Effective Classical Planning and Beyond: Dealing with Uncertainty and Reward

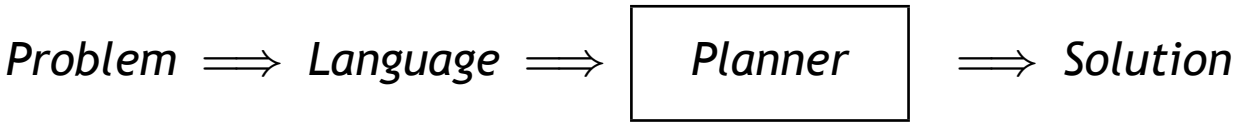
Héctor Geffner

ICREA & Universitat Pompeu Fabra
Barcelona, Spain

Joint work with B. Bonet, H. Palacios, . . .

Motivation

- Planning is a form of **general problem solving**

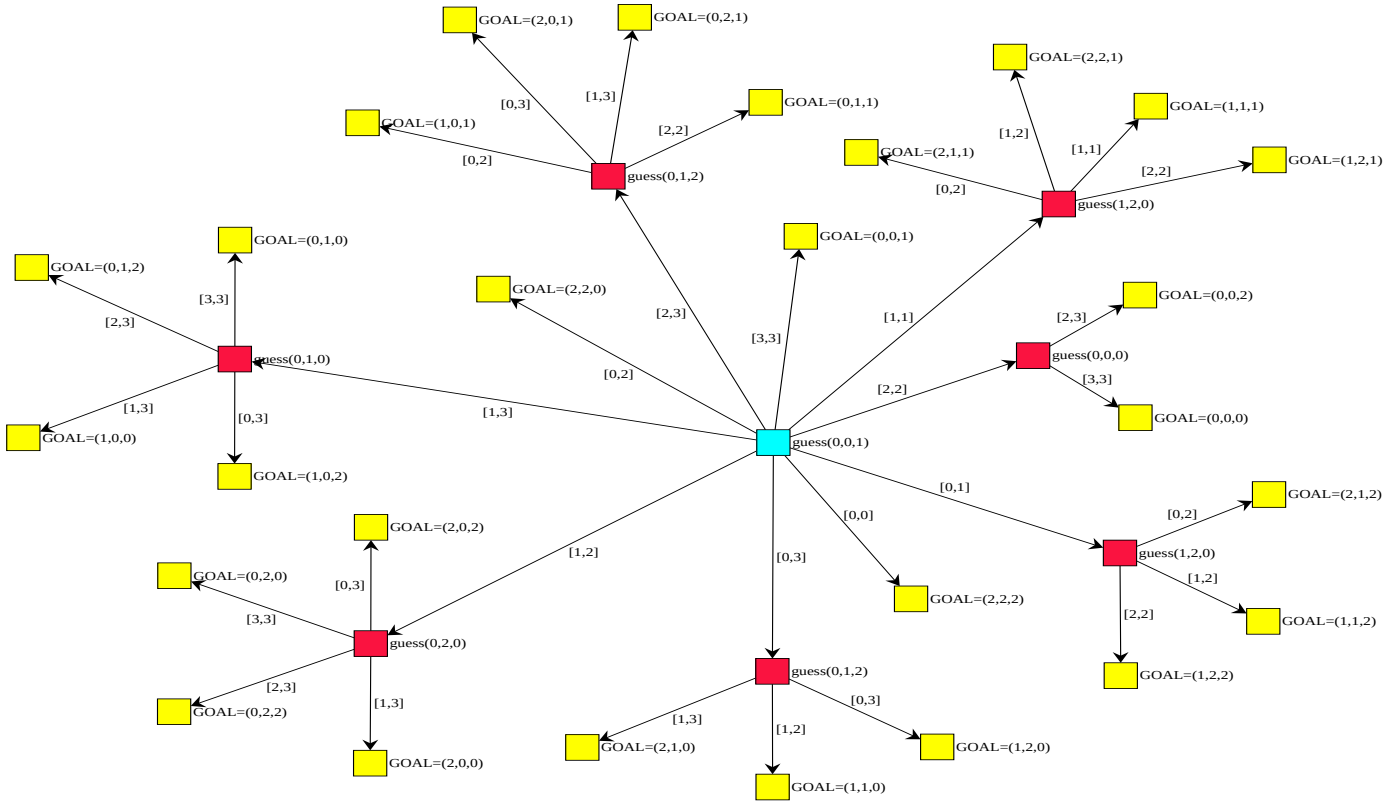


- **Idea:** problems **described** at high-level and **solved** automatically
- **Goal:** facilitate modeling with small penalty in performance

