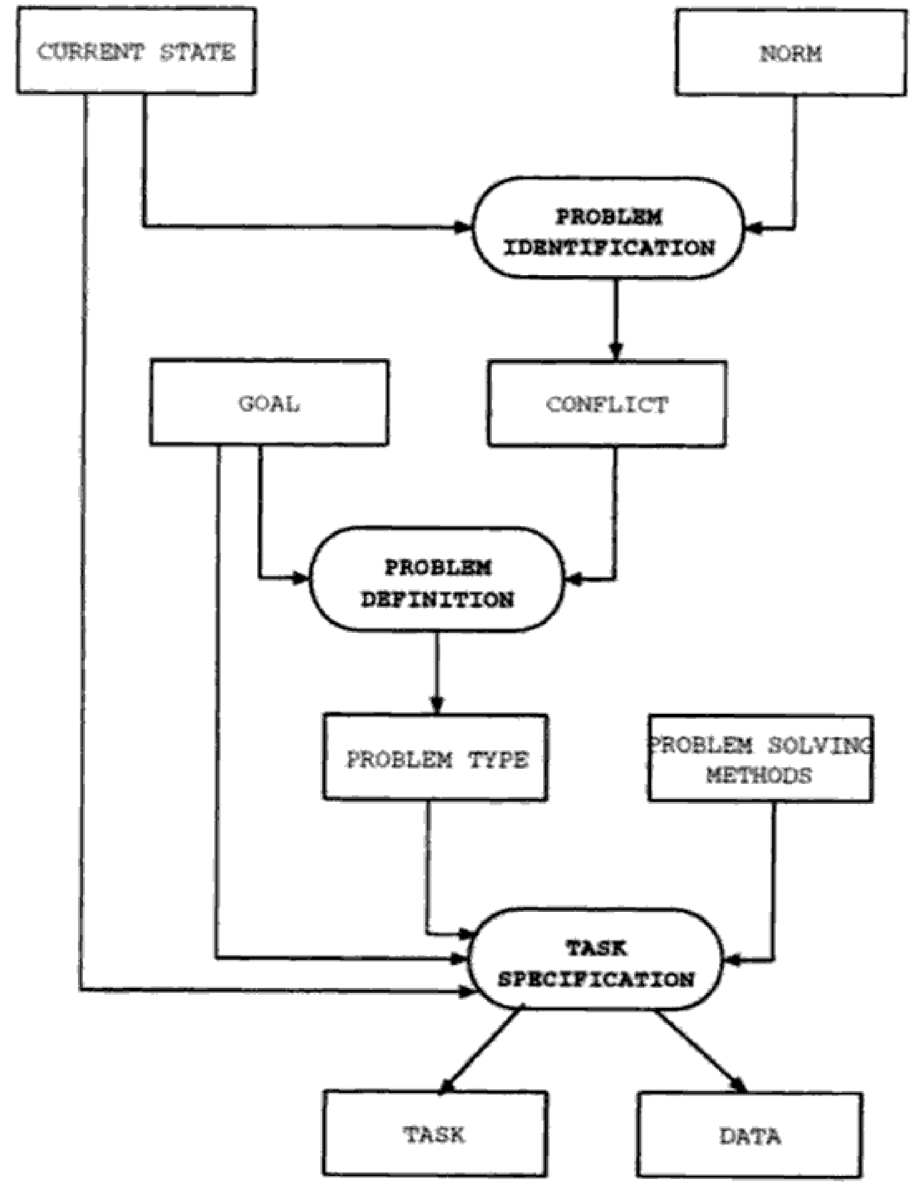
Defining the problem...

An old lady wants to visit her friend in a neighbouring village. She takes her car, but halfway the engine stops after some hesitations. On the side of the road she tries to restart the engine, but to no avail.

Which is the problem here?

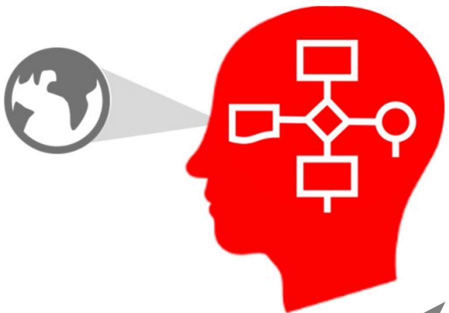
Breuker, J. (1994). Components of problem solving and types of problems.
A Future for Knowledge Acquisition, 867, 118–136.

from ill-defined to well-defined problems...



Breuker, J. (1994). Components of problem solving and types of problems. A Future for Knowledge Acquisition, 867, 118–136.

Suite of problem types



modelling

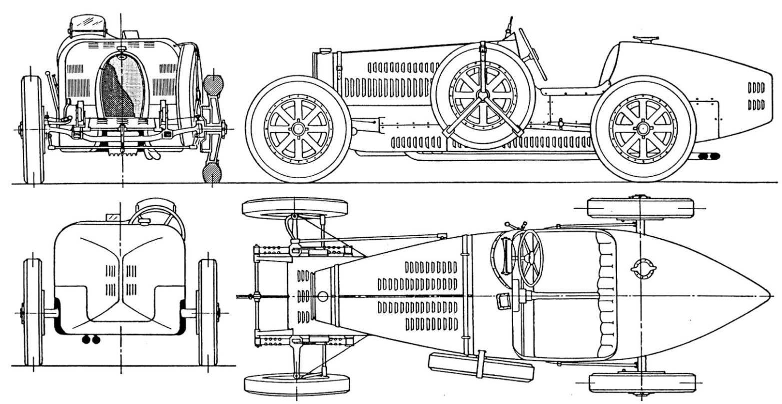
Breuker, J. (1994). Components of problem solving and types of problems. A Future for Knowledge Acquisition, 867, 118–136.

Suite of problem types

modelling

design

structural view: system



Breuker, J. (1994). Components of problem solving and types of problems.
A Future for Knowledge Acquisition, 867, 118–136.

Suite of problem types

behavioural view: system + environment

planning

modelling

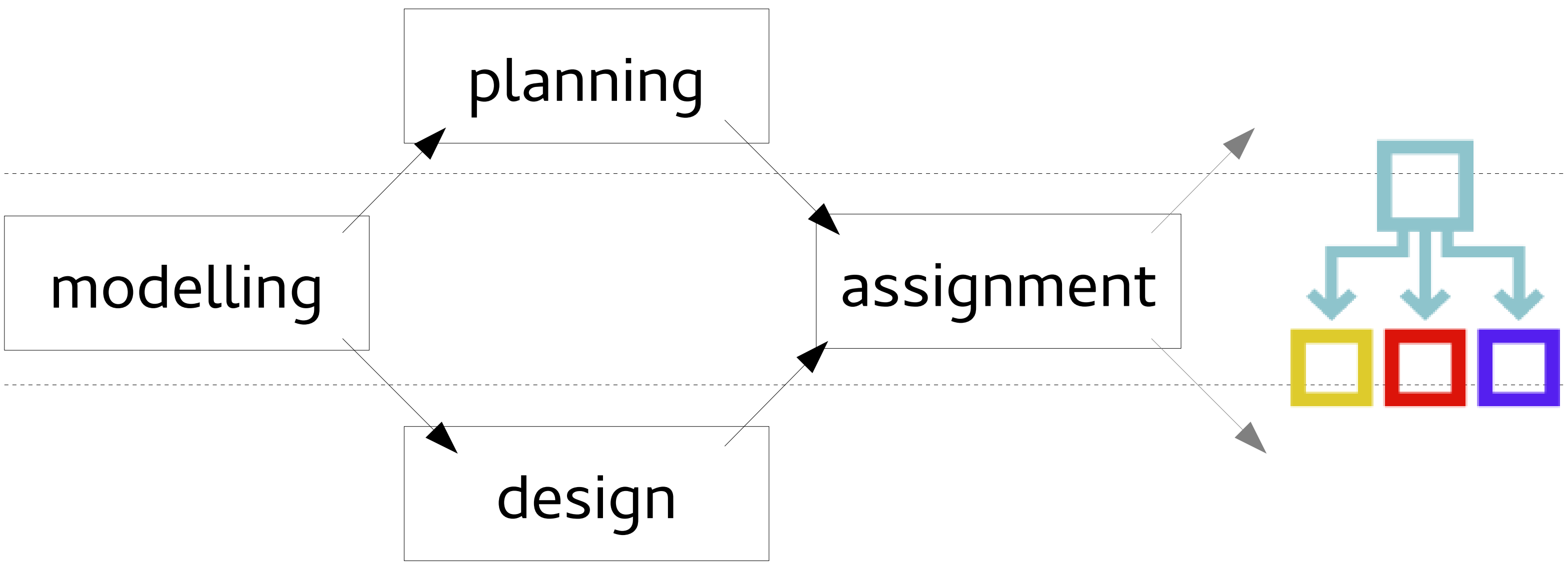
design

structural view: system

Breuker, J. (1994). Components of problem solving and types of problems. A Future for Knowledge Acquisition, 867, 118–136.

Suite of problem types

behavioural view: system + environment



structural view: system

Breuker, J. (1994). Components of problem solving and types of problems.
A Future for Knowledge Acquisition, 867, 118–136.

