INF2D – Reasoning and Agents

Coursework 2: Symbolic Planning

TA: Jay Park (jay.jh.park@ed.ac.uk)

Learning Objectives

- Gain hands-on experience of designing ‘effective & efficient’ formal representations of planning problems
  - Learn how to code in PDDL, a declarative language for formally expressing planning problems

- Understand how performance of a planning algorithm may be affected by different factors, including:
  - Parameters to the planning algorithm
  - Your design choices!

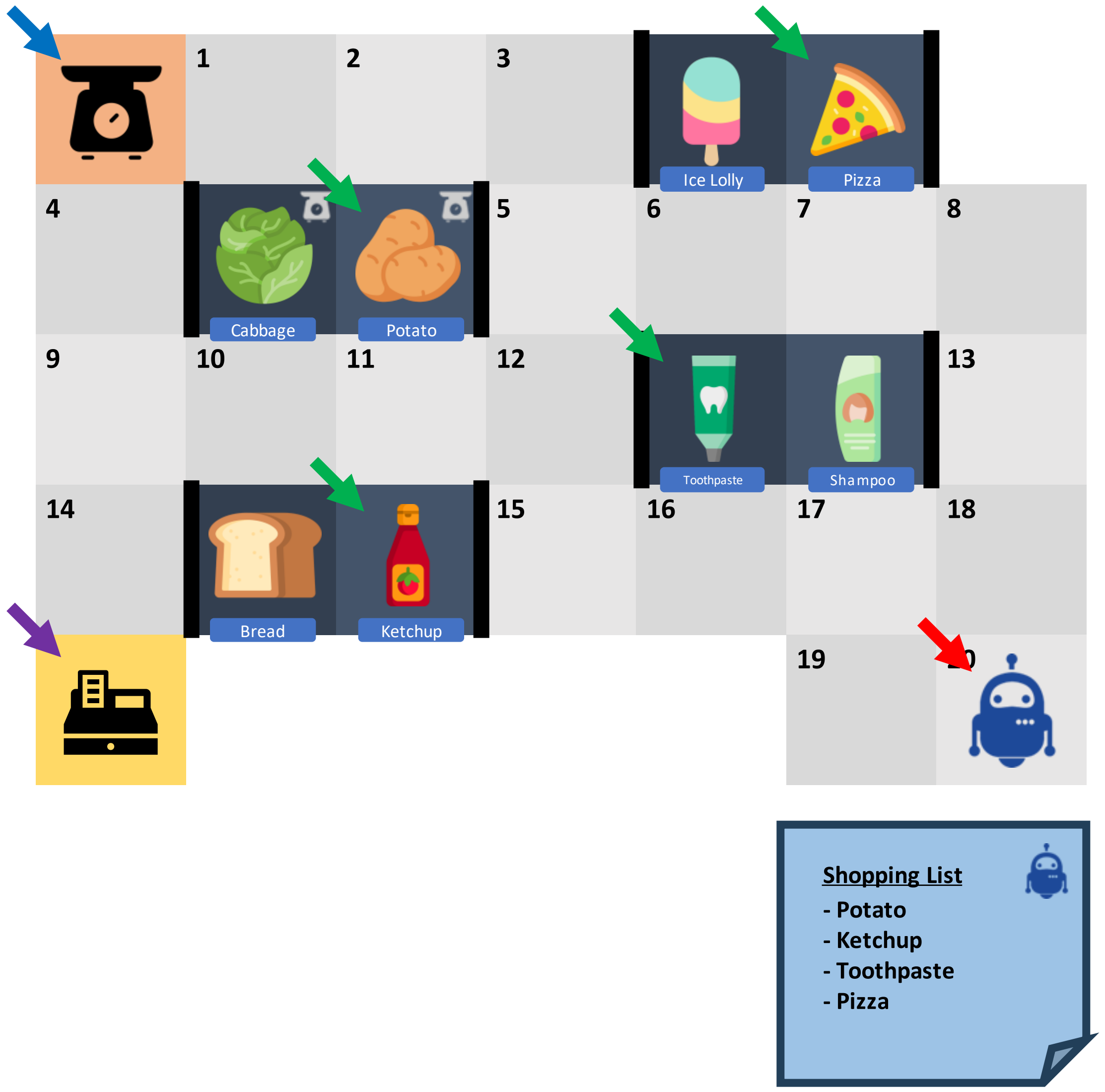
Coursework outline

- High-level goal: Design symbolic planning domains that model scenarios involving a robot shopping in a supermarket
    - Open-ended task requiring your own decisions about abstraction
    - No single ‘correct’ design… but there are such things as *inefficient* designs!

