Presentation for course Examination: Intelligent Systems Engineering by Keith Barnes

Prof. Andrea Omicini

Classical Logic to Modern Solvers: The Evolution of PDDL

Foundations, Standardization, and Comparative Analysis of Planning Technologies

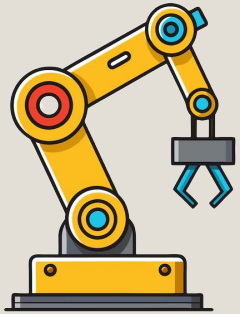
NEED FOR PLANNING

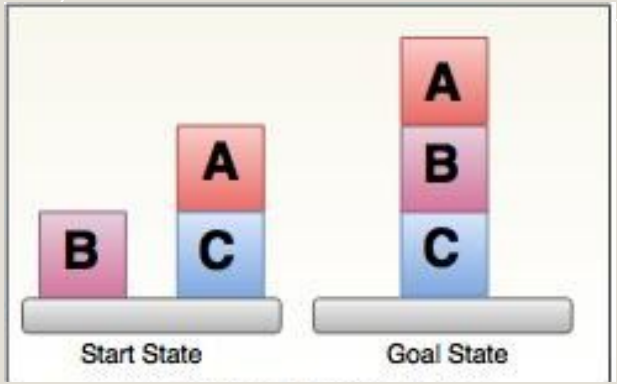
Agent to control a thermostat

IF (Temperature < target)
THEN (turn_on)

“Rely on hard coded rules”

REFLEX AGENTS





blocks	states	blocks	states
1	1	10	58,941,091
2	3	11	824,073,141
3	13	12	12,470,162,233
4	73	13	202,976,401,213
5	501	14	3,535,017,524,403
6	4,051	15	65,573,803,186,921
7	37,633	16	1,290,434,218,669,921
8	394,353	17	26,846,616,451,246,353
9	4,596,553	18	588,633,468,315,403,843

Planning: Compact Representation of such systems

THE FOUNDATION: STRIPS LOGIC (1971)

STATE

The world is a list of True facts.
For e.g. :
OnTable(BlockA)
HandEmpty

ACTIONS

Operator definition.
For e.g: **Pick_A**
Precondition: what must already be true? (**HandEmpty**)
Effect:
Add: Holding_A
Delete: HandEmpty

THE SIMPLE LOGIC

NEXT STATE = (CURRENT STATE - DELETE) + ADD

STRIPS defined the logic of Actions. By assuming “Unlisted = False” (The Closed World Assumption), it made planning mathematically efficient.

Need for a standard

Strips gave logic, but **not the standard**

- Researcher A: defined his problem in **.lisp**
- Researcher B: defined his solution in **.cpp**

How to compare and benchmark ?

