Informatics 2D Coursework 2: Symbolic Planning

Nick Ferguson and **Craig Innes**

Due at **12:00** on **27th March 2025**

Acknowledgements

Coursework Design, Slides, Teaching Assistance: **Nick Ferguson**

Learning Objectives

Gain hands-on experience of **designing effective and efficient formal representations** of planning problems by implementing a simple planning domain and problem in PDDL.

Learning Objectives

Gain hands-on experience of **designing effective and efficient formal representations** of planning problems by implementing a simple planning domain and problem in PDDL.

Understand how the **performance of a planning algorithm may be affected by different factors**, such as the parameters of the evaluation function and your design choices.

Coursework Outline

High-level goals: design, implement and evaluate domain and problem files for a **delivery courier scenario** using PDDL.

- Decisions about scenario design required - no single correct design, but there are inefficient designs!

Coursework Outline

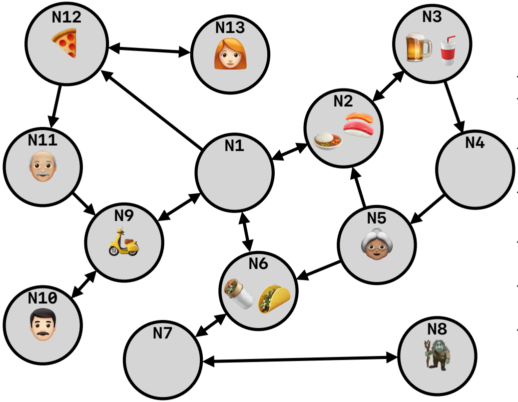
High-level goals: design, implement and evaluate domain and problem files for a **delivery courier scenario** using PDDL.

- Decisions about scenario design required - no single correct design, but there are inefficient designs!

CW2 is broken down into three components, with the coursework as a whole worth 15% of the overall course grade.

- **Modelling (35 marks)** Creating PDDL domain and problem files for a basic scenario.
- **Experiment (15 marks)** Designing an experiment to evaluate the planner.
- **Extensions (50 marks)** Extending our scenario by implementing additional features to bring it closer to the real world.

Scenario Overview

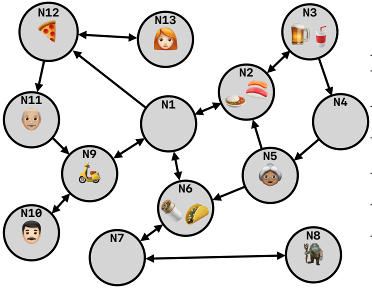


Person	Ordered
	Beer Pizza
	Sushi
	Burrito Soda
	Curry
	Taco

- A road network on which a Courier travels round, picking up and making deliveries according to the list of orders.

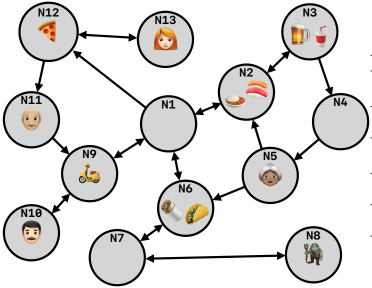
🛵 Modelling (35 marks)

- 🛵 A COURIER is required to make deliveries according to a list of orders.



Person	Ordered
👨	🍺 Beer 🍕 Pizza
👵🏽	🍣 Sushi
👩‍🦰	🌯 Burrito 🥤 Soda
👴🏼	🍛 Curry
🧌	🌮 Taco

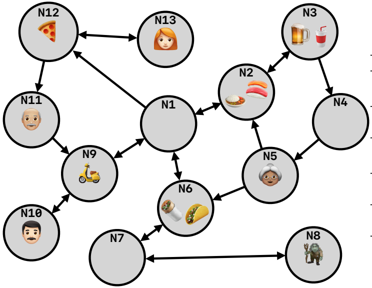
🛵 Modelling (35 marks)



Person	Ordered
👨	🍺 Beer 🍕 Pizza
👵	🍣 Sushi
👩	🌯 Burrito 🥤 Soda
👴	🍛 Curry
🧌	🌮 Taco

- 🛵 A Courier is required to make deliveries according to a list of orders.
- 🍕 Each order is required to be picked up from, and delivered to, a specific location.

