

Razonamiento y planificación automática

Alejandro Cervantes

Sesión de Laboratorio



Definición

Entorno

- ▶ Hay un plugin bastante cómodo para Visual Studio Code. Esta opción es la más recomendada.

VSCode: <https://code.visualstudio.com/>

Extensión: SHIFT-CONTROL-X Buscar PDDL
<https://marketplace.visualstudio.com/items?itemName=jan-dolejsi.pddl>

- ▶ Podéis usar directamente [Editor.planning.domains](https://editor.planning.domains)
- ▶ Podéis usar el planificador online u otro que deseéis. Asumiremos que usáis este por simplicidad.

VSCode + PDDL

The image shows a VS Code editor with a PDDL file open. The left pane displays the PDDL code for a 4-blocks world problem. The right pane shows the 'PDDL Overview' sidebar, which provides information about AI Planning and PDDL support in VS Code.

domain.pddl

```
1  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2  ;; 4 Op-blocks world
3  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
4
5  (define (domain BLOCKS)
6    (:requirements :strips)
7    (:predicates (on ?x ?y) 1 1 1)
8    (ontable ?x) 1 1 1
9    (clear ?x) 3 3 3
10   (handempty) 2 2 2
11   (holding ?x) 2 2 2
12 )
13
14 (:action pick-up
15   :parameters (?x)
16   :precondition (and (clear ?x) (ontable ?x) (
17   :effect
18   (and (not (ontable ?x))
19     (not (clear ?x))
20     (not (handempty))
21     (holding ?x)))
```

PDDL Overview

AI Planning and PDDL support in VS Code

Getting started

Try [Hello World](#) example
Generate [Nunjucks templated](#) problem file sample
[See or clone PDDL samples](#)

Configuration

Planning engine {} +

- ☒ <http://solver.planning.domains/solve>
- ☐ Planning as a service (preview)

Read [more info about PDDL planners](#)

Output into ☒ Output window ☐ Terminal ☐ Search debugger

PDDL parser

[c:\Users\Alex\AppData\Roaming\Code\User\g](#)

See [more info about PDDL parsers](#)

Plan Validator

[c:\Users\Alex\AppData\Roaming\Code\User\g](#)

Clone and compile [VAL from GitHub](#) or...
[Download](#) plan validation tools

☒ Show this overview page once a day when using PDDL

Resources

- YouTube [Modeling in PDDL channel](#)
- YouTube [PDDL Tooling channel](#)
- [Education.planning.domains](#)
- Explore [Planning.domains](#) PDDL examples
- [Ask a question on Stackoverflow](#)
- [PDDL Reference](#)
- [Slack community](#)
- [All features of PDDL support in VS Code](#)
- [What's new in PDDL support](#)

Getting more productive

- [VS Code Icons for PDDL files](#) e.g.
- [GraphViz support](#)
- [Keyboard shortcuts](#)

Giving feedback

- [Submit an issue](#)
- [Write a review](#)

VSCoDe + PDDL

The screenshot shows the VS Code editor with three panels. The left panel displays `domain.pddl`, the middle panel displays `probBLOCKS-10-0.pddl`, and the right panel displays the `Planner output`. The bottom status bar shows the `Planner output` tab is active.

```
1  ::::::::::::::::::::::::::::::::::::::
2  ;; 4 Op-blocks world
3  ::::::::::::::::::::::::::::::::::::::
4
5  (define (domain BLOCKS)
6    (:requirements :strips)
7    (:predicates (on ?x ?y) 1 1 1)
8    (ontable ?x) 1 1 1
9    (clear ?x) 3 3 3
10   (handempty) 2 2 2
11   (holding ?x) 2 2 2
12 )
13
14 (:action pick-up
15   :parameters (?x)
16   :precondition (and (clear ?x)
17                     (not (ontable ?x))
18                     (not (clear ?x))
19                     (not (handempty))
20                     (holding ?x)))
21   :effect
22   (and (not (ontable ?x))
23        (not (clear ?x))
24        (not (handempty))
25        (holding ?x)))
26 )
27
28 (:action put-down
29   :parameters (?x)
30   :precondition (and (ontable ?x)
31                     (not (clear ?x))
32                     (not (handempty))
33                     (holding ?x)))
34   :effect
35   (and (ontable ?x)
36        (clear ?x)
37        (handempty)
38        (not (holding ?x)))
39 )
40 )
```