Example: Mastermind

```
(define (domain mastermind)
  (:objects d0 d1 d2 d3 - [0,3] ..)

  (:action guess ?x0 ?x1 ?x2 ?x3 - [0,3]

      :observation (+ (:if (= ?x0 d0) 1 0)
                      (+ (:if (= ?x1 d1) 1 0)..))

                   (+ (:if (hit ?x0) 1 0)
                     (+ (:if (hit ?x1) 1 0)..)))

  (:predicate (hit ?x - [0,3])
     (:or (= ?x d0) (= ?x d1) (= ?x d2) .. )))

(define (problem m4)
  (:domain mastermind)
  (:init (:set d0 :in [0,3])
         (:set d1 :in [0,3]) ..)
  (:goal :certainty))
```

Optimal Strategy for Mastermind (3,3)



AI Planning and Autonomous Behavior

Three approaches in AI to the problem of **action selection** or **control**

1. **Programming: specify** control by hand

2. **Planning:** specify **problem** by hand, **derive** control automatically

3. **Learning: learn** control from experience

Planning is the **model-based** approach to action selection: produces the behavior from the model (**solves** the model)

State model for Classical AI Planning

- finite and discrete state space S
- an initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function $s' = f(a, s)$ for $a \in A(s)$
- uniform action costs $c(a, s) = 1$

A **solution** is a sequence of applicable actions that maps s_0 into S_G

An **optimal** solution minimizes sum of action costs

Models in Planning and elsewhere

- Mathematical models in Operations Research and Control Theory
  - *linear programs*
  - *integer programs*
  - *differential equations* + *quadratic costs*
  - . . .

- AI Planning differs in two ways
  - concerned with **different** class of models
  - models represented implicitly in **language**

Planning Languages

Play two roles: **specification** (concise model description); **computation** (reveal useful heuristic information)

- A **problem** in Strips is a tuple $\langle A, O, I, G \rangle$:
  - A stands for set of all **atoms** (boolean vars)
  - O stands for set of all **operators** (actions)
  - $I \subseteq A$ stands for **initial situation**
  - $G \subseteq A$ stands for **goal situation**

- Operators $o \in O$ **represented** by

  -- the **Add** list $Add(o) \subseteq A$
  -- the **Delete** list $Del(o) \subseteq A$
  -- the **Precondition** list $Pre(o) \subseteq A$

From Language to Models

A Strips problem $P = \langle A, O, I, G \rangle$ determines **state model** $\mathcal{S}(P)$ where

- the states $s \in S$ are **collections of atoms** from A
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are s.t. $Prec(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

-- (Optimal) **Solution** of P is (optimal) **solution** of $\mathcal{S}(P)$

-- Planning over this class of models called **Classical Planning**; other models possible too

Progress in Classical Planning

- **empirical methodology**
  - standard PDDL language
  - planners and benchmarks available
  - focus on performance, planning competitions, . . .
- **novel ideas and formulations**
  - Graphplan, SAT, HSP, FF, . . .
- **large problems solved fast**

Example: Logistics in Strips/PDDL

```
(define (domain logistics)
  (:requirements :strips :typing :equality)
  (:types airport - location  truck airplane - vehicle  vehicle packet - thing  ...)
  (:predicates (loc-at ?x - location ?y - city) (at ?x - thing ?y - location) ...)
  (:action load
    :parameters (?x - packet ?y - vehicle)
    :vars (?z - location)
    :precondition (and (at ?x ?z) (at ?y ?z))
    :effect (and (not (at ?x ?z)) (in ?x ?y)))
  (:action unload ..)
  (:action drive
    :parameters (?x - truck ?y - location)
    :vars (?z - location ?c - city)
    :precondition (and (loc-at ?z ?c) (loc-at ?y ?c) (not (= ?z ?y)) (at ?x ?z))
    :effect (and (not (at ?x ?z)) (at ?x ?y)))
...
(define (problem log3_2)
  (:domain logistics)
  (:objects packet1 packet2 - packet  truck1 truck2 truck3 - truck  airplane1 - airp
  (:init (at packet1 office1) (at packet2 office3) ...)
  (:goal (and (at packet1 office2) (at packet2 office2))))
```

Computation: how to search for plans?