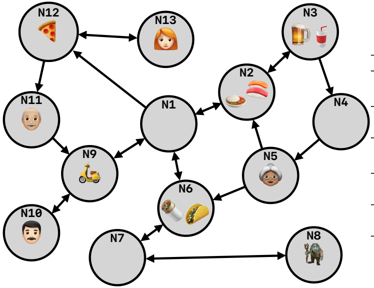
🛵 Modelling (35 marks)



Person	Ordered
👨	🍺 Beer 🍕 Pizza
👵	🍣 Sushi
👩	🌯 Burrito 🥤 Soda
👴	🍛 Curry
🧌	🌮 Taco

- 🛵 A COURIER is required to make deliveries according to a list of orders.
- 🍕 Each order is required to be picked up from, and delivered to, a specific location.
- 🌯 The courier can only pick up an item if it is at the same location as that item.

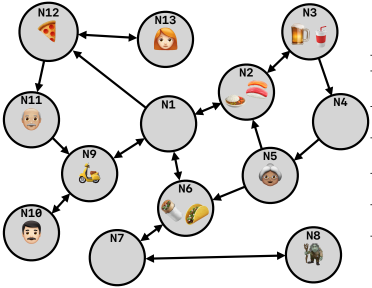
Modelling (35 marks)



Person	Ordered
	Beer Pizza
	Sushi
	Burrito Soda
	Curry
	Taco

- A Courier is required to make deliveries according to a list of orders.
- Each order is required to be picked up from, and delivered to, a specific location.
- The courier can only pick up an item if it is at the same location as that item.
- The courier can only make a delivery if they are at the location of the person who placed the order and the item is in their possession.

Modelling (35 marks)



Person	Ordered
	Beer Pizza
	Sushi
	Burrito Soda
	Curry
	Taco

- A COURIER is required to make deliveries according to a list of orders.
- Each order is required to be picked up from, and delivered to, a specific location.
- The courier can only pick up an item if it is at the same location as that item.
- The courier can only make a delivery if they are at the location of the person who placed the order and the item is in their possession.
- The road layout consists of nodes connected by edges. Each person and item is associated with a node.

🛵 Modelling (35 marks)

For this first task, you are required to create PDDL domain and problem files to implement this basic delivery courier scenario.

Domain File (20 marks)

- 🍕 Define predicates and actions necessary for modelling movement around the network, picking up deliveries, and making those deliveries.

🛵 Modelling (35 marks)

For this first task, you are required to create PDDL domain and problem files to implement this basic delivery courier scenario.

Domain File (20 marks)

- 🍕 Define predicates and actions necessary for modelling movement around the network, picking up deliveries, and making those deliveries.

Problem File (15 marks)

- 🌯 Define initial and goal states, reflecting the road layout and the list of orders.

🛵 Modelling (35 marks)

For this first task, you are required to create PDDL domain and problem files to implement this basic delivery courier scenario.

Domain File (20 marks)

- 🍕 Define predicates and actions necessary for modelling movement around the network, picking up deliveries, and making those deliveries.

Problem File (15 marks)

- 🌯 Define initial and goal states, reflecting the road layout and the list of orders.

Testing (0 marks)

- 🍣 Use the provided `ff` planner to test that your domain and problem files produce plans.

Anatomy of a PDDL Domain File

PDDL is *declarative*: specify *what* to solve, not *how* to solve it – solving is left to the `ff` planner.

Anatomy of a PDDL Domain File

PDDL is *declarative*: specify *what* to solve, not *how* to solve it – solving is left to the `ff` planner.

Let's use the classic 'block-world' scenario as an example. First we name the domain and declare requirements of our scenario.

```
(define (domain blocks-world-domain) ; Name of the domain
    (:requirements :adl) ; Action Description Language
```

Anatomy of a PDDL Domain File

PDDL is *declarative*: specify *what* to solve, not *how* to solve it – solving is left to the `ff` planner.