Suite of problem types

behavioural view: system + environment

planning

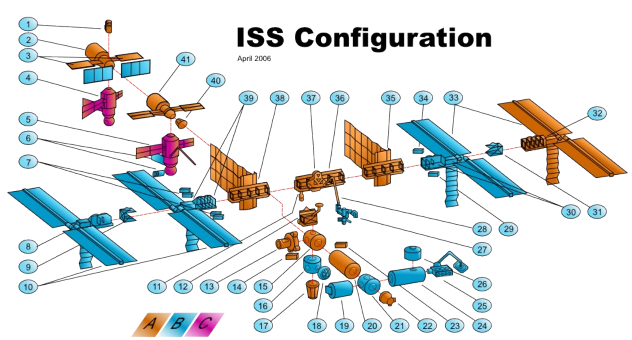
scheduling

modelling

assignment

design

configuration

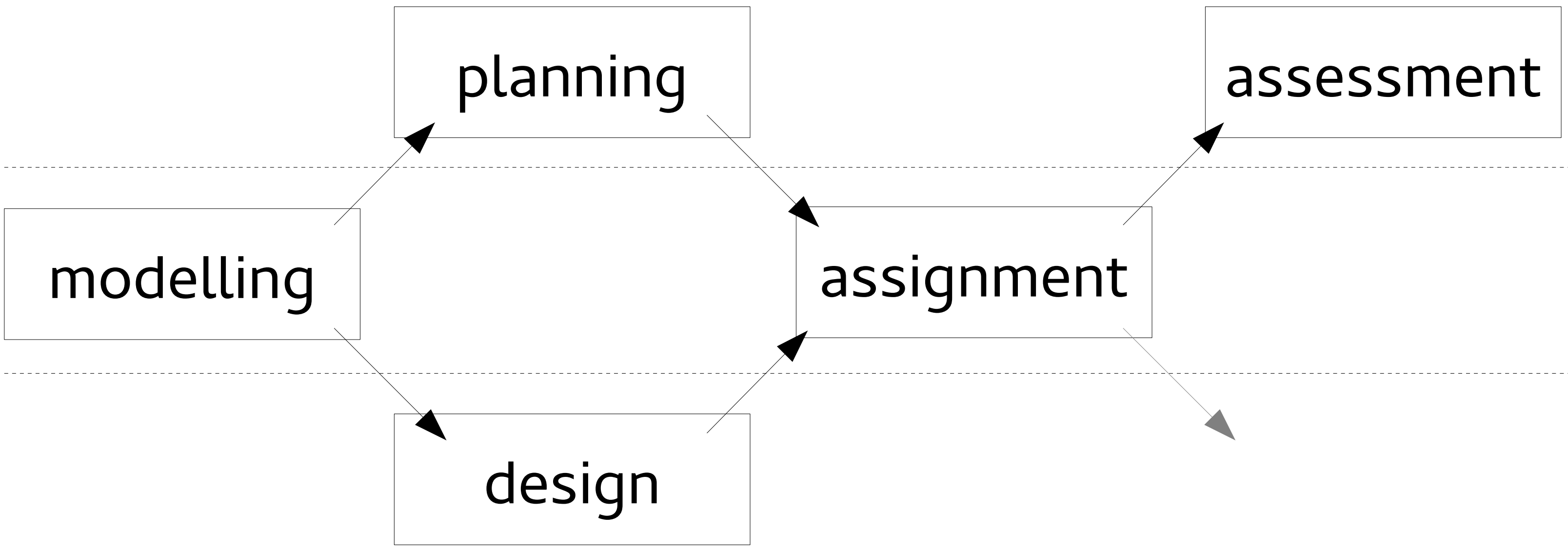


structural view: system

Breuker, J. (1994). Components of problem solving and types of problems. A Future for Knowledge Acquisition, 867, 118–136.

Suite of problem types

behavioural view: system + environment

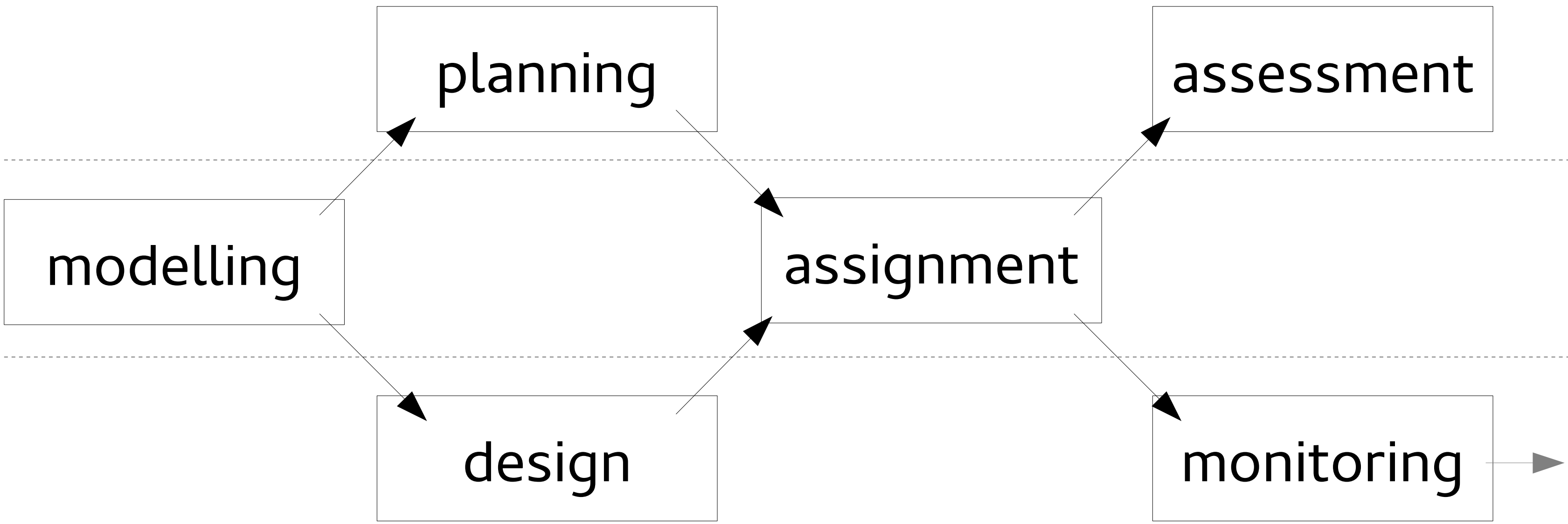


structural view: system

Breuker, J. (1994). Components of problem solving and types of problems. A Future for Knowledge Acquisition, 867, 118–136.

Suite of problem types

behavioural view: system + environment

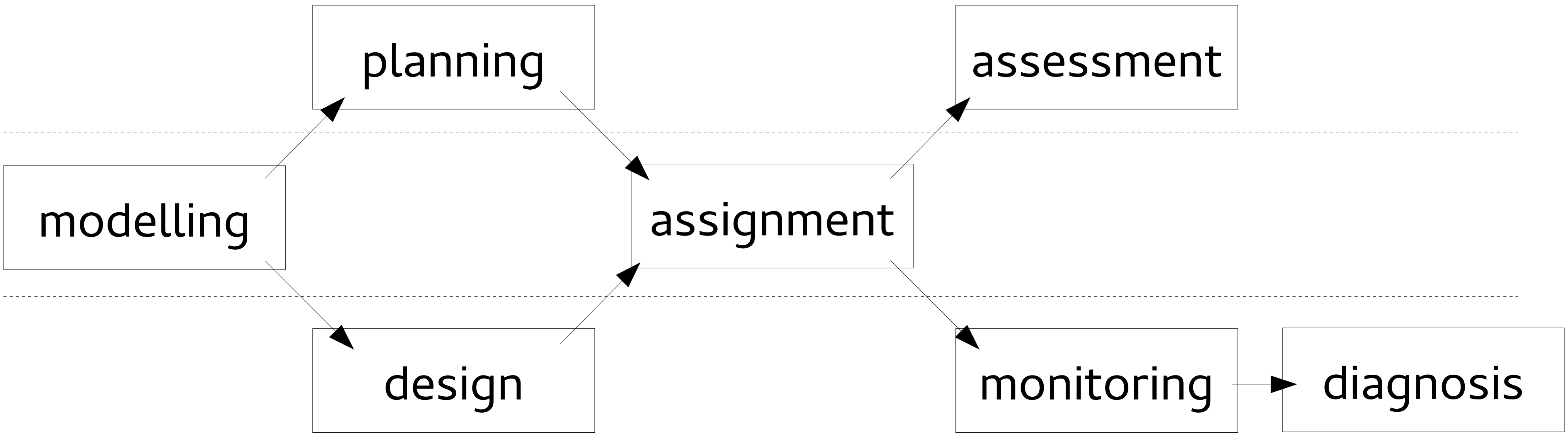


structural view: system

Breuker, J. (1994). Components of problem solving and types of problems.
A Future for Knowledge Acquisition, 867, 118–136.