Heuristic Search Planning is a simple and common approach

- Extracts heuristic function $h(s)$ **automatically** from problem P
- Uses h to guide the search for a plan in the state space $S(P)$

Heuristic values $h(s)$ provide **cost estimate** to reach goal from s

Question: how to get $h(s)$ automatically from P?

Heuristic Functions: h^+

- Heuristics derived as **optimal** cost function of **relaxed** problems
- A common relaxation P^+ obtained by dropping the **delete-lists**
- If $c^*(P)$ is the optimal cost of P, then heuristic $h^+(P)$ defined as

$$h^+(P) \stackrel{\text{def}}{=} c^*(P^+)$$

- Heuristic h^+ **intractable** but easy to **approximate**
  - *computing* **optimal** *plan for* P^+ *is* **intractable**
  - *computing a non-optimal plan for* P^+*, if one exists, is easy*
- Heuristics used in HSP and FF are approximations of h^+

Lessons from Classical Planning

- **Classical planning works!!**
  - *Large problems solved very fast*

Lessons from Classical Planning

- **Classical planning works!!**
  - *Large problems solved very fast*
- **Model simple but useful**
  - *Operators not primitive; can be policies themselves*
  - *Fast closed-loop replanning able to cope with uncertainty sometimes*

Lessons from Classical Planning

- **Classical planning works!!**
  - *Large problems solved very fast*
- **Model simple but useful**
  - *Operators not primitive; can be policies themselves*
  - *Fast closed-loop replanning able to cope with uncertainty sometimes*
- **Limitations**
  - *Simple Cost Structure; state-dependent costs, preferences*
  - *Incomplete Information (Partial Observability) . . .*

Beyond Classical Planning: Two Strategies

1. **Develop** solver for more general class of models; e.g., POMDPs

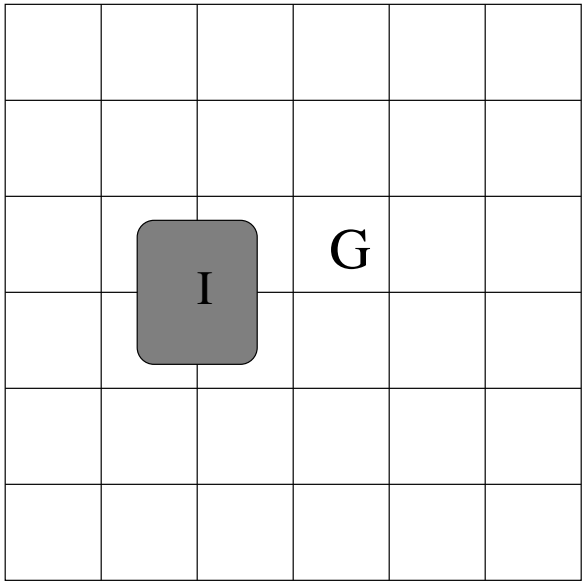
   $+$: generality
   $-$: complexity

2. **Extend** the scope of current 'classical' solvers

   $+$: efficiency
   $-$: incompleteness

We will pursue both approaches for dealing with **incomplete information** (conformant planning) and planning with **preferences** (rewards)

Incomplete Information makes Planning Harder



Problem: A robot must move from an **uncertain** I into G with **certainty**, one cell at a time, in a grid nxn

- Conformant and classical planning look similar except for uncertain I
- Yet plans may be quite different: best **conformant plan** above **must move the robot to a corner first!**

Model for Conformant Planning

- a **set** of possible initial states $b_0 \subseteq S$
- a set $b_F \subseteq S$ of possible goals
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a **non-deterministic** function F s.t. $F(a, s)$ is the **set** of next states