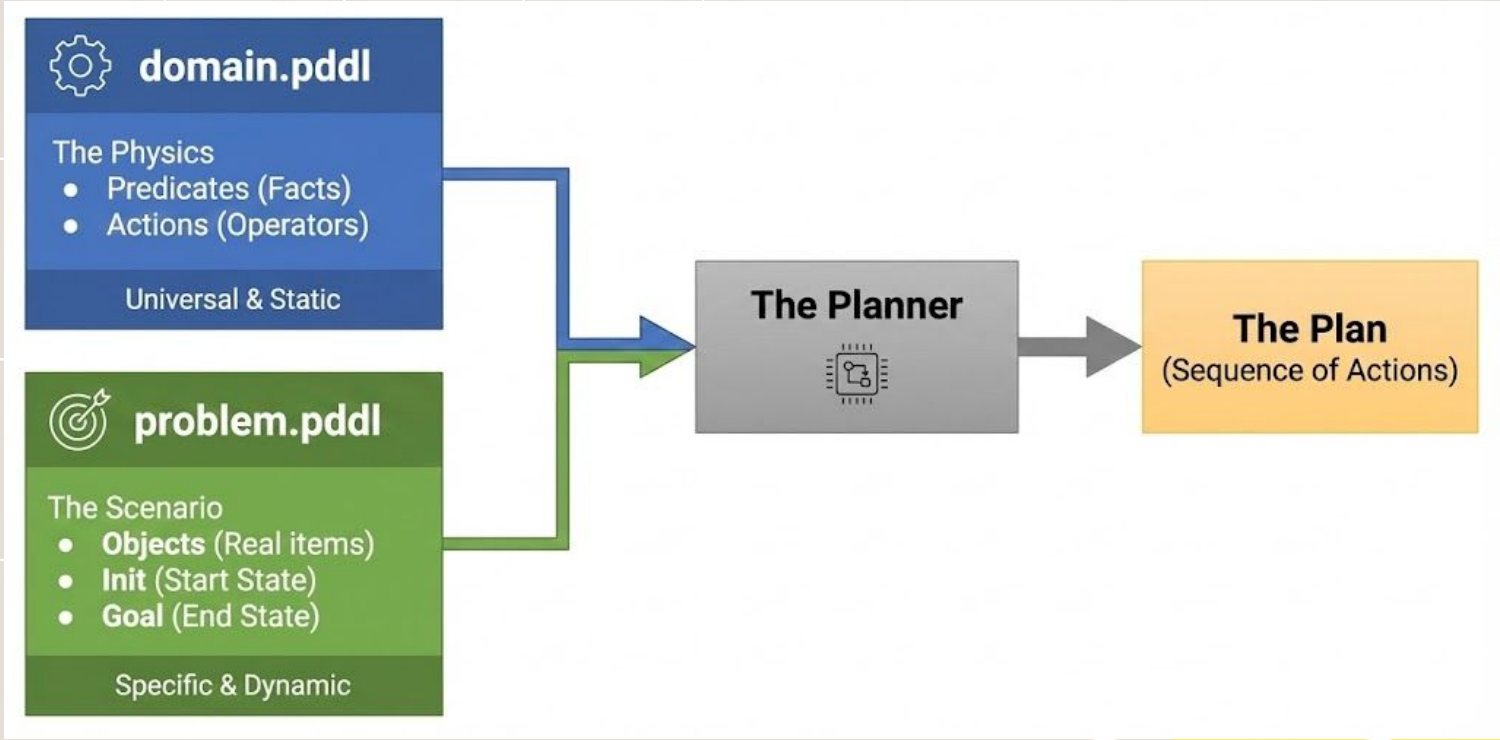
International Planning Competition

- established 1998

Need for a standard - **PDDL was defined**

PDDL Architecture

Planning domain definition Language



Anatomy: Domain file

The Rules

```
(define (domain blocksworld)
  (:requirements :strips)

  ;; PREDICATES: The Vocabulary (True/False Facts)
  (:predicates
    (on ?x ?y)          ; Block x is on Block y
    (ontable ?x)        ; Block x is on the table
    (clear ?x)          ; Nothing is on top of x
    (handempty)         ; Robot hand is free
    (holding ?x)        ; Robot is holding x
  )

  ;; ACTION: The Logic Rule for Stacking
  (:action stack
    :parameters (?x ?y)
    :precondition (and (holding ?x) (clear ?y))
    :effect (and (not (holding ?x))
                 (not (clear ?y))
                 (clear ?x)
                 (handempty)
                 (on ?x ?y))
  )
)
```

The Dictionary
Defines valid facts. If a fact isn't listed here (like 'color'), it doesn't exist in this world.

Abstraction
Variables (?x) allow this one rule to work for any block (A, B, or C).

Requirements
The 'IF'. The robot checks these facts before moving.

Update Logic
The 'THEN'. Using (not ...) removes facts (Delete List), and the rest are added (Add List).

Anatomy: Problem file

The Scenario

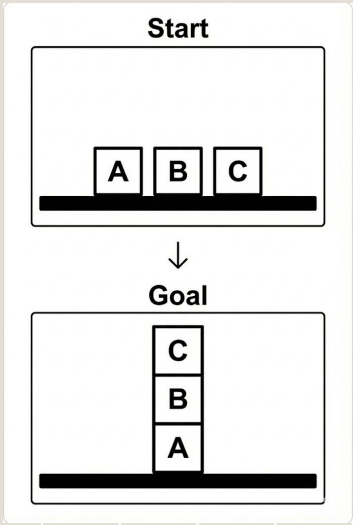
```
(define (problem stack-blocks-task)
  (:domain blocksworld)  ;; Links to the Rules file

  ;; OBJECTS: The actual entities
  (:objects
    blockA blockB blockC
  )
  ;; INIT: The "Start State"
  ;; (Note: Anything NOT listed here is FALSE)
  (:init
    (ontable blockA)
    (ontable blockB)
    (ontable blockC)
    (clear blockA)
    (clear blockB)
    (clear blockC)
    (handempty)
  )
  ;; GOAL: The "Destination"
  (:goal
    (and (on blockA blockB) (on blockB blockC))
  )
)
```

The Link
Connects this specific problem to the general rules defined in the domain file.