Suite of problem types

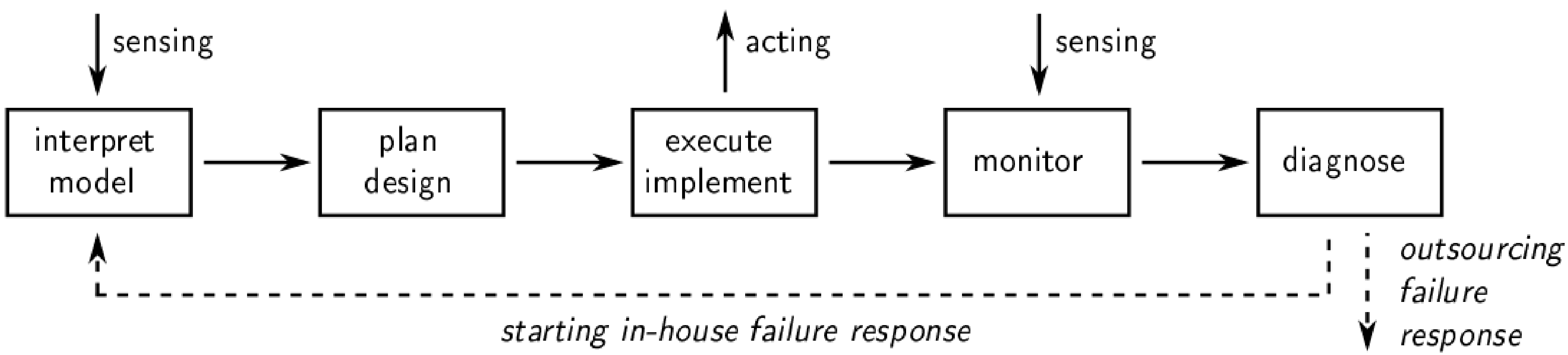
behavioural view: system + environment



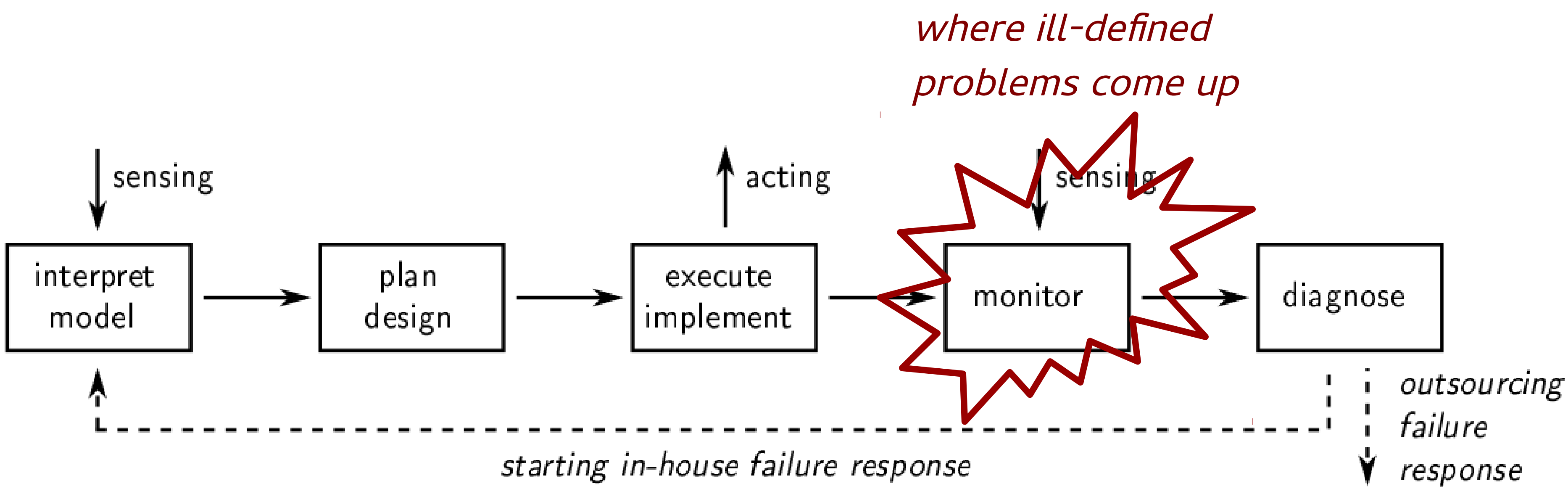
structural view: system

Breuker, J. (1994). Components of problem solving and types of problems. A Future for Knowledge Acquisition, 867, 118–136.

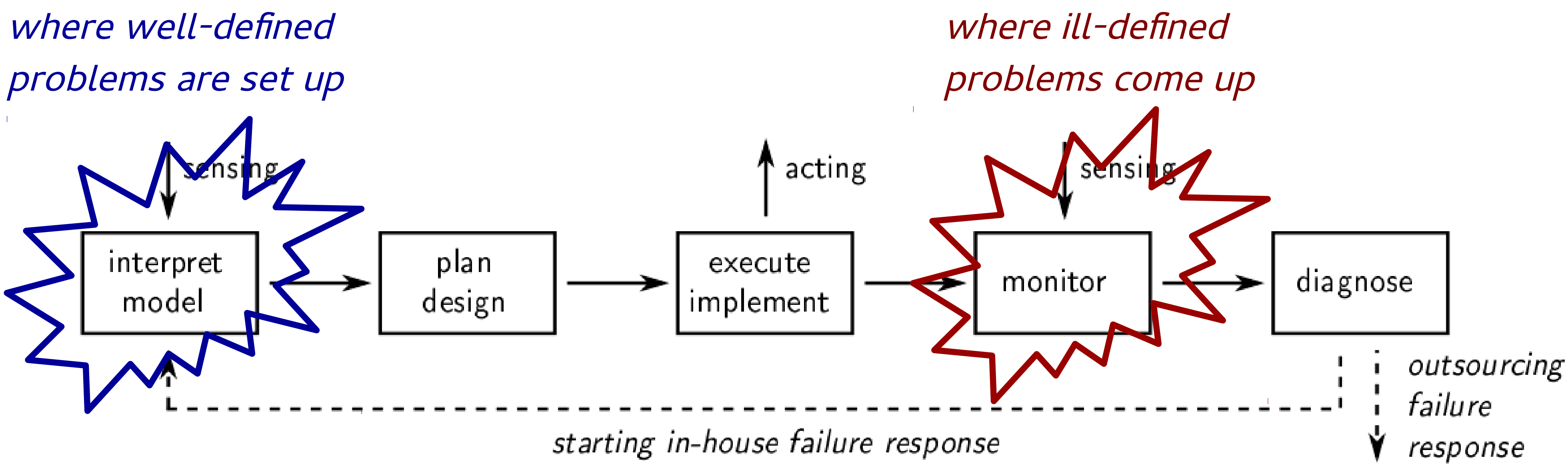
Agent problem-solving cycle



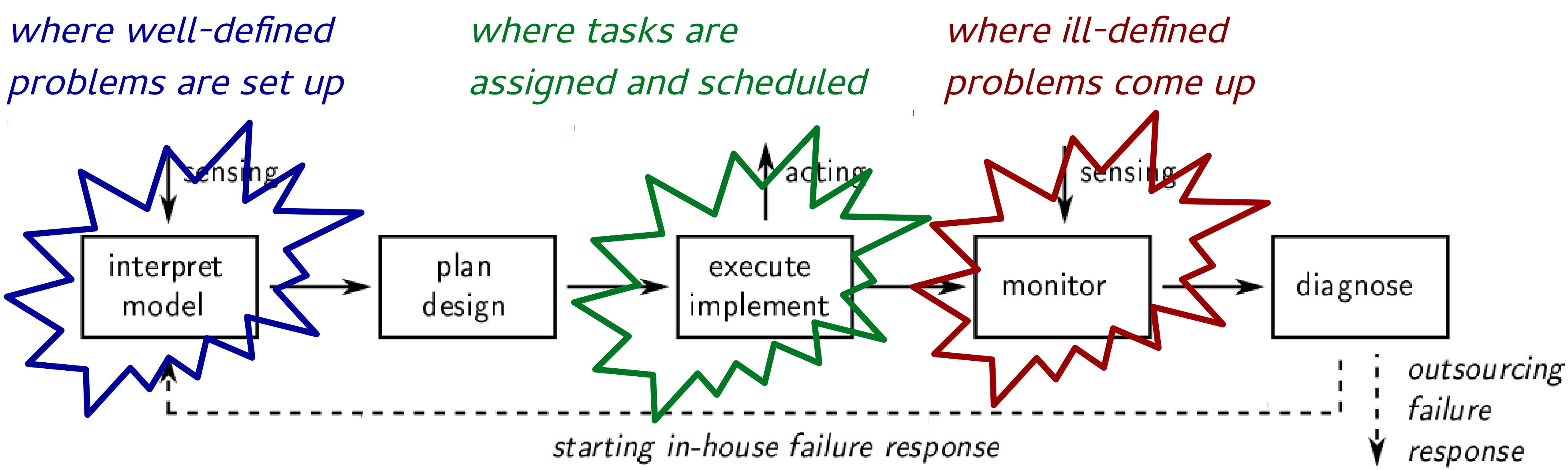
Agent problem-solving cycle



Agent problem-solving cycle

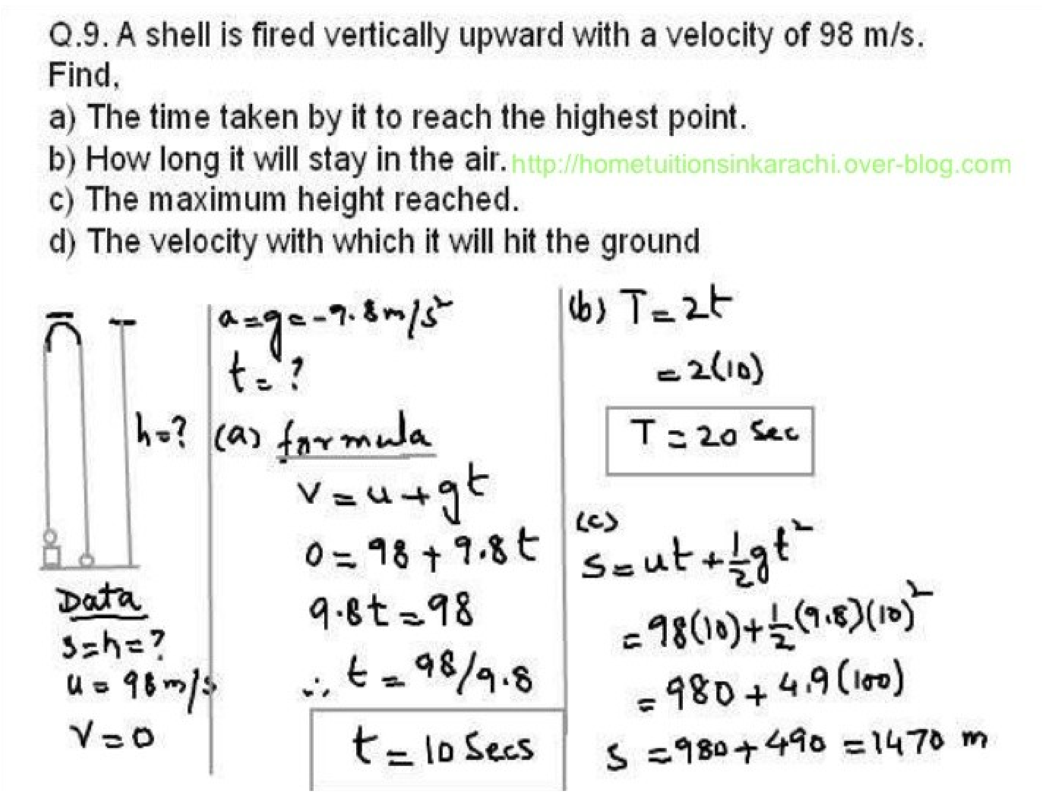


Agent problem-solving cycle



Forward and backward reasoning

How do we approach a problem?