-- *call a set of possible states, a* **belief state**

-- *actions then map a belief state* b *into a belief state* b_a

$$b_a \stackrel{def}{=} \{s' \ | s' \in F(a, s) \ \& \ s \in b\}$$

-- **task** *is to find* **action sequence** *that maps* b_0 *into target* b_F

Computing Conformant Plans

- **Complexity:** conformant planning harder than classical planning
  - *search takes place in* **belief space** *that is exponentially larger*
  - *bounded classical planning is NP-C while conformant* $\Sigma_2^P = NP^{NP}$

- **Computation:** search in belief space using an heuristic $h(bel)$
  - *belief states represented symbolically (formulas, OBDDs, . . .)*
  - *key problem:* **no good heuristics** $h(bel)$ **for belief space yet**

Heuristic for Belief Space from Classical Heuristics

Consider a 1xn grid where the goal G is to be at position 4; boolean X_i encodes position X of robot, initially unknown.

Heuristic for Belief Space from Classical Heuristics

Consider a 1xn grid where the goal G is to be at position 4; boolean X_i encodes position X of robot, initially unknown.

- While X_i initially unknown, after a single **right** move, it is **known** that X is **not in left column:** $K\neg X_1$

Heuristic for Belief Space from Classical Heuristics

Consider a 1xn grid where the goal G is to be at position 4; boolean X_i encodes position X of robot, initially unknown.

- While X_i initially unknown, after a single **right** move, it is **known** that X is **not in left column**: $K\neg X_1$
- Also if $K\neg X_1$, another **right** move makes $K\neg X_2$ true, . . .

Heuristic for Belief Space from Classical Heuristics

Consider a 1xn grid where the goal G is to be at position 4; boolean X_i encodes position X of robot, initially unknown.

- While X_i initially unknown, after a single **right** move, it is **known** that X is **not in left column:** $K\neg X_1$
- Also if $K\neg X_1$, another **right** move makes $K\neg X_2$ true, . . .
- Eventually KX_n (agent at rightmost position) follows from

$$K(X_1 \vee X_2 \vee \cdots \vee X_n) \quad \& \quad K\neg X_1 \ \& \ K\neg X_2 \ \& \ \cdots \ \& \ K\neg X_{n-1}$$

Heuristic for Belief Space from Classical Heuristics

Consider a 1xn grid where the goal G is to be at position 4; boolean X_i encodes position X of robot, initially unknown.

- While X_i initially unknown, after a single **right** move, it is **known** that X is **not in left column:** $K\neg X_1$
- Also if $K\neg X_1$, another **right** move makes $K\neg X_2$ true, . . .
- Eventually KX_n (agent at rightmost position) follows from

$$K(X_1 \vee X_2 \vee \cdots \vee X_n) \quad \& \quad K\neg X_1 \ \& \ K\neg X_2 \ \& \ \cdots \ \& \ K\neg X_{n-1}$$

- Reaching the goal KX_4 from KX_n is simple (no incomplete info)

Heuristic for Belief Space from Classical Heuristics

Consider a 1xn grid where the goal G is to be at position 4; boolean X_i encodes position X of robot, initially unknown.

- While X_i initially unknown, after a single **right** move, it is **known** that X is **not in left column:** $K\neg X_1$
- Also if $K\neg X_1$, another **right** move makes $K\neg X_2$ true, . . .
- Eventually KX_n (agent at rightmost position) follows from

$$K(X_1 \vee X_2 \vee \cdots \vee X_n) \quad \& \quad K\neg X_1 \ \& \ K\neg X_2 \ \& \ \cdots \ \& \ K\neg X_{n-1}$$

- Reaching the goal KX_4 from KX_n is simple (no incomplete info)

Can this reasoning be accounted for in a classical planner?

Encoding the conformant problem as a classical problem

A **classical planner** can solve this **conformant problem** provided that problem encoding is modified slightly:

1. Actions such as **move right** extended with effects

$$KX_i \rightarrow KX_{i+1} \qquad i < n$$

$$K\neg X_i \rightarrow K\neg X_{i+1} \qquad i < n$$

2. Model extended with **extra, 'dummy' action** with effects

$$K\neg X_1 \wedge \cdots \wedge K\neg X_{i-1} \wedge K\neg X_{i+1} \wedge \cdots \wedge K\neg X_n \quad \rightarrow \quad KX_i \qquad i = 1, \ldots, n$$

Can this transformation be done automatically?