Let's use the classic 'block-world' scenario as an example. First we name the domain and declare requirements of our scenario.

```
(define (domain blocks-world-domain) ; Name of the domain
    (:requirements :adl) ; Action Description Language
```

We can declare types of objects in our domain.

```
(:types table block - object) ; table and block are subtypes of object
```

Anatomy of a PDDL Domain File

Next we declare predicates, represent the state of the world.

```
(:predicates
        (on ?x - block ?y - block) ; x is on y
        (clear ?x - block) ; x is clear
)
```

Anatomy of a PDDL Domain File

Next we declare predicates, represent the state of the world.

```
(:predicates
        (on ?x - block ?y - block) ; x is on y
        (clear ?x - block) ; x is clear
)
```

Actions are defined with parameters, preconditions and effects.

```
(:action MOVE
    :parameters (?x - block ?y - block) ; x, y are blocks
    :precondition (and (on ?x ?y) (clear ?x)) ; x on y and x is clear
    :effect (and (not (on ?x ?y)) (clear ?y) (not (clear ?x))) ; x is no
    longer on y, y is now clear, x is not clear
)
```

Anatomy of a PDDL Domain File

Bringing it all together...

```
(define (domain blocks-world-domain)
    (:requirements :adl)
    (:types table block - object)

    (:predicates
        (on ?x - block ?y - block)
        (clear ?x - block)
    )

    (:action MOVE
        :parameters (?x ?y - block)
        :precondition (and (on ?x ?y) (clear ?x))
        :effect (and (not (on ?x ?y)) (clear ?y) (not (clear ?x)))
    )
    ... ; other actions
)
```

Anatomy of a PDDL Problem File

The problem file specifies the initial state and goal state.

Anatomy of a PDDL Problem File

The problem file specifies the initial state and goal state.

First we name the problem and declare the domain it uses.

```
(define (problem blocks-world-problem) ; Name of the problem
    (:domain blocks-world) ; Domain used by the problem
```

Anatomy of a PDDL Problem File

The problem file specifies the initial state and goal state.

First we name the problem and declare the domain it uses.

```
(define (problem blocks-world-problem) ; Name of the problem
    (:domain blocks-world) ; Domain used by the problem
```

We can declare objects in our problem.

```
(:objects
        A B C - block ; A, B, C are blocks
        T - table ; T is a table
)
```

Anatomy of a PDDL Problem File

The initial state is defined with the `init` predicate.

```
(:init
    (on A B) ; A is on B
    (on B T) ; B is on T
    (clear C) ; C is clear
)
```

Anatomy of a PDDL Problem File

The initial state is defined with the `init` predicate.

```
(:init
    (on A B) ; A is on B
    (on B T) ; B is on T
    (clear C) ; C is clear
)
```

The goal state is defined with the `goal` predicate.

```
(:goal
    (and (on A C) (on C B)) ; A is on C and C is on B
)
```

Anatomy of a PDDL Problem File

Bringing it all together...

```
(define (problem blocks-world-problem)
    (:domain blocks-world)
    (:objects
        A B C - block ; object declaration
        T - table
    )

    (:init
        (on A B) ; initial state
        (on B T)
        (clear C)
    )

    (:goal
        (and (on A C) (on C B)) ; goal state
    )
)
```

Testing Your Domain and Problem Files: Planner Invocation

Files can be tested using the `ff` planner, which is a binary executable included in the zip file provided on Learn.

```
./ff -o domain_file.pddl -f problem_file.pddl
```

Testing Your Domain and Problem Files: Planner Invocation

Files can be tested using the `ff` planner, which is a binary executable included in the zip file provided on Learn.

```
./ff -o domain_file.pddl -f problem_file.pddl
```

- We invoke the planner by running `./ff` in the terminal.
- The domain file is specified with the `-o` option.
- The problem file is specified with the `-f` option.

🛵 Testing Your Domain and Problem Files: Planner Output

```
ff: parsing domain file
domain 'blocks-world-domain' found
...done
ff: parsing problem file
problem 'blocks-world-problem' found
...done

no metric specified, plan length assumed.

checking for cyclic := effects --- OK

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

Counting down from goal distance:    2 into depth [1] # active state space search
                                     1 into depth [1]
                                     0

ff: found legal plan as follows

step    0: MOVE B TABLE C # valid plan printed if found
        1: MOVE A TABLE B

time spent:    0.00 seconds instantiating 18 easy, 0 hard action templates # measures of planner performance
               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
```

🥤 Experiment (15 marks)

By default, the `ff` planner uses a fast yet incomplete heuristic called Enforced Hill Climbing to find a plan.

🥤 Experiment (15 marks)

By default, the `ff` planner uses a fast yet incomplete heuristic called Enforced Hill Climbing to find a plan.

If no plan is found, the planner falls back best-first search heuristic to find a plan.