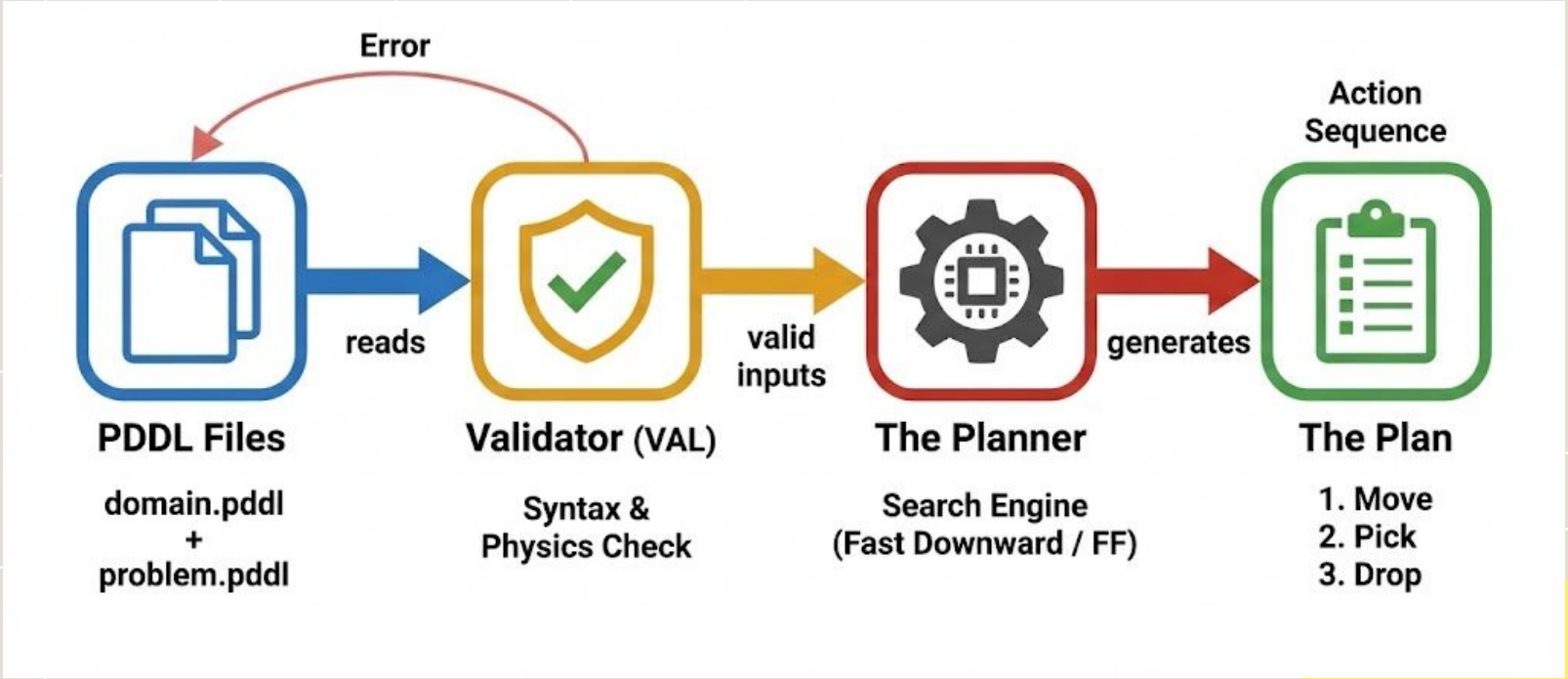
The Entities
Lists the concrete items (blocks A, B, C) that exist in this specific scenario.

The Start State
Defines the initial setup. Remember the 'Closed World Assumption': unlisted facts are false.