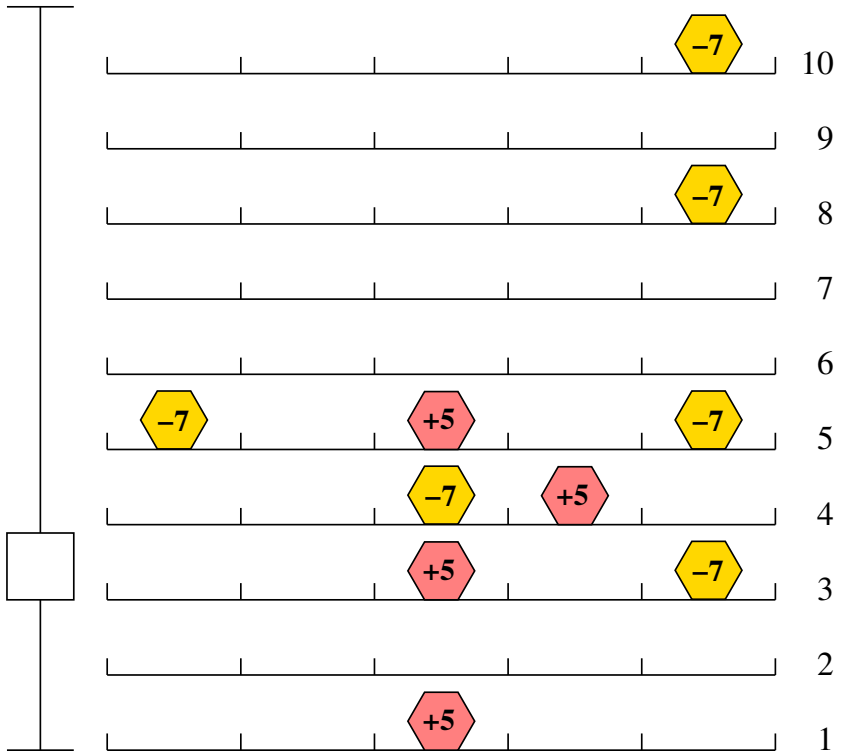
Translation of Conformant into Classical Problems

- Re-coding of **conformant** problem into **classical** one, can be generalized and extended
- A general **translator** `cs2cf` proposed in [Palacios & Geffner, AAAI-06]
- Translation is **sound**, **efficient**, and pretty **powerful** as it captures plans whose **validity proof** requires **disjunctive reasoning**
- The translation is **not complete** though, as **complex disjunctive proofs** (nested subderivations in Fitch-style systems) not captured
- Still, it deals with almost all benchmarks where it **scales up** very well

Some Empirical Results

	cf2cs(ff)		CFF	
Problem	Time	Length	Time	Length
Retrieve-4-1	0.02	41	0.04	33
Retrieve-4-2	0.08	75	0.23	49
Retrieve-4-3	0.17	91	0.8	65
Retrieve-8-1	1.18	220	204.68	210
Retrieve-8-2	3.13	291	----	----
Retrieve-8-3	132.49	415	----	----
Retrieve-12-1	----	----	----	----
Dispose-4-1	0.02	37	0.12	39
Dispose-4-2	0.05	54	0.47	56
Dispose-4-3	0.09	71	1.49	73
Dispose-8-1	1.83	265	361	227
Dispose-8-2	2.87	280	----	----
Dispose-8-3	6.87	367	----	----
Dispose-12-1	----	----	----	----
Push-4-1	0.07	41	0.09	33
Push-4-2	0.24	75	0.41	49
Push-4-3	0.53	91	1.23	65
Push-8-1	3.29	220	----	----
Push-8-2	12.89	291	----	----
Push-8-3	----	----	----	----
Push-to-3-1	0.32	21	0.03	29
Push-to-4-1	----	----	0.48	46

Planning with Penalties and Rewards: Example



- **Elevator Problem** 10-5-1 with 10 floors, 5 positions, 1 elevator.
- **Penalties** and **rewards** associated with positions; no **hard goals**.
- **Question:** How to model and solve these problems effectively?