Q.9. A shell is fired vertically upward with a velocity of 98 m/s.
Find,
a) The time taken by it to reach the highest point.
b) How long it will stay in the air. http://hometuitionsinkarachi.over-blog.com
c) The maximum height reached.
d) The velocity with which it will hit the ground

$h = ?$

Data
$s = h = ?$
$u = 98\,m/s$
$v = 0$

$a = g = -9.8\,m/s^2$
$t = ?$

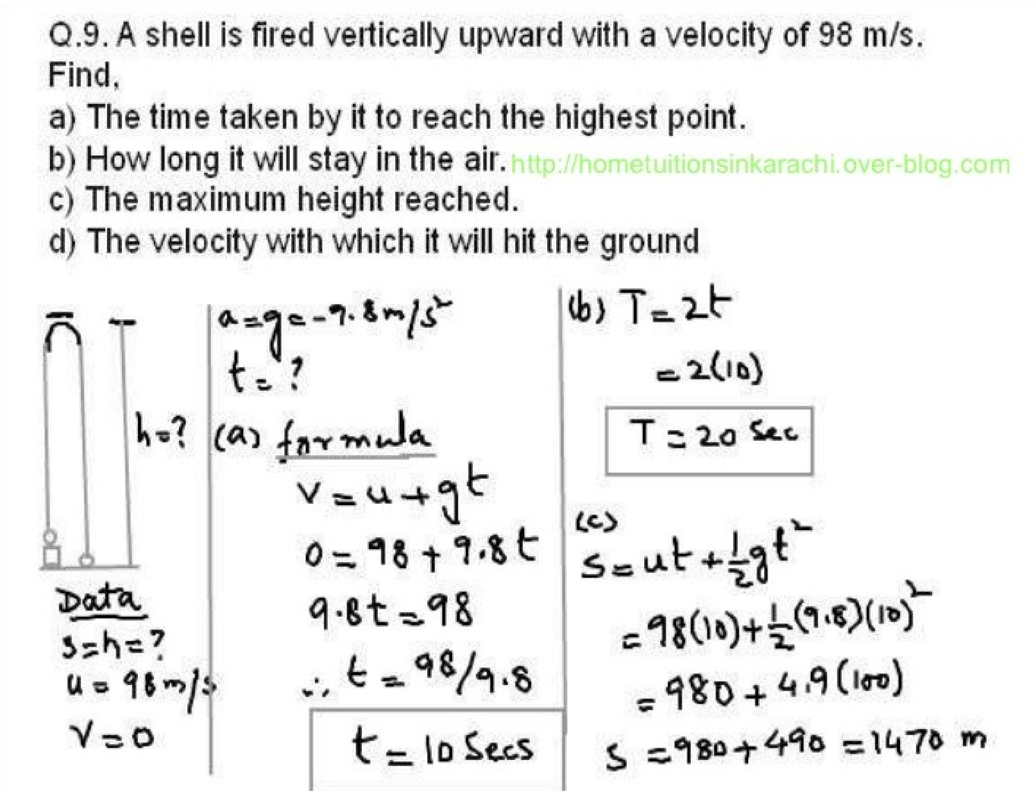
(a) formula
$v = u + gt$
$0 = 98 + 9.8t$
$9.8t = 98$
$\therefore t = 98/9.8$
$\boxed{t = 10\text{ Secs}}$

(b) $T = 2t$
$= 2(10)$
$\boxed{T = 20\text{ Sec}}$

(c)
$s = ut + \frac{1}{2}gt^2$
$= 98(10) + \frac{1}{2}(9.8)(10)^2$
$= 980 + 4.9(100)$
$s = 980 + 490 = 1470\text{ m}$

J. Larkin, J. McDermott, D. P. Simon, H. A. Simon (1980), Expert and Novice Performance in Solving Physics Problems, Science, 208(4450)

How do we approach a problem?



Q.9. A shell is fired vertically upward with a velocity of 98 m/s.
Find,
a) The time taken by it to reach the highest point.
b) How long it will stay in the air. http://hometuitionsinkarachi.over-blog.com
c) The maximum height reached.
d) The velocity with which it will hit the ground

$h = ?$

Data
$s = h = ?$
$u = 98\,m/s$
$v = 0$

$a = g = -9.8\,m/s^2$
$t = ?$
(a) formula
$v = u + gt$
$0 = 98 + 9.8t$
$9.8t = 98$
$\therefore t = 98/9.8$
$\boxed{t = 10\text{ Secs}}$

(b) $T = 2t$
$= 2(10)$
$\boxed{T = 20\text{ Sec}}$

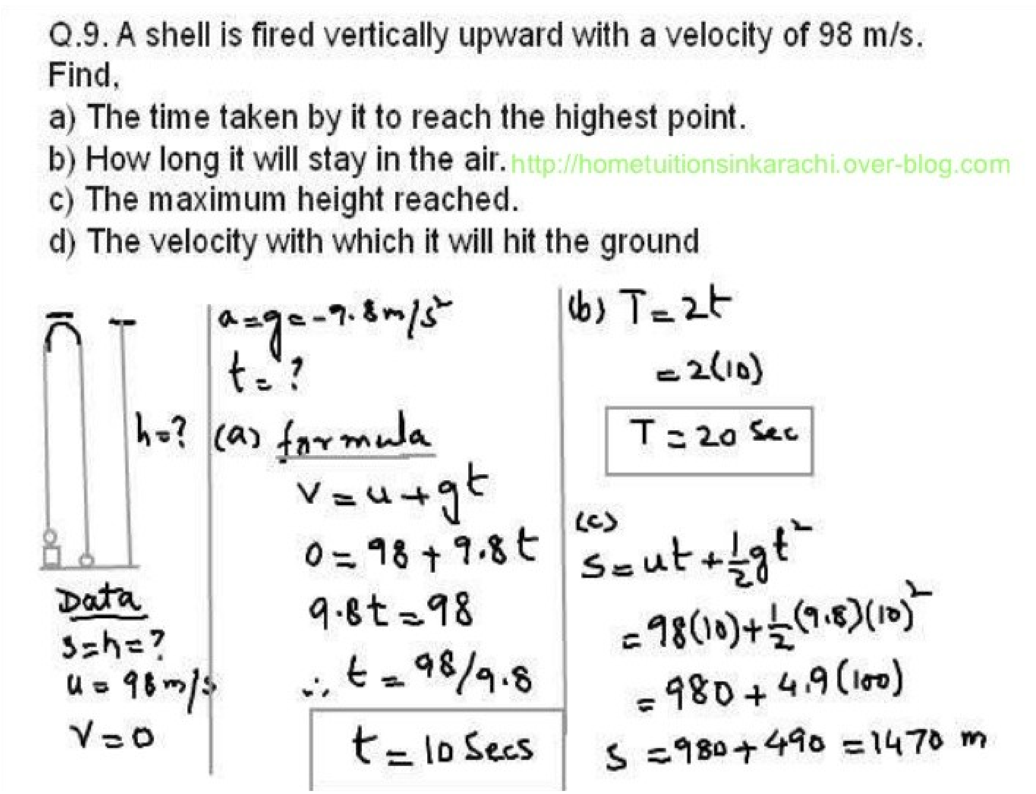
(c)
$s = ut + \frac{1}{2}gt^2$
$= 98(10) + \frac{1}{2}(9.8)(10)^2$
$= 980 + 4.9(100)$
$s = 980 + 490 = 1470\text{ m}$

Novice students start from the goal.

They look for a formula returning the goal, and then for formulas returning what it is needed by the previous one, up they formulas satisfied with the given data.

J. Larkin, J. McDermott, D. P. Simon, H. A. Simon (1980), Expert and Novice Performance in Solving Physics Problems, Science, 208(4450)

How do we approach a problem?


Q.9. A shell is fired vertically upward with a velocity of 98 m/s.
Find,
a) The time taken by it to reach the highest point.
b) How long it will stay in the air. http://hometuitionsinkarachi.over-blog.com
c) The maximum height reached.
d) The velocity with which it will hit the ground

h=?

Data
s=h=?
u = 98 m/s
V = 0

a = g = -9.8 m/s²
t = ?
(a) formula
v = u + gt
0 = 98 + 9.8t
9.8t = 98
∴ t = 98/9.8
t = 10 Secs

(b) T = 2t
= 2(10)
T = 20 Sec

(c)
s = ut + ½gt²
= 98(10) + ½(9.8)(10)²
= 980 + 4.9(100)
s = 980 + 490 = 1470 m

Novice students start from the goal.