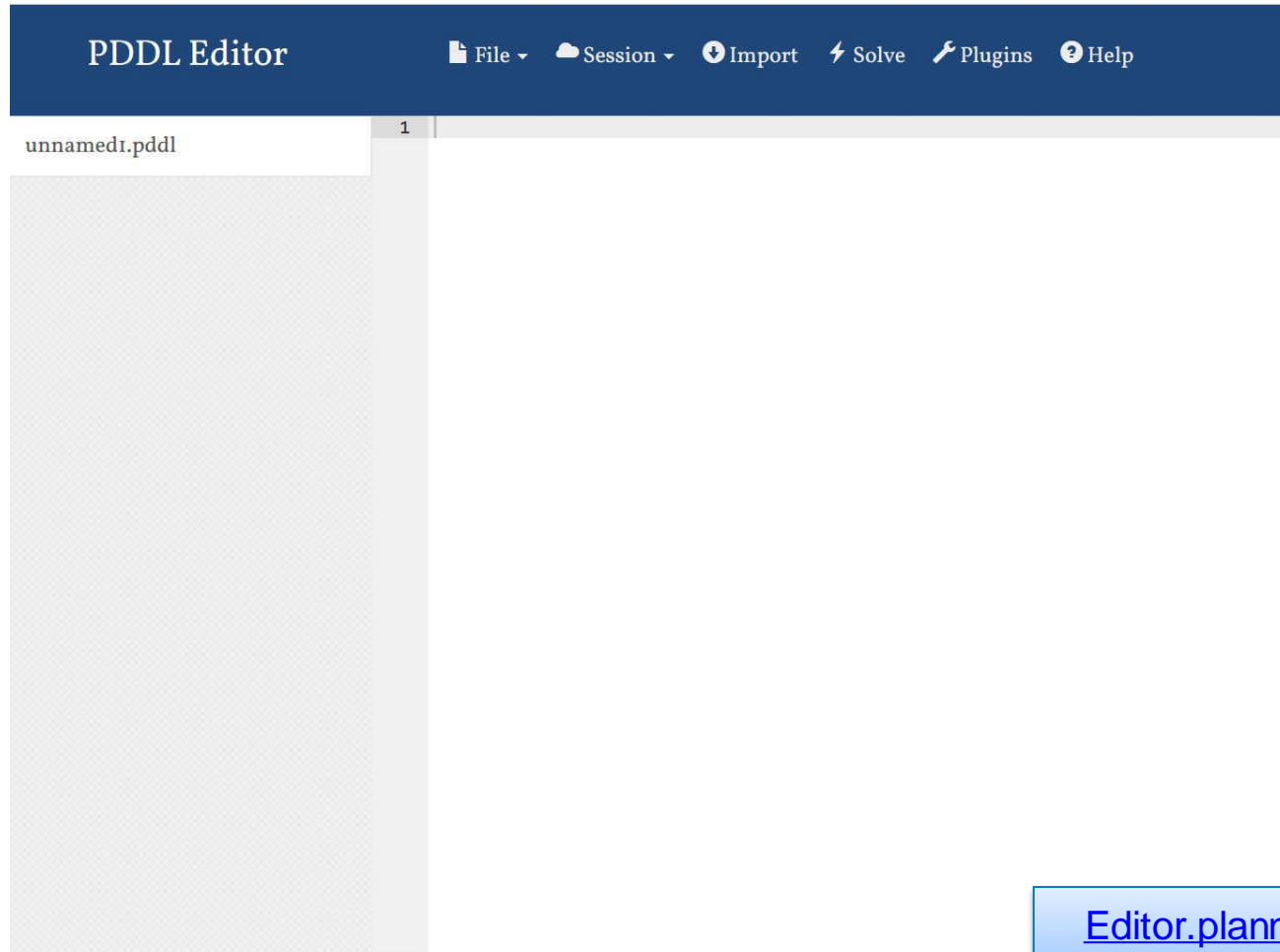
```
1  (define (problem BLOCKS-10-0)
2    (:domain BLOCKS)
3    Show hierarchy
4    (:objects D A H G B J E I F C)
5    View
6    (:INIT (CLEAR C) (CLEAR F) (ONTABLE C)
7           (ON B G) (ON G H) (ON H A) (ON A G)
8           (:goal (AND (ON D C) (ON C F) (ON A G) (ON G I))))
9  )
```

```
1  unstack C E
2  stack C F
3  unstack E J
4  put-down E
5  unstack J B
6  stack J E
7  unstack B G
8  put-down B
9  unstack G H
10 put-down G
11 unstack H A
12 stack H B
13 unstack A D
14 put-down A
15 unstack D I
16 stack D C
17 pick-up G
18 stack G I
19 pick-up A
20 stack A G
21 unstack H A
22 unstack G H
23 unstack J B
24 unstack E J
25 stack C F
26 unstack C E
```

```
#RP_fluents 0Plan found with cost: 42
Total time: 0.028
Nodes generated during search: 1026
Nodes expanded during search: 783
IW search completed

Plan found:
0.00100: (unstack c e)
0.00200: (stack c f)
0.00300: (unstack e j)
0.00400: (put-down e)
```