- CW2 is broken down into three tasks:
    - Modelling (35%)
    - Experiment (15%)
    - Extensions (50%)

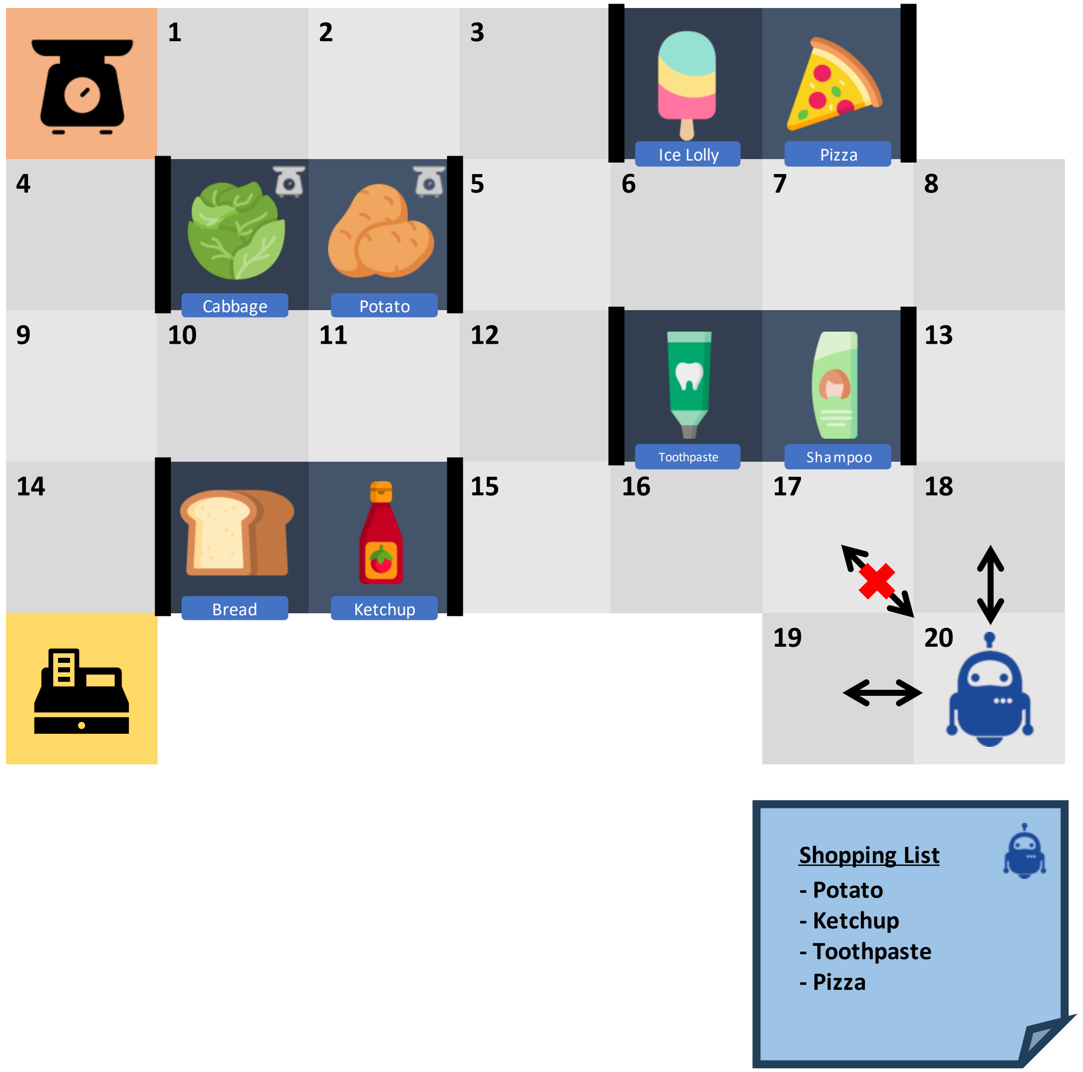
Task 1: Modelling

- Scenario
  - A ShopBot is tasked to shop for *all* of the shopping items specified by the provided shopping list
  - A shopping item can be picked up if ShopBot is standing next to the shelf containing the item
    - If an item needs weighing (), ShopBot needs to weigh it at the weighing scale before checking it out
  - An item is considered 'shopped' if it is placed on the checkout stand and checked out



Task 1: Modelling

- Scenario (cont'd)
  - The layout of a supermarket consists of aisle cells (numbered squares in the figure), shelves (squares with shopping items), a weighing scale and a checkout stand
  - A MINEBOT can occupy and move between two adjacent aisle cells
    - No diagonal movements!
  - A MINEBOT can pick up at most one object – iff it is not already holding anything



Task 1: Modelling

- Programming in PDDL (Planning Domain Definition Language)
  - A declarative programming language
    - Specifies *what* problems to solve, NOT *how* to solve problems
    - Solving of the planning problems is entirely delegated to the Metric FF planner
  - A PDDL representation of planning problems consists of:
    - One PDDL domain file, defining the “universal” aspects of problems
    - One or more PDDL problem files, each instantiating a particular planning problem

Task 1: Modelling

- Anatomy of a PDDL domain file
  - (Example in handout)

```
(define (domain blocks-world)
    (:requirements :adl)