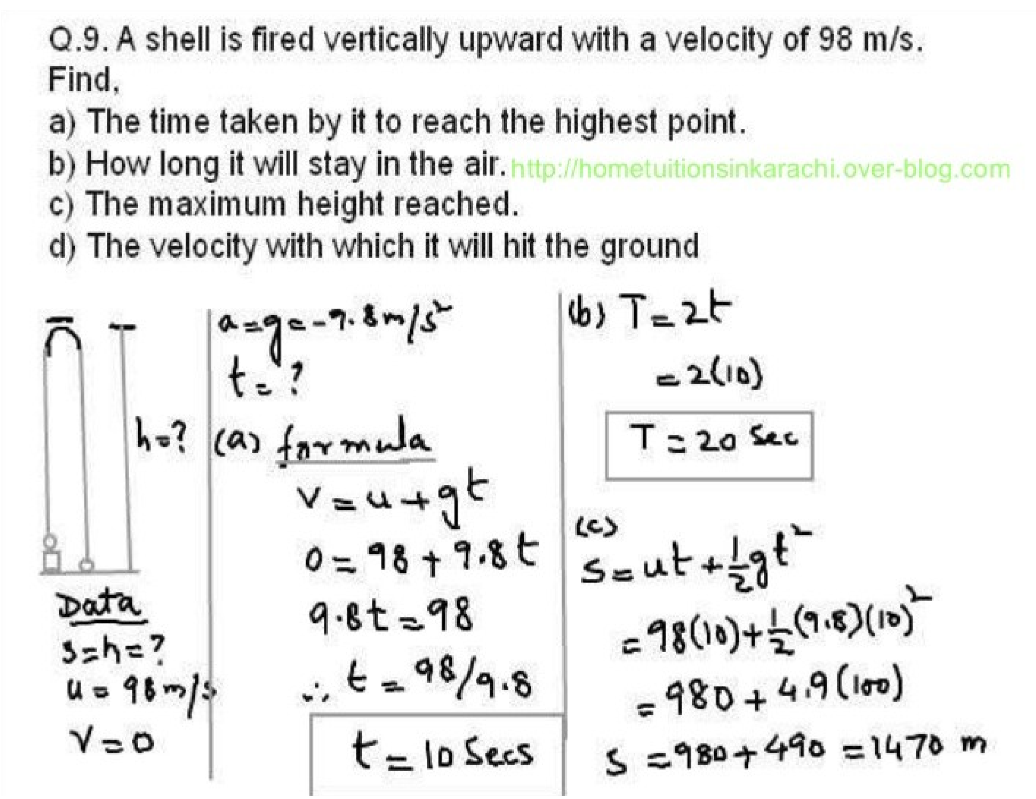
They look for a formula returning the goal, and then for formulas returning what it is needed by the previous one, up they formulas satisfied with the given data.

Experts start from the data.

They apply formulas directly to data, according to some heuristics.

J. Larkin, J. McDermott, D. P. Simon, H. A. Simon (1980), Expert and Novice Performance in Solving Physics Problems, Science, 208(4450)

How do we approach a problem?



Novice students start from the goal.

They look for a formula returning the goal, and then for formulas returning what it is needed by the previous one, up they formulas satisfied with the given data.

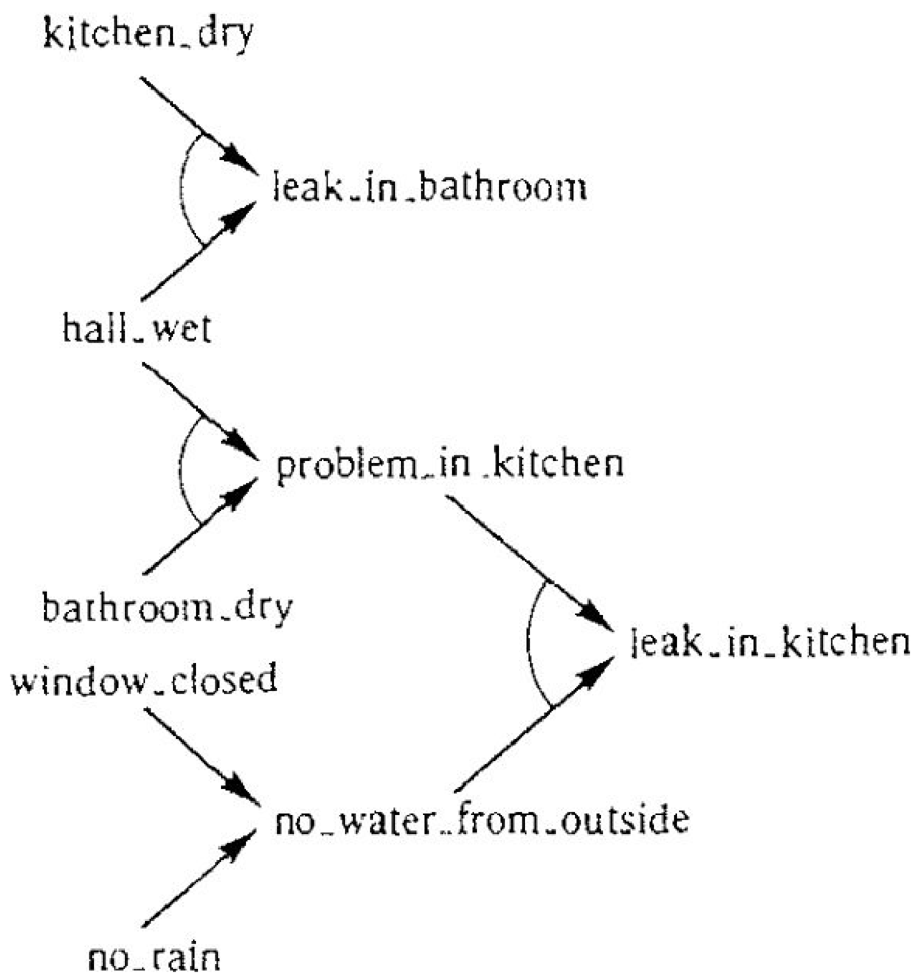
Experts start from the data.

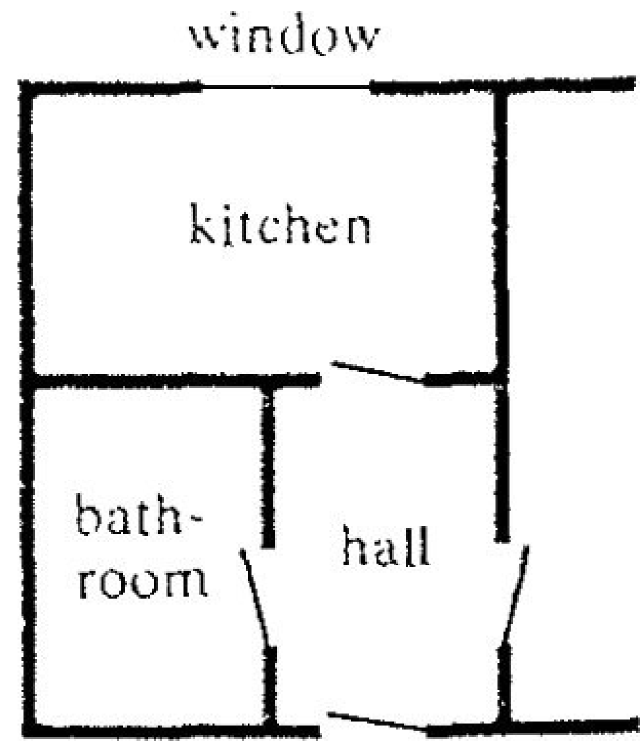
They apply formulas directly to data, according to some heuristics.

and Prolog?

J. Larkin, J. McDermott, D. P. Simon, H. A. Simon (1980), Expert and Novice Performance in Solving Physics Problems, Science, 208(4450)

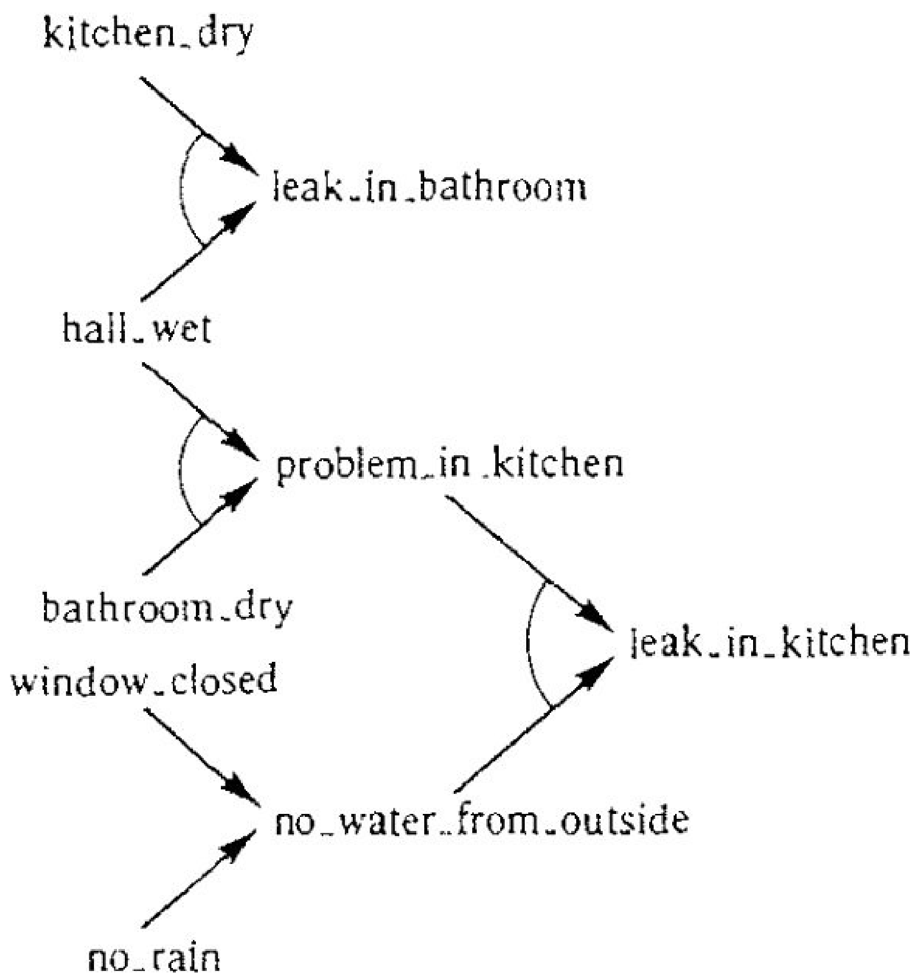
Example of diagnosis task





“expert” knowledge

Example of diagnosis task




```
leak_in_bathroom :-
    hall_wet,
    kitchen_dry.

problem_in_kitchen :-
    hall_wet,
    bathroom_dry.

no_water_from_outside :-
    window_closed ;
    no_rain.

leak_in_kitchen :-
    problem_in_kitchen,
    no_water_from_outside.

hall_wet.
bathroom_dry.
window_closed.
```