The Objective
Describes the desired final state the planner needs to achieve.



The Execution pipeline

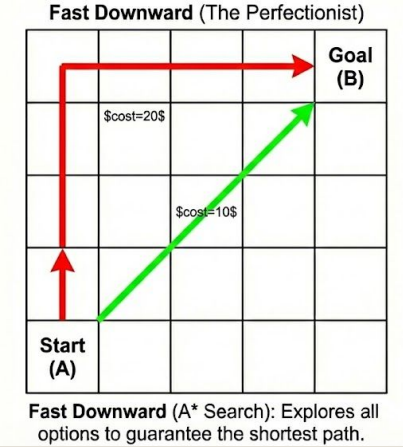


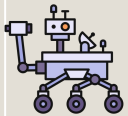
Heuristic planners

Smart guess

Fast Downward – Optimal

- A Search* algorithm
- F (score) = g (past cost) + h (future guess)
- Guarantees shortest path - but slow

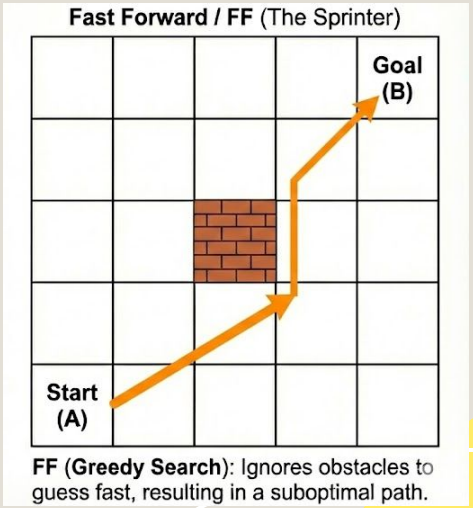




Uses case: MARS ROVER - limited resources sacrifice, computational time to find best path

Fast Forward – satisficing

- Ignore the past and only looks at the future to guess fast
- Ignores the negative effects
- Focus is on the speed



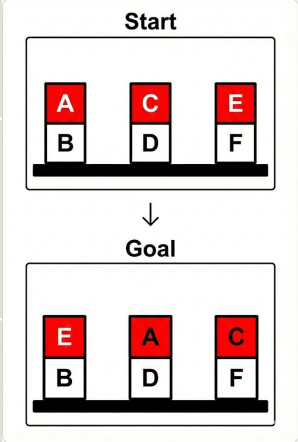
Might not be optimal
But fast!

E.g – floor cleaning robot

COMPARISON
DEMO

Simulated on PDDL Editor
Delfi – fast downward
Optimal Solution – 8 Steps – slow

```
Solution found!
Actual search time: 0.00031476s [t=3.19768s]
unstack a b (1)
put-down a (1)
unstack e f (1)
stack e b (1)
unstack c d (1)
stack c f (1)
pick-up a (1)
stack a d (1)
Plan length: 8 step(s).
Plan cost: 8
Expanded 11 state(s).
Reopened 0 state(s).
Evaluated 30 state(s).
Evaluations: 30
Generated 40 state(s).
Dead ends: 0 state(s).
```