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

🥤 Experiment (15 marks)

By default, the `ff` planner uses a fast yet incomplete heuristic called Enforced Hill Climbing to find a plan.

If no plan is found, the planner falls back best-first search heuristic to find a plan.

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

Here,

- $g(s)$ is the cost of the path from the initial state to the current state s.
- $h(s)$ is the estimated cost of the path from s to the goal state.
- w_g and w_h are the weights of the two components of the heuristic.

🥤 Experiment (15 marks)

By default, the `ff` planner uses a fast yet incomplete heuristic called Enforced Hill Climbing to find a plan.

If no plan is found, the planner falls back best-first search heuristic to find a plan.

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

Here,

- $g(s)$ is the cost of the path from the initial state to the current state s.
- $h(s)$ is the estimated cost of the path from s to the goal state.
- w_g and w_h are the weights of the two components of the heuristic.

In this experimental component, you will evaluate the effect of modifications to the weights w_g and w_h that this heuristic uses.

🥤 Experiment (15 marks)

The research question that is to be answered in this component is: *how does our choice of* w_g *and* w_h *affect planner performance?*. The tasks involved are as followed.

🥤 Experiment (15 marks)

The research question that is to be answered in this component is: *how does our choice of w_g and w_h affect planner performance?*. The tasks involved are as followed.

Design (5 marks)

- 🍛 Design a variation of the scenario that is harder for the planner.

🥤 Experiment (15 marks)

The research question that is to be answered in this component is: *how does our choice of w_g and w_h affect planner performance?*. The tasks involved are as followed.

Design (5 marks)

- 🍛 Design a variation of the scenario that is harder for the planner.

Evaluation (10 marks)

- 🍣 Evaluate and discuss the effect of modifications to the heuristic's weights on the planner's performance.

Experiment (15 marks)

The research question that is to be answered in this component is: *how does our choice of w_g and w_h affect planner performance?*. The tasks involved are as followed.

Design (5 marks)

- Design a variation of the scenario that is harder for the planner.

Evaluation (10 marks)

- Evaluate and discuss the effect of modifications to the heuristic's weights on the planner's performance.

To invoke `./ff` in this setting, we can use the command

```
./ff -E -g <w_g> -h <w_h> -o domain_file.pddl -f problem_file.pddl
```

where `-g` is the value of w_g and `-h` is the value of w_h. `-E` disables the EHC algorithm.

⛽ Extensions (50 marks)

In the extensions, we modify our domain and problem files from the basic scenario to handle real-world challenges.

⛽ Extensions (50 marks)

In the extensions, we modify our domain and problem files from the basic scenario to handle real-world challenges.

Multiple Couriers (10 marks)

- 🛵 Model multiple couriers moving around the network and making deliveries.

⛽ Extensions (50 marks)

In the extensions, we modify our domain and problem files from the basic scenario to handle real-world challenges.

Multiple Couriers (10 marks)

- 🛵 Model multiple couriers moving around the network and making deliveries.

Courier Capacity (5 marks)

- 🍕 Model a limited carrying capacity for couriers with different item weights.

⛽ Extensions (50 marks)

In the extensions, we modify our domain and problem files from the basic scenario to handle real-world challenges.

Multiple Couriers (10 marks)

🛵 Model multiple couriers moving around the network and making deliveries.

Courier Capacity (5 marks)

🍕 Model a limited carrying capacity for couriers with different item weights.

Refuelling (10 marks)

⛽ Model fuel constraints and refuelling actions.

⛽ Extensions (50 marks)

In the extensions, we modify our domain and problem files from the basic scenario to handle real-world challenges.

Multiple Couriers (10 marks)

🛵 Model multiple couriers moving around the network and making deliveries.

Courier Capacity (5 marks)

🍕 Model a limited carrying capacity for couriers with different item weights.

Refuelling (10 marks)

⛽ Model fuel constraints and refuelling actions.

Your Extension (25 marks)

🍺 Propose and implement your own (realistic) extension to the scenario!

⛽ Extensions: Multiple Couriers (10 marks)

In this first extension, we will modify our domain and problem files to allow multiple couriers to move around the network and make deliveries.

⛽ Extensions: Multiple Couriers (10 marks)

In this first extension, we will modify our domain and problem files to allow multiple couriers to move around the network and make deliveries.

This requires the use of *existential quantification* in the domain file, which allows us to specify in an action's preconditions that a predicate is true for at least one object of a given type.