"expert" knowledge → **prolog program**

Some technical detail

```
:- dynamic(kitchen_dry/0, no_rain/0).

leak_in_bathroom :-
    hall_wet,
    kitchen_dry.

problem_in_kitchen :-
    hall_wet,
    bathroom_dry.

no_water_from_outside :-
    window_closed ;
    no_rain.

leak_in_kitchen :-
    problem_in_kitchen,
    no_water_from_outside.

hall_wet.
bathroom_dry.
window_closed.
```

necessary to define ***fluents***, i.e. facts that might change along the execution

```
?- leak_in_bathroom.
   FALSE
?- leak_in_kitchen.
   TRUE
```

prolog program

Some technical detail

```
:- dynamic(kitchen_dry/0, no_rain/0).

leak_in_bathroom :-
    hall_wet,
    kitchen_dry.

problem_in_kitchen :-
    hall_wet,
    bathroom_dry.

no_water_from_outside :-
    window_closed ;
    no_rain.

leak_in_kitchen :-
    problem_in_kitchen,
    no_water_from_outside.

hall_wet.
bathroom_dry.
window_closed.
```

necessary to define ***fluents***, i.e. facts that might change along the execution

How? ... using **`assert`** and **`rectract`**

```
?- leak_in_bathroom.
   FALSE
?- leak_in_kitchen.
   TRUE
```

prolog program

Generalization

- Using Prolog's own syntax for rules may gave certain disadvantages however:
  - this syntax may not be the most suitable for a user unfamiliar with Prolog; e.g. experts
  - the knowledge base is not syntactically distinguishable from the rest of the program
- Let us create a small DSL (*domain specific language*)!

A simple interpreter for rules

```
% symbols of DSL and priority

:- op(800, fx, if).
:- op(700, xfx, then).
:- op(300, xfy, or).
:- op(200, xfy, and).
```

priority

type of composition
(unary, binary, ...)

symbol

A simple interpreter for rules

```
% symbols of DSL and priority

:- op(800, fx, if).
:- op(700, xfx, then).
:- op(300, xfy, or).
:- op(200, xfy, and).
```

```
% knowledge base

if
   hall_wet and kitchen_dry
then
   leak_in_bathroom.

if
   hall_wet and bathroom_dry
then
   problem_in_kitchen.

if
   window_closed or no_rain
then
   no_water_from_outside.

if
   problem_in_kitchen and
   no_water_from_outside
then
   leak_in_kitchen.

fact(hall_wet).
fact(bathroom_dry).
fact(window_closed).
```

Backward chaining

```
% backward chaining rule interpreter

is_true(P) :-
   fact(P).

is_true(P) :-
   if Condition then P,
   is_true(Condition).

is_true(PI and P2) :-
   is_true(PI), is_true( P2).

is_true(PI or P2) :-
   is_true(PI) ; is_true( P2).
```

for forward chaining we need to materialize the (partial) conclusions!

```
% necessary with SWI-Prolog.
:- dynamic(sunshine/0, raining/0, fog/0).

nice :-
    sunshine, not(raining).

funny :-
    sunshine, raining.

disgusting :-
     raining,fog.

raining.
fog.
```

```
?- nice.

?- disgusting.

?- retract(fog).

?- disgusting.

?- assert(sunshine).

?- funny.
```

Forward chaining

```
% forward chaining rule interpreter
forward :-
    new_derived_fact(P),
    !,
    write('Derived:'), write(P), nl,
    assert(fact(P)),
    forward
    ;
    write('No more facts').

new_derived_fact(Concl) :-
    if Cond then Concl,
    not(fact( Concl)),
    composed_fact( Cond).
```

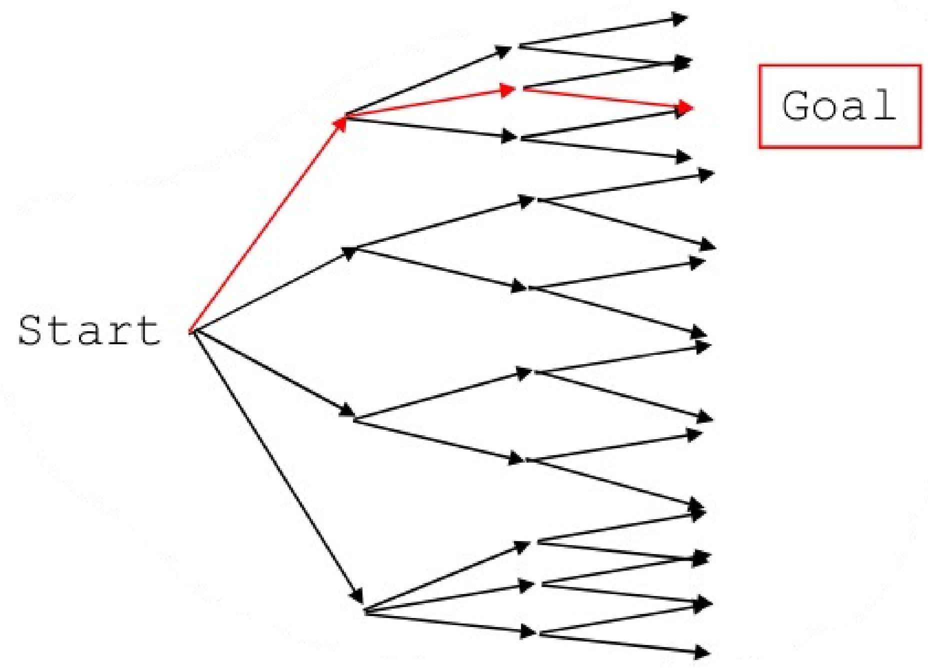
```
composed_fact(Cond) :-
    fact(Cond).

composed_fact(Cond1 and Cond2) :-
    composed_fact(Cond1),
    composed_fact( Cond2).

composed_fact(Cond1 or Cond2) :-
    composed_fact(Cond1)
    ;
    composed_fact( Cond2).
```

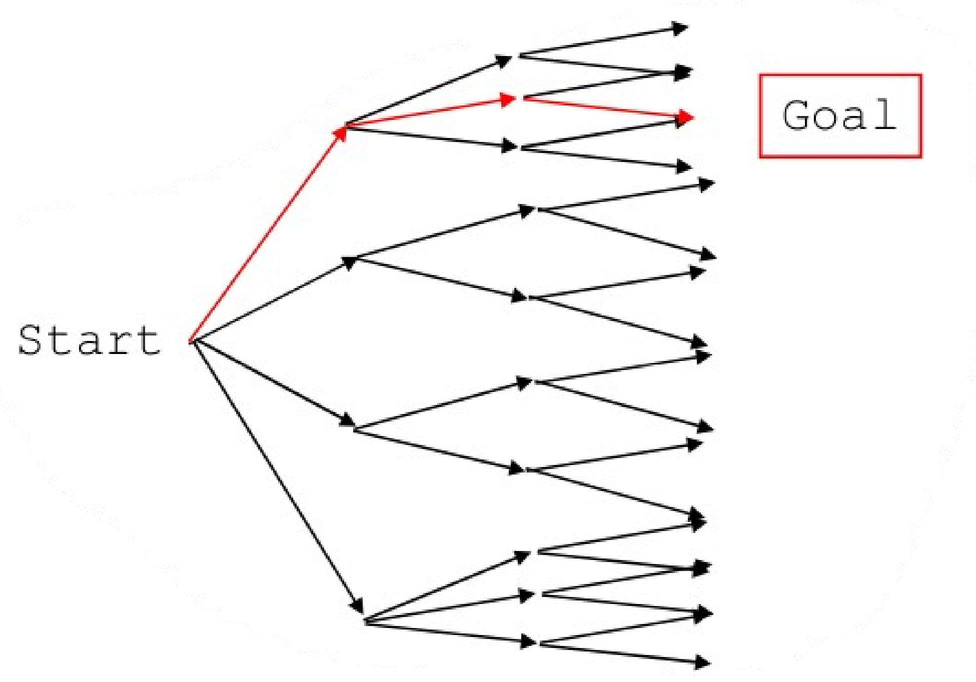
Backward vs Forward chaining

- Going from an initial state to a goal state can be unsuitable for problems with a large number of rules (all facts are derived, even the not useful ones).