    (:types table block)

    (:predicates
        (On ?x - block ?y - object)
        (Clear ?b - object)
    )

    (:constants Table - table)
```

(cont'd in next slide…)

Some domain name of your choice

Similar to importing packages in other languages; would only need minimal modifications when necessary

Declaration of object 'types' in this domain

Declaration of predicates, by their names and arguments

Declaration of 'constant' entities that will be present in all problem instances in the domain

Task 1: Modelling

- Anatomy of a PDDL domain file
  - (Example in handout)

Declaration of an action schema, by its name, parameters, preconditions and effects

(cont'ing from previous slide…)

```
    (:action MOVE-TO-TABLE
        :parameters (?b - block ?x - block)
        :precondition (and (On ?b ?x) (Clear ?b) (not (= ?b ?x)))
        :effect (and (On ?b Table) (Clear ?x) (not (On ?b ?x)))
     )
 )
```

Implementation of:
Action(MoveToTable(b, x),
Precond: $On(b, x) \land Clear(b) \land b \neq x$)
Effect: $On(b, Table) \land Clear(x) \land \neg On(b, x)$)

Task 1: Modelling

- Anatomy of a PDDL problem file
  - (Example in handout)

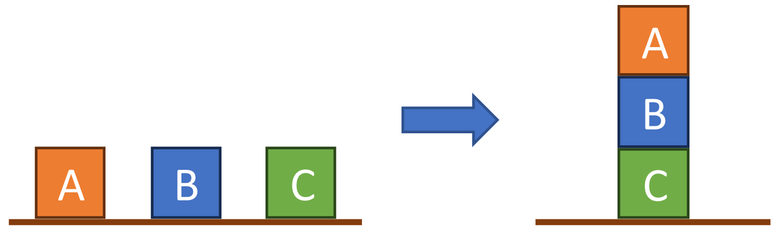
```
(define (problem block-problem)
    (:domain blocks-world)
    (:objects
        A B C - block
    )

    (:init
        (On A Table) (On B Table) (On C Table)
        (Clear A) (Clear B) (Clear C)
    )
    (:goal (and
        (On A B)
        (On B C)
    ))
)
```

Some problem name of your choice

Specification of the planning domain for the problem (defined earlier)