Some Results using Heuristic h_c^+

Elevator instance n-m-k has n floors, m positions, and k elevators

Problem	Length	Optimal Cost	Solved with h_c^+		Solved blind	
			Time	Nodes	Time	Nodes
4-4-2	12	-9	0.35	1,382	4.19	29,247
6-6-2	23	-14	21.44	24,386	2,965.90	6,229,815
6-6-3	23	-14	133.48	76,128	---	---
10-5-1	11	-3	0.39	238	161.85	445,956
10-5-2	32	-5	330.72	189,131	---	---

- Admissible heuristic h_c^+, a generalization of classical h^+, handles **costs** and **rewards**, and enables the **optimal** solution of large problems
- E.g., 10-5-1 solved optimally using h_c^+ in 0.39 secs and 238 nodes; while blind search requires 161.85 secs and $445,956$ nodes

Model and Heuristic

- **Costs** $c(x)$ associated to **actions** and **atoms**, where they can be negative
- **Cost of a plan** π given by the cost of the actions and the atoms $F(\pi)$ it makes true (at some time)

$$c(\pi) \stackrel{\text{def}}{=} \sum_{a \in \pi} c(a) + \sum_{p \in F(\pi)} c(p)$$

- **Cost of a problem** P, $c^*(P)$, is the cost of the best (min) plan $c(\pi)$
- **Heuristic** $h^+_c(P)$ defined in terms of the **delete-relaxation** P^+:

$$h^+_c(P) \stackrel{\text{def}}{=} c^*(P^+)$$

- Heuristic h^+_c is **informative** and **admissible** . . .

Computation of the Heuristic h_c^+

- Unlike h^+ in classical planning, h_c^+ not only must estimate cost to goal, but **must select the goals too!** (rewards are **soft goals**)

Computation of the Heuristic h_c^+

- Unlike h^+ in classical planning, h_c^+ not only must estimate cost to goal, but **must select the goals too!** (rewards are **soft goals**)

- In [Bonet & Geffner, KR-06], a **circuit** is defined whose input is a state s, and whose output, computed in **linear time**, is $h_c^+(s)$ for **any** s

Computation of the Heuristic h_c^+

- Unlike h^+ in classical planning, h_c^+ not only must estimate cost to goal, but **must select the goals too!** (rewards are **soft goals**)

- In [Bonet & Geffner, KR-06], a **circuit** is defined whose input is a state s, and whose output, computed in **linear time**, is $h_c^+(s)$ for **any** s

  – **Heuristic** $h_c^+(s)$ *formulated as* **rank** $r(T(P) \wedge I(s))$ *of a suitable propositional theory* $T(P) \wedge I(s)$ *obtained from* P *and* s*:*

$$r(T) = \min_{M \models T} r(M) \quad \textit{and} \quad r(M) = \sum_{L \in M} r(L)$$

Computation of the Heuristic h_c^+

- Unlike h^+ in classical planning, h_c^+ not only must estimate cost to goal, but **must select the goals too!** (rewards are **soft goals**)

- In [Bonet & Geffner, KR-06], a **circuit** is defined whose input is a state s, and whose output, computed in **linear time**, is $h_c^+(s)$ for **any** s

  - **Heuristic** $h_c^+(s)$ *formulated as* **rank** $r(T(P) \wedge I(s))$ *of a suitable propositional theory* $T(P) \wedge I(s)$ *obtained from* P *and* s*:*

$$r(T) = \min_{M \models T} r(M) \quad \textit{and} \quad r(M) = \sum_{L \in M} r(L)$$

  - *Rank* $r(T(P) \wedge I(s))$ **intractable** *in general but computable in* **linear time** *if* $T(P)$ **compiled** *into d-DNNF (a tractable logical form)*

Computation of the Heuristic h_c^+

- Unlike h^+ in classical planning, h_c^+ not only must estimate cost to goal, but **must select the goals too!** (rewards are **soft goals**)

- In [Bonet & Geffner, KR-06], a **circuit** is defined whose input is a state s, and whose output, computed in **linear time**, is $h_c^+(s)$ for **any** s

  - **Heuristic** $h_c^+(s)$ *formulated as* **rank** $r(T(P) \wedge I(s))$ *of a suitable propositional theory* $T(P) \wedge I(s)$ *obtained from* P *and* s*:*

$$r(T) = \min_{M \models T} r(M) \quad \textit{and} \quad r(M) = \sum_{L \in M} r(L)$$