Backward vs Forward chaining

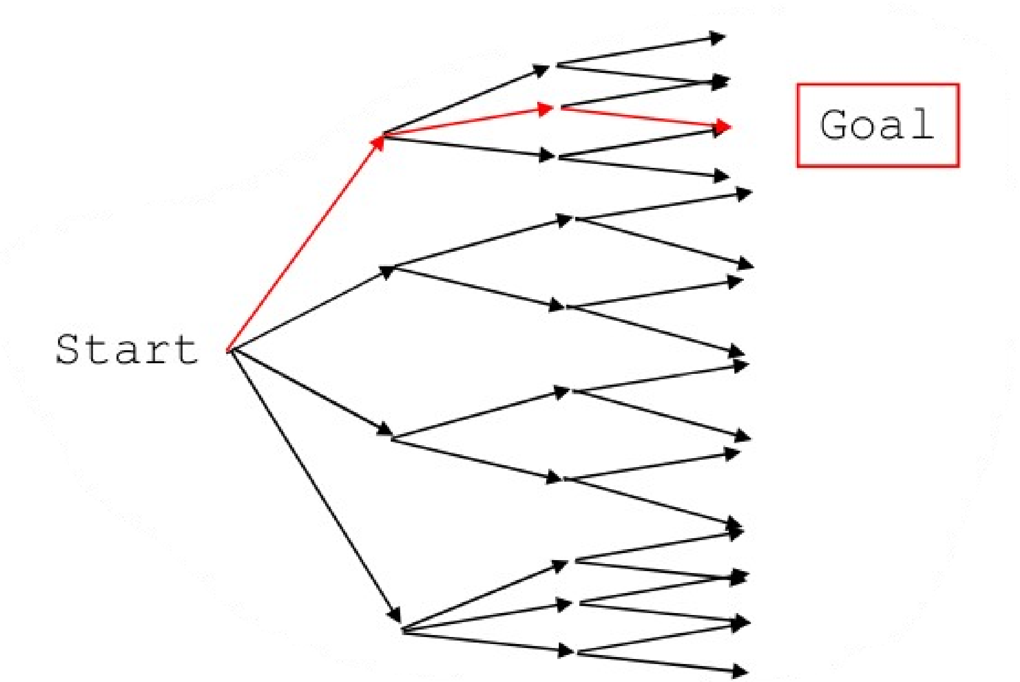
- Going from an initial state to a goal state can be unsuitable for problems with a large number of rules (all facts are derived, even the not useful ones).



- Searching backwards from the goal state usually eliminates spurious paths.

Backward vs Forward chaining

- Going from an initial state to a goal state can be unsuitable for problems with a large number of rules (all facts are derived, even the not useful ones).

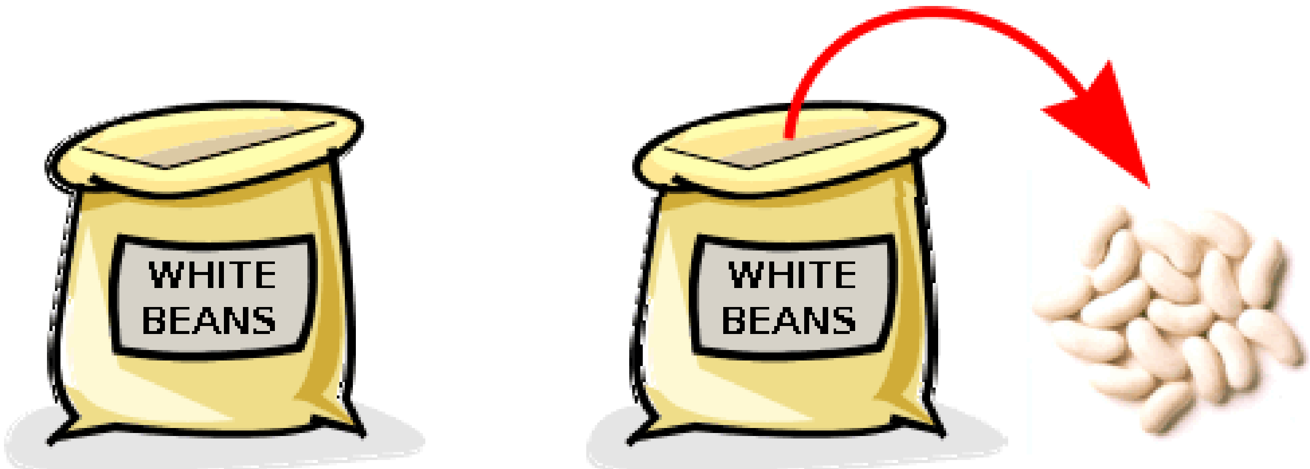


- Searching backwards from the goal state usually eliminates spurious paths.

but it does not enjoy caching abilities!

Types of reasoning (with rules)

Types of reasoning (Pierce) - 1



Deduction

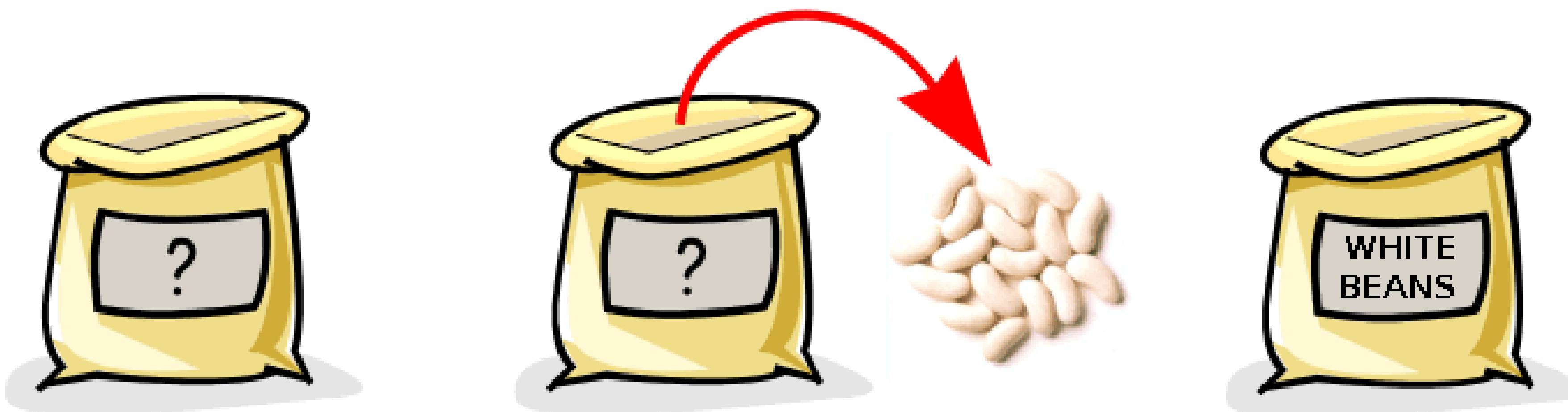
Rule: All the beans from this bag are white.

Fact: These beans are from this bag.

⇒ Result: These beans are white.

This conclusion is *certainly* true,
if the premises are true.

Types of reasoning (Pierce) - 2



Induction

Fact: These beans are from this bag.

Fact: These beans are white.

⇒ Hyp. rule: All the beans from this bag are white.

This conclusion is true until proved otherwise.

Types of reasoning (Pierce) - 3

Abduction

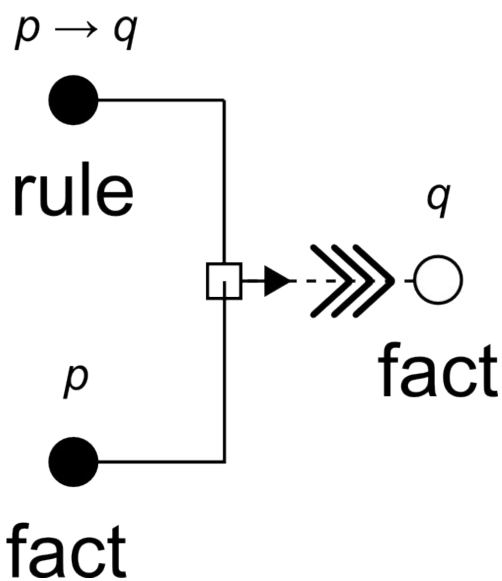
Rule: All the beans from this bag are white.

Observed fact: These beans are white.

⇒ Hyp. fact: These beans are from this bag.

This conclusion is *plausibly* true.

Resuming...

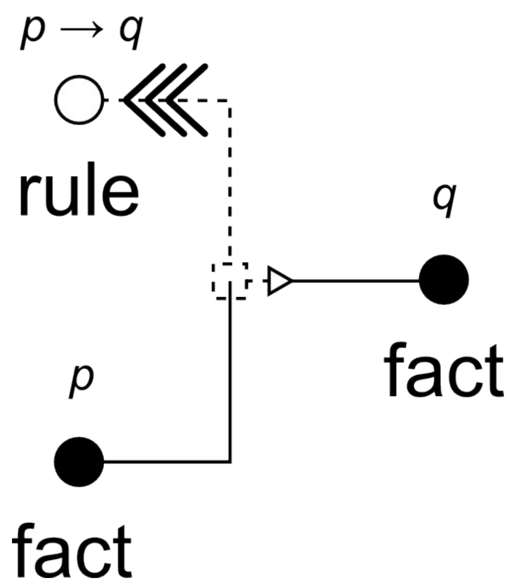


Deduction

Asserted Rule

+ Asserted Fact

= Asserted Fact

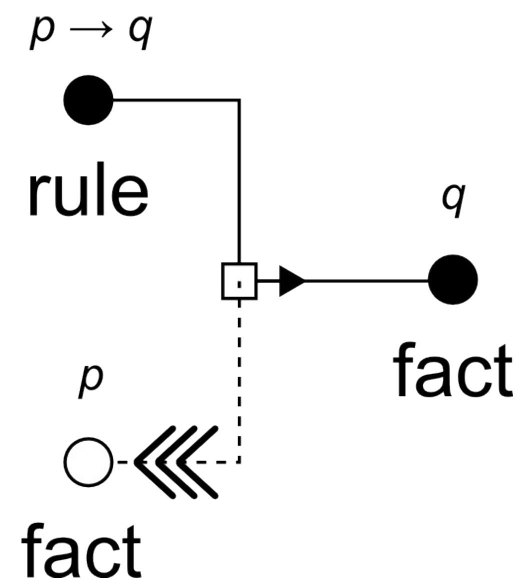


Induction

Observed Fact

+ Observed Fact

= Hypothetical Rule



Abduction

Observed Fact

+ Known Rule

= Plausible Fact

Planning

Monkeys and bananas

- A hungry monkey is in a room. Suspended from the roof, just out of his reach, is a bunch of bananas. In the corner of the room is a **box**. The monkey desperately wants the bananas but he can't reach them. What shall he do?