Editor en línea



Editor.planning.domains

Objetivos

No tiene por qué encontrar el plan óptimo

Dominio: El mismo para toda la actividad (hecho por el estudiante)

Parte 1: Caso base

- Se indica la distribución concreta de contenedores, fábricas y depósito destino

Parte 2: Modificaciones

- Se trata de hacer modificaciones interesantes del problema del caso base, manteniendo el dominio

Opciones: Varios planificadores (puntuación extra)

- Se puede instalar alguno de los planificadores mencionados y comparar resultados y posibilidades de cada uno. Aquí cabría usar variaciones del dominio (optimización, costes, números).

Operadores

- **Mover** a una localización **directamente conectada** sobre la vía (es decir, de A a P, o de F1 a A, etc.). **Puede moverse en direcciones opuestas en acciones sucesivas** (es decir, puede darse la vuelta ya que tiene máquina en ambos extremos).
- **Cargar** un contenedor en cualquier localización donde se encuentre, **siempre y cuando no se exceda el espacio disponible**.
- **Descargar** un contenedor en cualquier localización donde se encuentre, a excepción del puerto. Recuerde que al descargar no se puede exceder la capacidad de la localización.
- **Procesar** los contenedores que están en las fábricas. Nota: el enunciado hace referencia a que el tren en este caso está parado.
- Simplificación: Es válido que se procesen los contenedores en el instante de descargar.



PDDL

Definición del dominio

- ▶ El dominio es la información sobre el mundo en general
- ▶ En el dominio no aparecen proposiciones instanciadas

Ejemplo:

```
(define (domain DOMAIN_NAME)

  (:requirements [:strips] [:equality] [:typing] [:adl])

  (:predicates
    (PREDICATE_1_NAME [?A1 ?A2 ... ?AN])
    (PREDICATE_2_NAME [?A1 ?A2 ... ?AN]) ...)

  (:action ACTION_1_NAME
    [:parameters (?P1 ?P2 ... ?PN)]
    [:precondition PRECOND_FORMULA]
    [:effect EFFECT_FORMULA] )

  (:action ACTION_2_NAME ...)

  ...)
```

```
(:action LOAD-TRUCK
  :parameters (?obj ?truck ?loc)
  :precondition
    (and (OBJ ?obj) (TRUCK ?truck) (LOCATION ?loc)
      (at ?truck ?loc) (at ?obj ?loc))
  :effect
    (and (not (at ?obj ?loc)) (in ?obj ?truck)))
```

!! En PDDL agrupamos los effects y los deletes de STRIPS en :effect

Notación prefija para funciones: (and ?x ?y) (ver LISP)

Definición del problema

- El problema es una **instancia** del mundo

```
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects OBJ1 OBJ2 ... OBJ_N)
  (:init ATOM1 ATOM2 ... ATOM_N)
  (:goal CONDITION_FORMULA) )
```

- En el problema de planificación aparecerán instanciadas la mayoría de las proposiciones. Contiene el estado inicial y meta
- Usamos elementos **atómicos** para formar **literales** que pueden ser **afirmativos** o **negativos**.
- Los **fluents** son átomos instanciados con objetos del estado del mundo.
- Al igual que en la descripción de STRIPS, la hipótesis del mundo cerrado hace que **los fluents que no sean nombrados** explícitamente en una descripción sean considerados como **falsos**.