  - *Rank* $r(T(P) \wedge I(s))$ **intractable** *in general but computable in* **linear time** *if* $T(P)$ **compiled** *into d-DNNF (a tractable logical form)*

- The **circuit** is the compiled $T(P)$ formula; the compilation may take exponential time and space, but not necessarily so (like OBDDs)

Summary

- Planning is the **model-based** approach to intelligent behavior
- **Heuristic/Evaluation** functions h play key role in solution of many models
- Different ways for **computing** (and using) such functions:
  - *Find a relaxed-plan (h_{FF})*
  - *Find a best model (h_c^+)*
  - *Learn it by udpates (Reinf Learning and Dyn Programming)*
  - *Handcraft it (Chess)*
- Methods can be extended **effectively** to deal with simple forms of **incomplete information** and **preferences**
- Many challenges ahead . . .

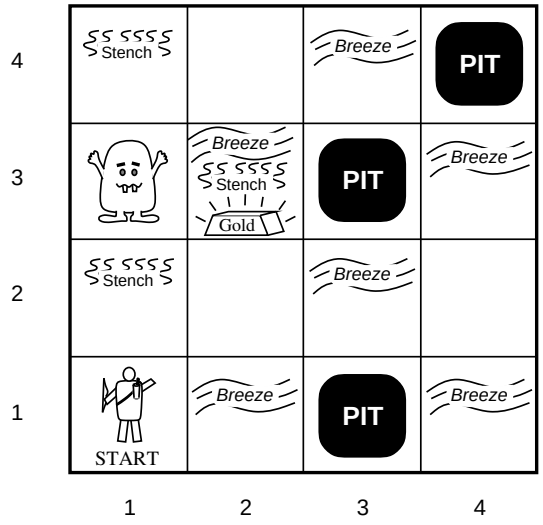
Wumpus World PEAS description

Performance measure
- gold +1000, death -1000
- -1 per step, -10 for using the arrow

Environment
- Squares adjacent to wumpus are smelly
- Squares adjacent to pit are breezy
- Glitter iff gold is in the same square
- Shooting kills wumpus if you are facing it
- Shooting uses up the only arrow
- Grabbing picks up gold if in same square
- Releasing drops the gold in same square

Actuators Left turn, Right turn,
Forward, Grab, Release, Shoot

Sensors Breeze, Glitter, Smell



Basic Translation: Conformant into Classical

THEOREM: If plan solves $K_0(P)$, then it solves P

where $P = \langle F, O, I, G \rangle$ is any **conformant problem**

- F stands for the fluents in P
- O for the operators
- I for the initial situation (clauses over F-literals)
- G for the goal situation (set of F-literals)

and $K_0(P) = \langle F', O', I', G' \rangle$ is the **classical problem**

- $F' = \{KL, K\neg L \mid L \in F\}$
- $I' = \{KL, \neg K\neg L \mid L \in I\} \cup \{\neg KL', \neg K\neg L' \mid L' \notin I\}$
- $G' = \{KL \mid L \in G\}$
- $O' = O$ but with each literal L replaced by KL . . .

Extending Basic Translation: Disjunctive Reasoning

- **Action Compilation:** If $a : C \wedge \sim L \rightarrow L$ in P, $a : KC \rightarrow KL$ in $K(P)$
- **Split:** If $a : C \wedge X_i \rightarrow L$ in P, then $a : C \rightarrow L/X_i$ in $K(P)$, where L/X_i is an atom that stands for 'if X_i then L', initially **false**
- **Merge:** If $X : X_1 \vee \cdots X_n$ in P, then new action with effect

$$(L/X_1 \vee K\neg X_1) \wedge \cdots \wedge (L/X_n \vee K\neg X_n) \wedge FLAG_{X,L} \rightarrow KL$$

  in $K(P)$, where $FLAG_{X,L}$ is a boolean initialized to **true**

- **Protect:** If $a : C \rightarrow Y$ in P, then $a : C \rightarrow \neg FLAG_{X,L}$ in $K(P)$ if Y deletes certain invariants involving X and L.

Theorem: If plan solves $K(P)$, then it solves P