⛽ Extensions: Multiple Couriers (10 marks)

In this first extension, we will modify our domain and problem files to allow multiple couriers to move around the network and make deliveries.

This requires the use of *existential quantification* in the domain file, which allows us to specify in an action's preconditions that a predicate is true for at least one object of a given type.

We must first ensure that we import existential preconditions.

```
(:requirements :adl :existential-preconditions)
```

Extensions: Multiple Couriers)

Existential preconditions are invoked using the following signature.

```
(exists (?x - type ?y - type ...)
    (condition)
)
```

Extensions: Multiple Couriers)

Existential preconditions are invoked using the following signature.

```
(exists (?x - type ?y - type ...)
    (condition)
)
```

In the classic *block world* example, we could use existential quantifiers to express that there is a block on a table.

```
(exists (?b - block ?t - table)
    (and
        (on ?b ?loc)
        (clear ?b)
    )
)
```

⛽ Extensions: Bag Capacity and Refuelling

In extensions 2 and 3, the use of *numeric fluents*, which allow objects to hold values during a plan.[1] Again, we start by importing fluents.

```
(:requirements :adl :fluents)
```

⛽ Extensions: Bag Capacity and Refuelling

In extensions 2 and 3, the use of *numeric fluents*, which allow objects to hold values during a plan.[1] Again, we start by importing fluents.

```
(:requirements :adl :fluents)
```

Fluents are declared as functions at the top of the domain file.

```
(:functions
    (<variable_name> <parameter_name> - <object_type>)
)
```

Further information is available at `https://planning.wiki/ref/pddl21/domain#numeric-fluents` (link also provided in handout)

⛽ Extensions: Bag Capacity and Refuelling

Fluents can be used in both the preconditions and effects of actions, and their values can be modified in different ways depending on the operator. For +, −, /, ∗:

```
(+ <variable_name_x> <variable_name_y>) ; addition used for example
    purposes
```

⛽ Extensions: Bag Capacity and Refuelling

Fluents can be used in both the preconditions and effects of actions, and their values can be modified in different ways depending on the operator. For +, −, /, ∗:

```
(+ <variable_name_x> <variable_name_y>) ; addition used for example
    purposes
```

Operators such as `increase`, `decrease`, and `assign` can modify the value of a variable:

```
(increase (<variable_name> <object_type>) <value>) ; increase `
    variable_name' by `value'
```

Extensions: Bag Capacity and Refuelling

Fluents can be used in both the preconditions and effects of actions, and their values can be modified in different ways depending on the operator. For $+$, $-$, $/$, $*$:

```
(+ <variable_name_x> <variable_name_y>) ; addition used for example purposes
```

Operators such as `increase`, `decrease`, and `assign` can modify the value of a variable:

```
(increase (<variable_name> <object_type>) <value>) ; increase `variable_name' by `value'
```

It is possible to use another numeric variable in place of `value`.

```
(decrease
    (<variable_name_x> <parameter_name_x>)
    (<variable_name_y> - <parameter_name_y>) ;
```

Extensions: Your Extension

Finally, for an extra challenge, the last extension enables you to motivate, design, and implement an additional realistic modification to the scenario.

Extensions: Your Extension

Finally, for an extra challenge, the last extension enables you to motivate, design, and implement an additional realistic modification to the scenario.

Note!

- Please do not attempt this extension unless the previous tasks were a breeze.
- We will be much pickier when marking this part; and 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 coursework, even catching up on sleep) before spending time on this subtask, in the interest of efficiency!

Submission

Submission is via Gradescope.

- Please include all domain and problem files, and your report (as a pdf).
- An autograder will run your PDDL files to ensure valid outputs are generated.
- Deadline: **12:00** on **27th March 2025**.

Submission

Submission is via Gradescope.

- Please include all domain and problem files, and your report (as a pdf).
- An autograder will run your PDDL files to ensure valid outputs are generated.
- Deadline: **12:00** on **27th March 2025**.

Please feel free to ask questions on Piazza.

Submission

Submission is via Gradescope.

- Please include all domain and problem files, and your report (as a pdf).
- An autograder will run your PDDL files to ensure valid outputs are generated.
- Deadline: **12:00** on **27th March 2025**.

Please feel free to ask questions on Piazza.

Good luck, and have fun!