Mundo de los bloques

- Ficheros: competición 2000, id: 112

ID	Domain	Tags	Description
112	blocks	["adl","strips","conditional-effects","action-costs","derived-predicates","equality","negative-preconditions","typing"]	<p>The blocks world is one of the most famous planning domains in artificial intelligence. Imagine a set of cubes (blocks) sitting on a table. The goal is to build one or more vertical stacks of blocks. The catch is that only one block may be moved at a time: it may either be placed on the table or placed atop another block. Because of this, any blocks that are, at a given time, under another block cannot be moved.</p>

Mundo de los bloques

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;
;;; 4 Op-blocks world
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;

(define (domain BLOCKS)
  (:requirements :strips)
  (:predicates (on ?x ?y)
    (ontable ?x)
    (clear ?x)
    (handempty)
    (holding ?x)
  )

  (:action pick-up
    :parameters (?x)
    :precondition (and
      (clear ?x) (ontable ?x)
      (handempty))
    :effect
      (and (not (ontable ?x))
        (not (clear ?x))
        (not (handempty))
        (holding ?x)))

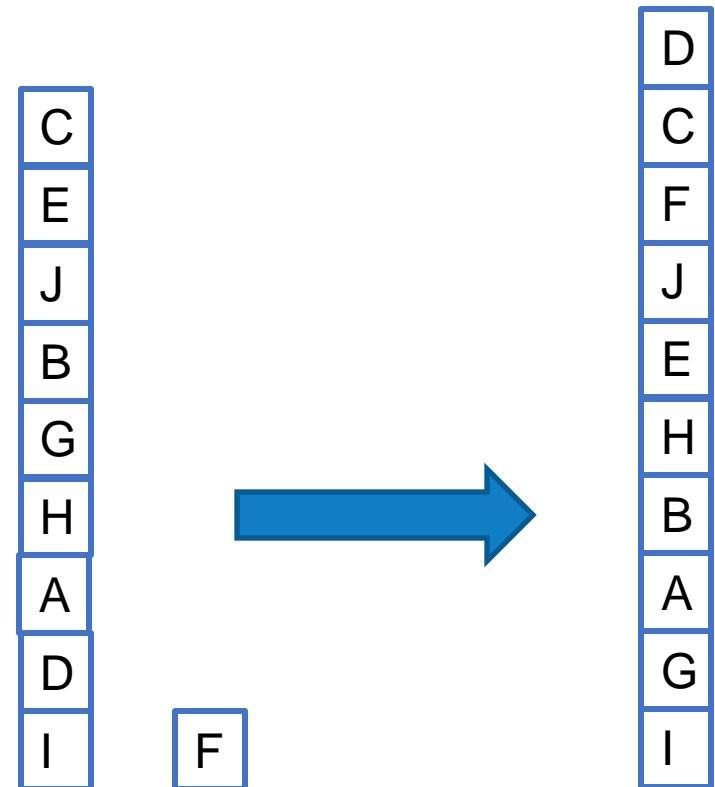
  (:action put-down
    :parameters (?x)
    :precondition (holding ?x)
    :effect
      (and (not (holding ?x))
        (clear ?x)
        (handempty)
        (ontable ?x)))

  (: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)))

  (:action unstack
    :parameters (?x ?y)
    :precondition (and (on ?x ?y) (clear ?x)
      (handempty))
    :effect
      (and (holding ?x)
        (clear ?y)
        (not (clear ?x))
        (not (handempty))
        (not (on ?x ?y)))))
```

Mundo de los bloques