Objects existing in the scope of the problem and their types

Initial state specification



Goal specification

Task 1: Modelling

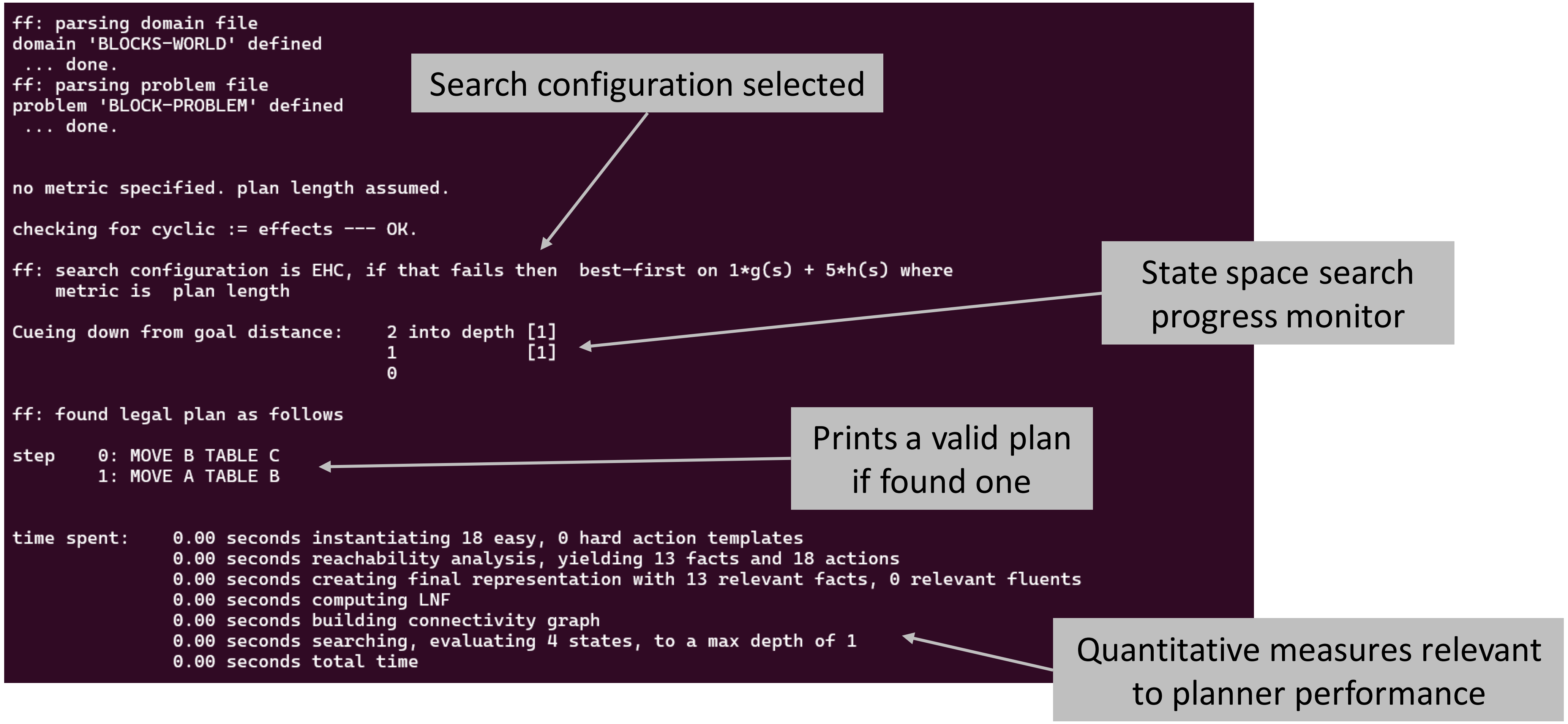
- Testing your domains & problems
  - You can test your PDDL domain & problem files locally before submitting by running the Metric FF planner included in the handout (binary executable with the name `ff`)

  - To run the planner, execute the following command on your shell, in the directory you unzipped the handout:

```
./ff -o {domain_file_name} -f {problem_file_name}
```

Task 1: Modelling

- Typical planner result output:



```
ff: parsing domain file
domain 'BLOCKS-WORLD' defined
 ... done.
ff: parsing problem file
problem 'BLOCK-PROBLEM' defined
 ... done.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

Cueing down from goal distance:    2 into depth [1]
                                   1            [1]
                                   0

ff: found legal plan as follows

step    0: MOVE B TABLE C
        1: MOVE A TABLE B


time spent:    0.00 seconds instantiating 18 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 13 facts and 18 actions
               0.00 seconds creating final representation with 13 relevant facts, 0 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 4 states, to a max depth of 1
               0.00 seconds total time
```