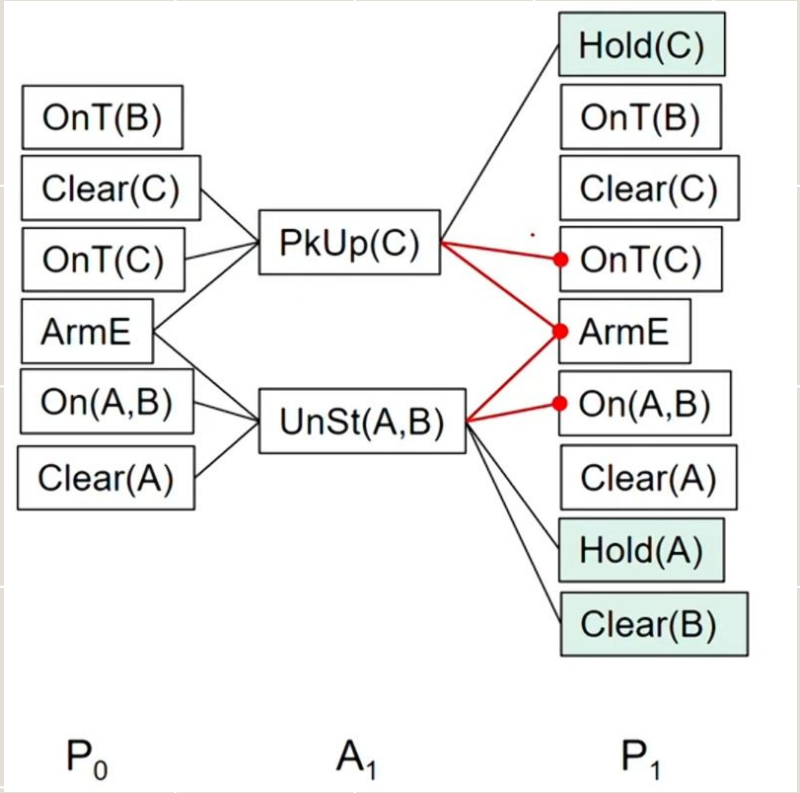
FF planner*
18 steps – fast

```
unstack a b (1)
put-down a (1)
unstack c d (1)
stack c a (1) <-- MISTAKE: Stacks C on A just to get it out of the way
unstack e f (1)
stack e d (1) <-- MISTAKE: Stacks E on D (filling the spot C left)
unstack c a (1) <-- CORRECTION: Realizes it needs A free
put-down c (1)
pick-up a (1) <--  Tries to stack A on D...
put-down a (1) <-- ...but realizes D is covered by E!
unstack e d (1) <-- CORRECTION: Moving E out of the way
put-down e (1)
pick-up a (1)
stack a d (1) <-- GOAL 1 ACHIEVED
pick-up c (1)
stack c f (1) <-- GOAL 2 ACHIEVED
pick-up e (1)
stack e b (1) <-- GOAL 3 ACHIEVED
Plan length: 18 step(s).
```

***Proposed**

Graphplan

The layered approach



- Graphplan builds a graph layer by layer
- (p0 -> a1 -> p1)
- Considering all actions and effects
- Highlighting negative effects

- Builds the Graph till goal state appears in the future layer

- Backward search -avoiding red lines

- Pro: can schedule actions in parallel
- Con: high memory usage, not ideal for long simple chains

SatPlan

The encoding approach

THE LOGIC:

$$A_{act} \rightarrow (P_{pre} \wedge E_{eff})$$

- A_{act}: Action (Pick Up)
- P_{pre}: Precondition (On Table)
- E_{eff}: Effect (Holding)

"Can we represent the problem as an equation?"

- Satplan translates it into a single logical equation
- Feeds the equation to a solver -> returns the plan

- Pros: handles strict constraints well
- Cons: encoding explosion: simple problem might need multiple variables - memory heavy

Action_PickUp(A)=>(OnTable(A) ^ (HandEmpty) ^ Holding(A)

The PDDL Toolbox

Selecting the right solver

PDDL introduced standardization and decoupling – allowing field to advance rapidly

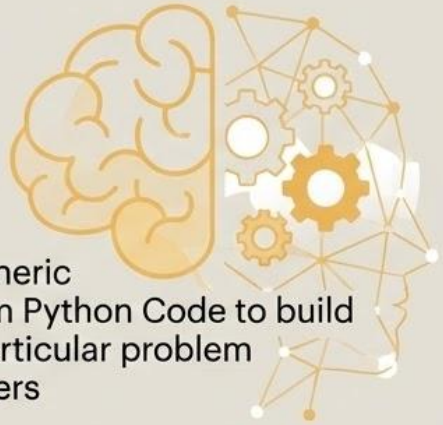
No best planner – trade off between time, optimization and constraints.

Need Speed? -> Use **FF (Satisficing).**

Need Perfection? -> Use **Fast Downward (Optimal).**

Need Speed + Parallelism? -> Use **Graphplan.**

Have Complex Constraints? -> Use **SatPlan.**



Ongoing research

LLM-Generated Heuristics:

- Traditional planners are generic
- Trained LLMs writes custom Python Code to build optimized heuristics for particular problem
- Leading to Generative solvers

Combining Creativity of LLMs and Reliability of PDDL.

Allows agents to mathematically derive the best solution instead of following instructions.

Thank you