```
(define (problem BLOCKS-10-0)
  (:domain BLOCKS)
  (:objects D A H G B J E I F C )
  (:INIT (CLEAR C) (CLEAR F) (ONTABLE I) (ONTABLE F)
    (ON C E) (ON E J) (ON J B)
    (ON B G) (ON G H) (ON H A) (ON A D) (ON D I)
    (HANDEEMPTY))
  (:goal (AND (ON D C) (ON C F) (ON F J) (ON J E) (ON
    E H) (ON H B) (ON B A)
    (ON A G) (ON G I)))
)
```



Mundo de los bloques

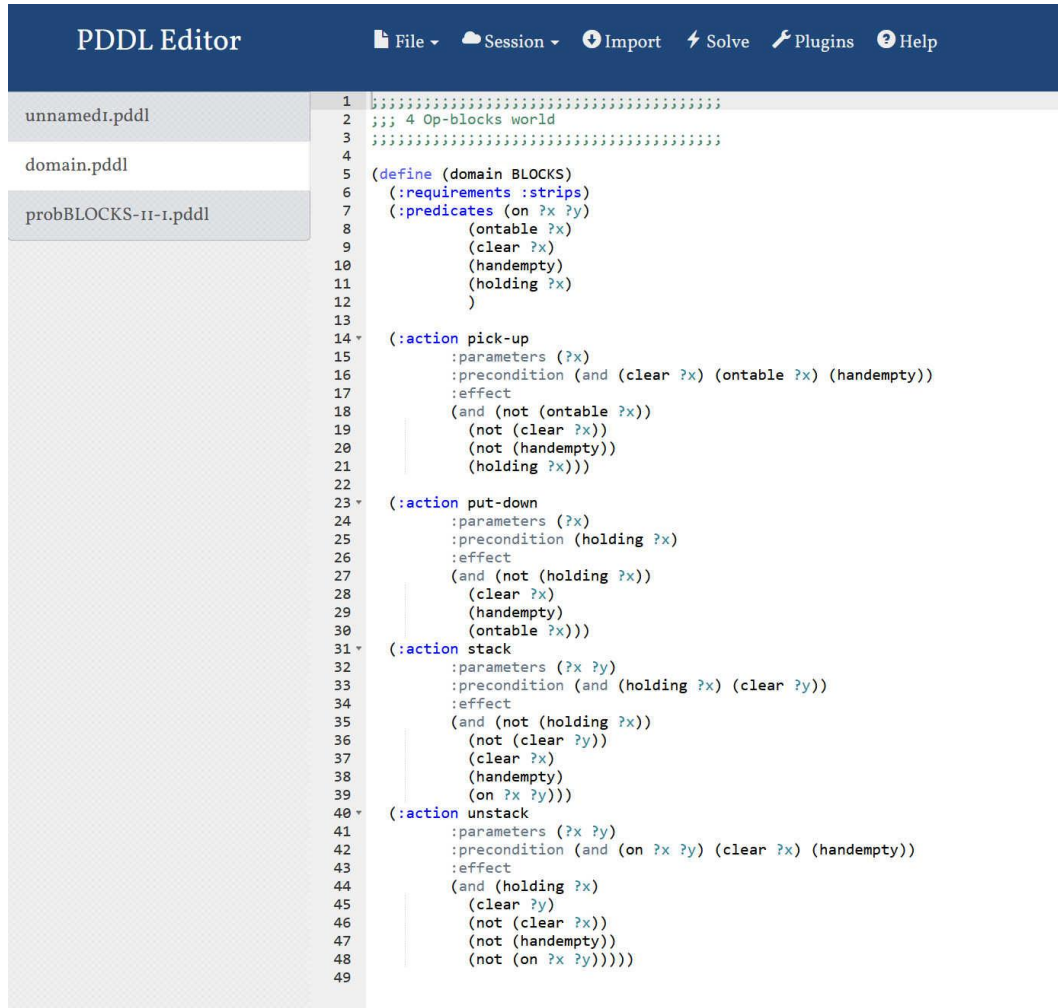
```
Plan found:
0.00100: (unstack c e)
0.00200: (stack c f)
0.00300: (unstack e j)
0.00400: (put-down e)
0.00500: (unstack j b)
0.00600: (stack j e)
0.00700: (unstack b g)
0.00800: (put-down b)
0.00900: (unstack g h)
0.01000: (put-down g)
0.01100: (unstack h a)
0.01200: (stack h b)
0.01300: (unstack a d)
0.01400: (put-down a)
0.01500: (unstack d i)
0.01600: (stack d c)
0.01700: (pick-up g)
0.01800: (stack g i)
0.01900: (pick-up a)
0.02000: (stack a g)
0.02100: (unstack h b)
0.02200: (put-down h)
0.02300: (pick-up b)
0.02400: (stack b a)
```