Search configuration selected
State space search progress monitor
Prints a valid plan if found one
Quantitative measures relevant to planner performance

Task 2: Experiment

- Now that we have our first suite of planning domain & problems, let’s conduct some experiments…

- Task 2.1: Design a harder problem
  - The planning problem instance encoded in Task 1.2 is not challenging enough for the Metric FF planner
  - Can you come up with a harder problem that would further distress the Metric FF planner?
    - Make it challenging enough that you can observe more noticeable differences in planner performance in the next task
    - Justify your design choice in your report

Task 2: Experiment

- Task 2.2: Extensive evaluation of planner performance
  - By default, the Metric FF planner runs a faster – yet incomplete – heuristics-based algorithm (enforced hill-climbing) to solve a PDDL planning problem

  - If the Metric FF fails to find a legitimate plan at the first attempt, then it falls back to a more thorough heuristics-based best-first search to try again…
    - … using the following evaluation function for a state s:

$$f(s) = w_g g(s) + w_h h(s)$$

where $g(s)$ is the actual cost accumulated so far, $h(s)$ is the *estimated* cost hereafter from s to the goal, and w_g (default: 1) & w_h (default: 5) are integer weight parameter

Task 2: Experiment

- Task 2.2: Extensive evaluation of planner performance (cont'd)
  - Experiment question: How does our choice of (w_g, w_h) affect the planner performance?
  - Design a suite of experiments to evaluate the effect of setting different (w_g, w_h) values on the performance. Analyse the results in your report.
  - To start a single run with an experiment configuration, execute the following command on your shell:

```
./ff -E -g {value of w_g} -h {value of w_h} -o {domain_file_name} -f {problem_file_name}
```

Flag for disabling the default hill-climbing search algorithm

Task 3: Extensions

- One could argue the domain we have designed in Task 1 is rather too simple and still misses out many aspects of the real-world scenarios…
- Incrementally extend your domain to accommodate the following scenarios, and write problem files as specified in the handout:
  - Task 3.1: Now a ShopBot can opt to carry a shopping basket, which can contain multiple shopping items.
  - Task 3.2: Now the prices of shopping items are explicitly considered. ShopBot should have sufficient credit balance when checking out items, and more credits can be acquired at a top-up station. (hint: numeric fluents)
  - Task 3.3: Now there might be more than one ShopBots in a supermarket shopping around. We don't want them bump into each other.

Task 3: Extensions

- Extend your domain (cont'd):
  - Task 3.4: Extra challenge; freely motivate, design and implement another real-world-like aspect of your own, describe and justify in your report

- **A word of caution**
  - **Please do not attempt** this subtask unless the CW so far was a breeze, and prior tasks took less than 10 hours to perfect
  - We will be seriously picky when marking this part; we expect extensions and reports that go well beyond our expectations to warrant a meaningful mark
    - We'd like to encourage you to prioritise other important tasks (e.g. refining answers to previous parts, other CWs, even catching up on sleep) before spending time on this subtask, in the interest of efficiency ☺

Submission

- We will use Gradescope for both submitting and marking
- Follow the submission instruction in Gradescope as specified in the handout, having your submission files named accurately
- Autograder will verify the submitted PDDL files are syntactically valid and plans are generated in a reasonable timeframe
  - You can submit as many times as you want, up until submission deadline
  - Your latest submission will be marked
  - For late submission policy, consult the handout pdf
- <u>Export your report as a pdf file</u>

Getting support

- Labs
  - CW2 clinic sessions start on Friday in Week 7 (8th March)
  - Exploit the lab sessions if you feel you could use some help from our demonstrators or fellow coursemates

- Piazza
  - Whenever you have questions, ask away; public questions are encouraged and may help other fellow students as well, though you can choose to post private questions only visible to course staff

- Please do not ask anyone to just give away solutions!
- Also, do not give away your solutions to anyone!

"That's all Folks!"