Monkeys and bananas

- After several unsuccessful attempts to reach the bananas, the monkey *walks to the box*, *pushes* it under the bananas, *climbs* on the box, *picks* the bananas and eats them.

Planning

- To solve this problem the monkey needed to devise a plan, a sequence of actions that would allow him to reach the desired goal.
- Planning is a topic of traditional interest in AI.
- To be able to plan, a system needs to be able to reason about the ***individual*** and ***cumulative effects*** of a series of actions.
- This is a skill that is only observed in a few animal species and only mastered by humans.

Ingredients

- Actions, with conditions and consequences:

  `action(InitialState, Action, ObtainedState)`

Ingredients

- Actions, with conditions and consequences:

```
action(InitialState, Action, ObtainedState)
```

- States of the world

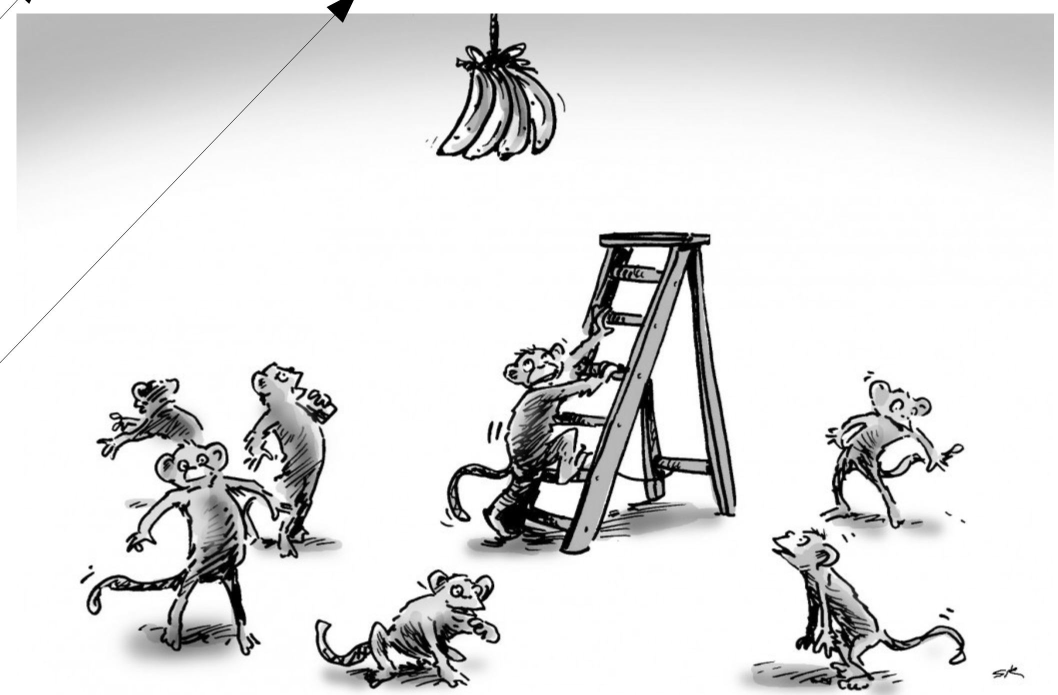
```
state(middle, onbox, middle, not_holding)
```

monkey horizonal position

monkey vertical position

box horizonal position

hand with banana relation



Ingredients

- Actions, with conditions and consequences:

```
action(state(P, floor, P, T), climb, state(P, onbox, P, T)).
```

- States of the world

```
state(middle, onbox, middle, not_holding)
```

monkey horizonal position

monkey vertical position

box horizonal position

hand with banana relation

Cooking everything

```
action(state(middle, onbox, middle, not_holding),
        grab,
        state(middle, onbox, middle, holding)).
action(state(P, floor, P, T),
        climb,
        state(P, onbox, P, T)).
action(state(P1, floor, P1, T),
        push(P1, P2),
        state(P2, floor, P2, T)).
action(state(P1, floor, B, T),
        walk(P1, P2),
        state(P2, floor, B, T)).

success(state(_, _, _, holding)).
success(State1) :-
    action(State1, A, State2),
    write("Action : "), write(A), nl,
    write(" --> "), write(State2), nl,
    success( State2).
```

goal condition

```
?- success(state(door, floor, window, not_holding)).
```

Another exercise: the tower of Hanoi

Recursion

Recursion

- Recursion is a concept widely used in computer science and linguistic.
  - an object defined in terms of itself
  - a procedure invoking itself

Recursion

- Hypothesis: natural language is recursive, as (some) syntaxic categories are recursive.

- $\underbrace{\text{John thinks } \underbrace{\text{Emily plays well.}}_{\textit{statement}}}_{\textit{statement}}$

Recursion

- Three phases: ***descent***, ***stop at bottom***, ***ascent***.

```
even([ ]).
even([_,_|L]) :- even(L).
```

Recursion

- Three phases: ***descent***, ***stop at bottom***, ***ascent***.

```
even([ ]).
even([_,_|L]) :- even(L).

?- even([3, 5, 3]).
```

Recursion

- Three phases: ***descent***, ***stop at bottom***, ***ascent***.

```
even([ ]).
even([_,_|L]) :- even(L).
```

```
?- even([3, 5, 3]).
```

Seek for an entrance

Recursion

- Three phases: ***descent***, ***stop at bottom***, ***ascent***.

```
even([ ]).
even([_,_|L]) :- even(L).
```

```
?- even([3, 5, 3]).
```

Propagate recursively

Recursion

- Three phases: ***descent***, ***stop at bottom***, ***ascent***.

```
even([ ]).
even([_,_|L]) :- even(L).
```

```
?- even([3, 5, 3]).
```

Reach the bottom of the recursion

Recursion

- Three phases: ***descent***, ***stop at bottom***, ***ascent***.

```
even([ ]).
even([_,_|L]) :- even(L).
```

```
?- even([3, 5, 3]).
```

Retrace back

Recursion

- Three phases: ***descent***, ***stop at bottom***, ***ascent***.

```
even([ ]).
even([_,_|L]) :- even(L).
```

```
?- even([3, 5, 3]).
```

Propagate the result back

Recursion

- Three phases: ***descent***, ***stop at bottom***, ***ascent***.

```
mirror(Left, Right) :-
    invert(Left, [], Right).

invert([X|L1], L2, L3) :-
    invert(L1, [X|L2], L3).
invert([], L, L).
```

Conclusions

Conclusions

- Symbolic AI presents relevant techniques to solve problems that can be described in symbolic terms, that in many cases outperform humans.

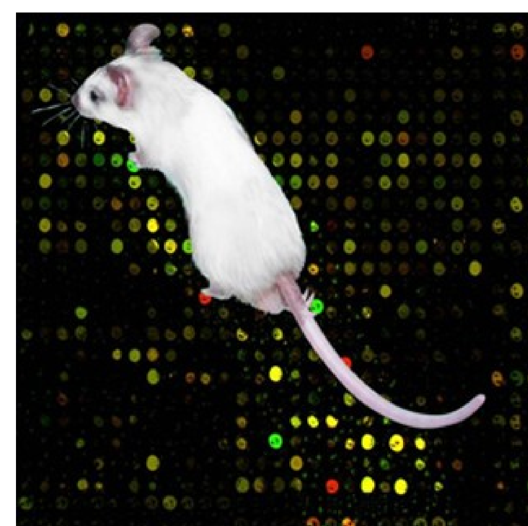
https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov

Conclusions

- Symbolic AI presents relevant techniques to solve problems that can be described in symbolic terms, that in many cases outperform humans.
- AI methods are today implemented in new generations of expert systems and all IT infrastructures of organizations.

Conclusions

- Symbolic AI presents relevant techniques to solve problems that can be described in symbolic terms, that in many cases outperform humans.
- AI methods are today implemented in new generations of expert systems and all IT infrastructures of organizations.
- However, many problems cannot be adequately handled by symbolic techniques, as e.g. those faced by *sensory-motor* modules:
  - vision, action



Conclusions

- In robotics, starting from the 80s, a radically different paradigm started to be considered, renouncing to symbolic representations.
- As Rodney Brooks famously put it: "Elephants don't play chess"
  - overlap with *machine learning*

Conclusions

- In robotics, starting from the 80s, a radically different paradigm started to be considered, renouncing to symbolic representations.
- As Rodney Brooks famously put it: "Elephants don't play chess"
  - overlap with *machine learning*
- Similarly, failures of symbolic AI explains today interest for *deep learning* techniques.

Conclusions

- However, we should not forget, that *a good deal of our interactions* with other people is not too far from playing chess.
  - expressing *how* and *why* is fundamental for humans, and for this we need symbols.
- Symbolic AI and Statistical AI occupy different sides of the spectrum of intelligent behaviour.