```
0.02500: (pick-up h)
0.02600: (stack h b)
0.02700: (unstack j e)
0.02800: (put-down j)
0.02900: (pick-up e)
0.03000: (stack e h)
0.03100: (pick-up j)
0.03200: (stack j e)
0.03300: (unstack d c)
0.03400: (put-down d)
0.03500: (unstack c f)
0.03600: (put-down c)
0.03700: (pick-up f)
0.03800: (stack f j)
0.03900: (pick-up c)
0.04000: (stack c f)
0.04100: (pick-up d)
0.04200: (stack d c)
Metric: 0.042
Makespan: 0.042
States evaluated: undefined
Planner found 1 plan(s) in
0.434secs.
```



Ejemplo de uso

Importación de Dominio

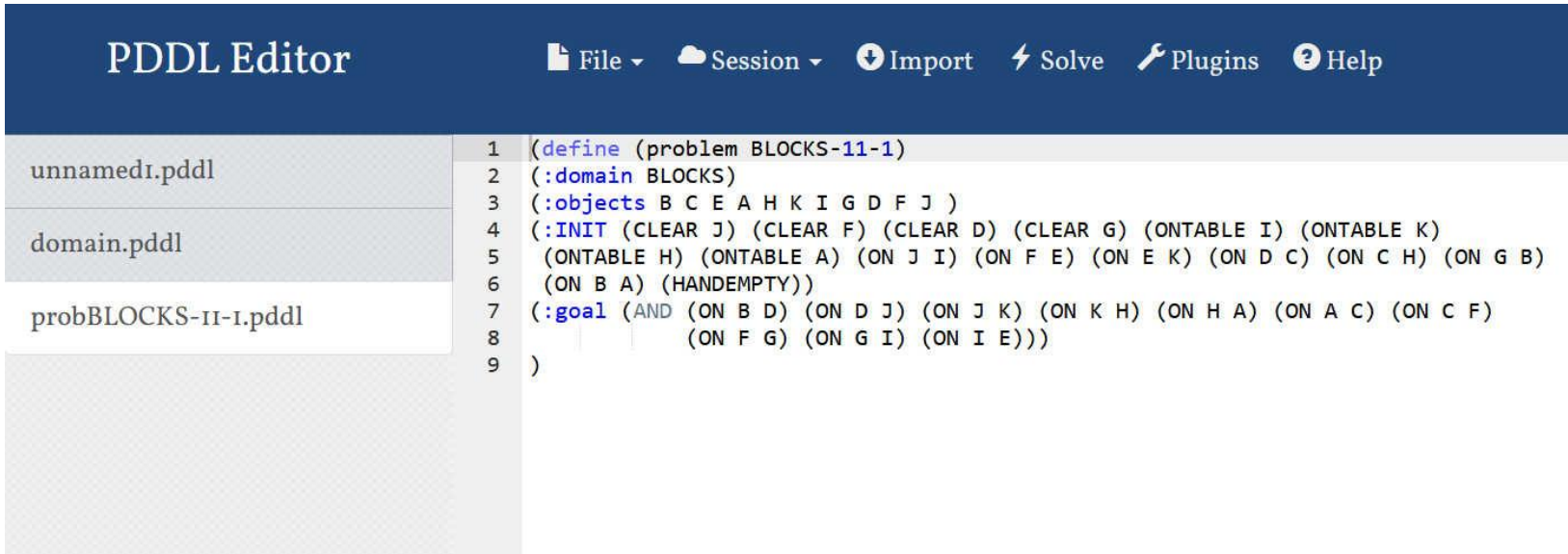


The screenshot shows the PDDL Editor interface. The left sidebar lists three files: 'unnamed1.pddl', 'domain.pddl', and 'probBLOCKS-11-1.pddl'. The 'domain.pddl' file is selected and its content is displayed in the main editor area. The code defines a domain named 'BLOCKS' with the following structure:

```
1 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2 ;; 4 Op-blocks world
3 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
4
5 (define (domain BLOCKS)
6   (:requirements :strips)
7   (:predicates (on ?x ?y)
8     (ontable ?x)
9     (clear ?x)
10    (handempty)
11    (holding ?x)
12  )
13
14   (:action pick-up
15     :parameters (?x)
16     :precondition (and (clear ?x) (ontable ?x) (handempty))
17     :effect
18     (and (not (ontable ?x))
19          (not (clear ?x))
20          (not (handempty))
21          (holding ?x)))
22
23   (:action put-down
24     :parameters (?x)
25     :precondition (holding ?x)
26     :effect
27     (and (not (holding ?x))
28          (clear ?x)
29          (handempty)
30          (ontable ?x)))
31
32   (:action stack
33     :parameters (?x ?y)
34     :precondition (and (holding ?x) (clear ?y))
35     :effect
36     (and (not (holding ?x))
37          (not (clear ?y))
38          (clear ?x)
39          (handempty)
40          (on ?x ?y)))
41
42   (:action unstack
43     :parameters (?x ?y)
44     :precondition (and (on ?x ?y) (clear ?x) (handempty))
45     :effect
46     (and (holding ?x)
47          (clear ?y)
48          (not (clear ?x))
49          (not (handempty))
50          (not (on ?x ?y))))
```

[Editor.planning.domains](#)

Importación Problema



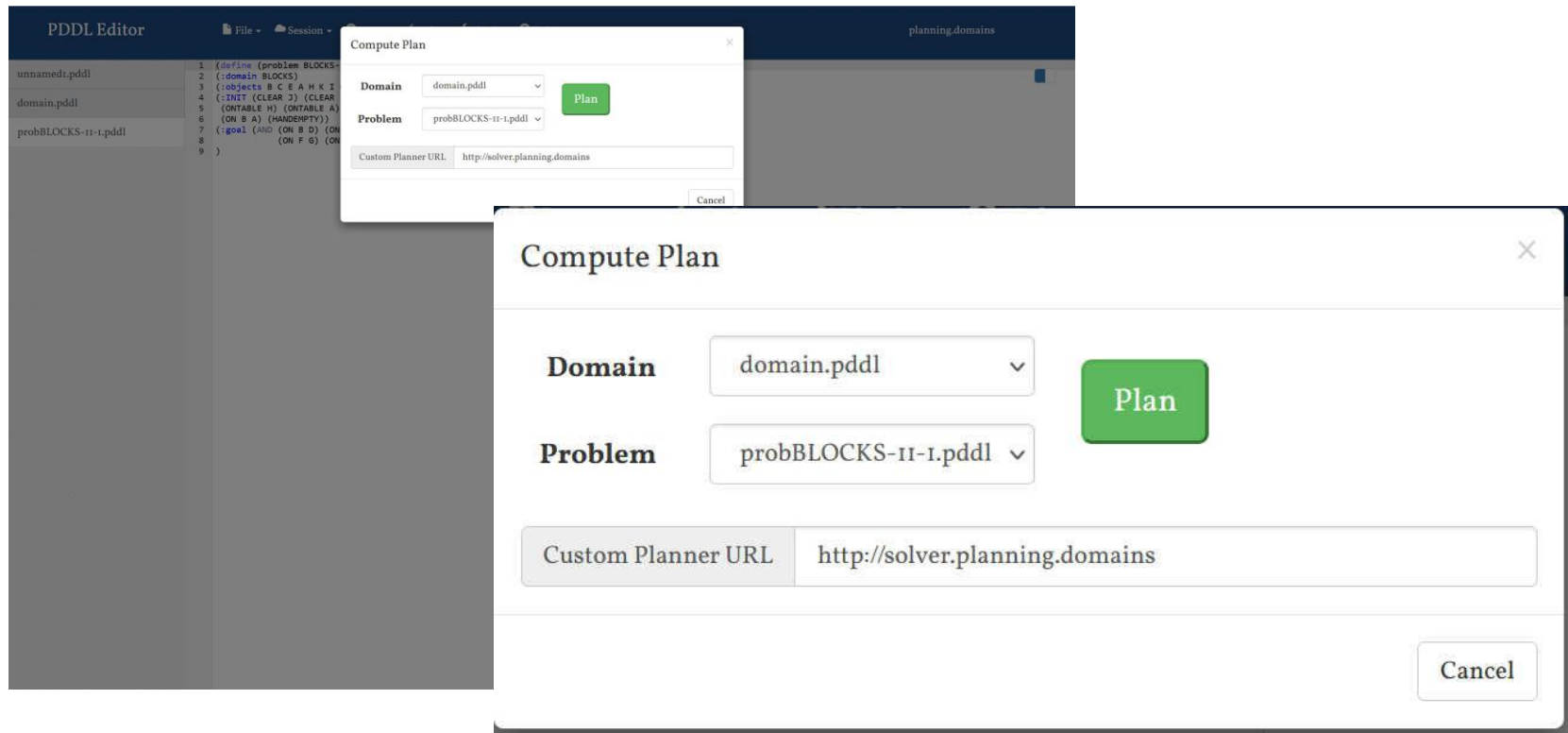
The screenshot shows the PDDL Editor application. The title bar reads "PDDL Editor". The menu bar includes "File", "Session", "Import", "Solve", "Plugins", and "Help". The left sidebar shows a file list with three items: "unnamed1.pddl", "domain.pddl", and "probBLOCKS-II-1.pddl". The main editor area displays the following PDDL code:

```
1 (define (problem BLOCKS-II-1)
2   (:domain BLOCKS)
3   (:objects B C E A H K I G D F J )
4   (:INIT (CLEAR J) (CLEAR F) (CLEAR D) (CLEAR G) (ONTABLE I) (ONTABLE K)
5     (ONTABLE H) (ONTABLE A) (ON J I) (ON F E) (ON E K) (ON D C) (ON C H) (ON G B)
6     (ON B A) (HANDEEMPTY))
7   (:goal (AND (ON B D) (ON D J) (ON J K) (ON K H) (ON H A) (ON A C) (ON C F)
8     (ON F G) (ON G I) (ON I E)))
9 )
```

¿Podemos dibujar el estado inicial y el (un) objetivo?

[Editor.planning.domains](#)

Ejecutando el Solver (Planificador)



[Editor.planning.domains](http://editor.planning.domains)

Consultando el plan

The screenshot shows the PDDL Editor interface. On the left, a file list contains 'unnamed1.pddl', 'domain.pddl', 'probBLOCKS-II-I.pddl', and 'Plan (1)'. The main area is titled 'Found Plan (output)' and displays a list of actions on the left and their corresponding PDDL code on the right. The actions listed are: (unstack g b), (put-down g), (unstack f e), (stack f g), (unstack e k), (put-down e), (unstack d c), (stack d k), (unstack j i), (put-down j), (unstack c h), (stack c f), (unstack b a), (stack b d), (pick-up h), and (stack b a). The PDDL code for the first action is shown as an example:

```
(:action unstack
:parameters (g b)
:precondition
  (and
    (on g b)
    (clear g)
    (handempty)
  )
:effect
  (and
    (holding g)
    (clear b)
    (not
      (clear g)
    )
    (not
      (handempty)
    )
    (not
      (on g b)
    )
  )
)
```

Recomiendo
plugins:

- Torchlight
- Misc PDDL
Generators

Editor.planning.domains

Más información

- <http://education.planning.domains/>
- <https://planning.wiki/>

► Diferencias entre versiones (strips/adl/etc.):

<https://users.cecs.anu.edu.au/~patrik/pddlman/writing.html>

Ejemplos clase: requirements (PDDL 1.2, sin valores numéricos):

```
(:requirements :strips :typing :equality :negative-  
preconditions)
```

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

Ojo, PDDL 2.0+, fluents numéricos **no soportados por el solver de planning.online**

Otros planificadores

- ▶ Planificador avanzado y muchos ejemplos:

<https://fai.cs.uni-saarland.de/hoffmann/ff.html>



Ejercicio de clase

Ejemplo de clase: dominio rutas

Crear el dominio y un ejemplo de problema para un vehículo que puede moverse entre distintos puntos, y vamos ampliando.

Sugerencias para continuar:

- Incorporar tipos
- Recoger personas en el coche
- Incluir algún tipo de numeración (ejemplo capacidad del coche o capacidad al descargar)
- Analizar el dominio transport (2008) y/o storage (2006) y usarlo como referencia para vuestra actividad

Ejemplo de clase: dominio rutas

Crear el dominio y un ejemplo de problema para un vehículo que puede moverse entre distintos puntos, y vamos ampliando luego para que cargue un objeto.

Detectar el error en los ficheros incorrectos y corregirlo

Recomendaciones (IMPORTANTE)

- No le decimos al planificador qué es lo que hay que hacer ni en qué orden. Queremos que encuentre el plan por sí mismo.

Por ejemplo no le forzamos a hacer ciertas acciones antes que otras (ojo, esto es en cierto modo lo que se hace en Tema9). Podéis analizar en la entrega si esto sería útil en algún caso, pero ojo, sin perder generalidad.

- Es más cómodo utilizar **tipos** en lugar de predicados adicionales. Esto permite dibujar la jerarquía y sobre todo reducir el número de flujos.
- Si no se usan tipos, siempre "filtrar" usando esos predicados de "clase" (es_ciudad, es_cargable) para evitar que el planificador genere instancias de los operadores que son inválidas (mover un contenedor, cargar una fábrica, etc.)
- Si se agota la memoria, probablemente se están generando demasiadas ramas; hay que añadir restricciones para reducir el factor de ramificación (quizá las anteriores).
- Si no se encuentra la solución puede que los operadores tengan restricciones incorrectas (o que supere el límite de tiempo impuesto). Quitar restricciones y ver si así funciona. Luego ir añadiéndolas una a una.

Tutorial (incluye tipos)

<https://fareskalaboud.github.io/LearnPDDL/>



www.unir.net