Razonamiento y Planificación Automática

César Augusto Guzmán Álvarez

Doctor en Inteligencia Artificial

Tema 9 : Sistemas basados en STRIPS

Sesión 1/2

LA UNIVERSIDAD EN INTERNET

Universidad Internacional de La Rioja

Resumen – Tema anterior

Tema 8 : Problemas de planificación

- Qué es un problema de planificación?
- Aproximaciones de planificación
- Practica : Instalar Visual Studio Code con soporte para PDDL

Índice

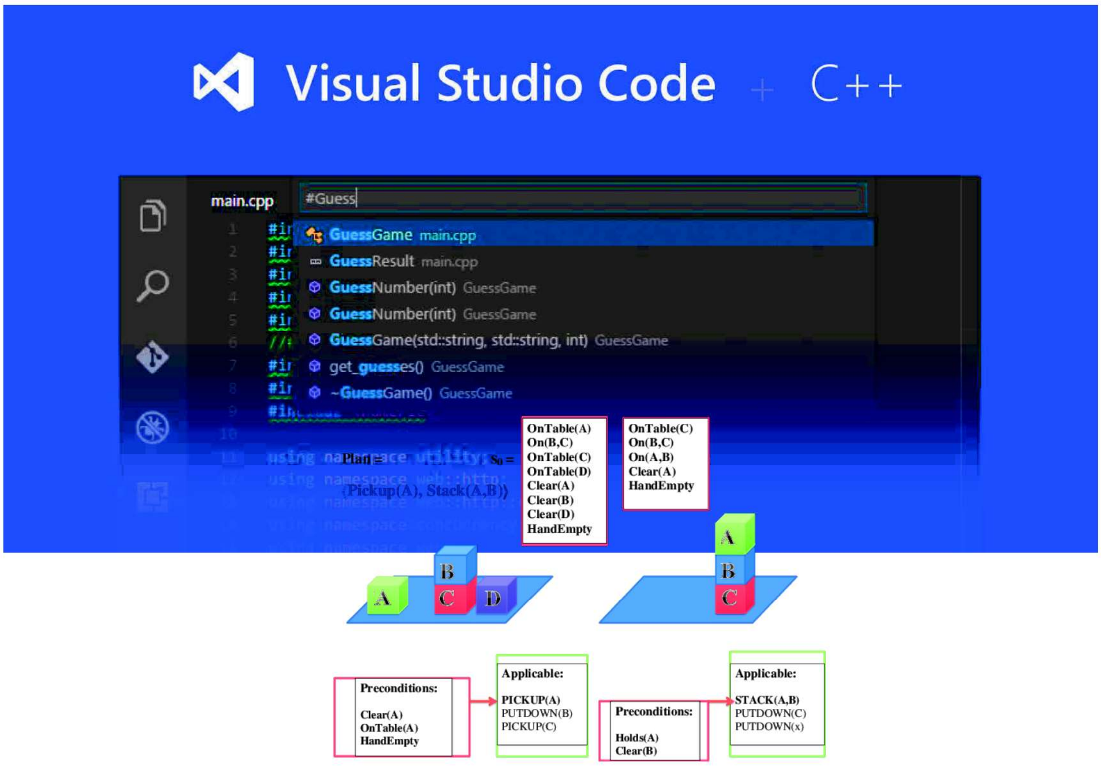
Sesión 1:

- Practica STRIPS / PDDL Windows

Sesión 2 :

- Practica STRIPS / PDDL Linux

Practica STRIPS / PDDL Windows



Plan = (Pickup(A), Stack(A,B))

S_0 = OnTable(A) On(B,C) OnTable(C) OnTable(D) Clear(A) Clear(B) Clear(D) HandEmpty

OnTable(C) On(B,C) On(A,B) Clear(A) HandEmpty

Preconditions: Clear(A) OnTable(A) HandEmpty

Applicable: PICKUP(A) PUTDOWN(B) PICKUP(C)

Preconditions: Holds(A) Clear(B)

Applicable: STACK(A,B) PUTDOWN(C) PUTDOWN(x)

Fuentes de imágenes:
https://code.visualstudio.com/assets/docs/languages/cpp/languages_cpp.png
Jobczyk, Krystian & Ligęza, Antoni. (2017). STRIPS in Some Temporal-Preferential Extension. 10.1007/978-3-319-59063-9_22.

PDDL STRIPS : Definir un dominio y problema

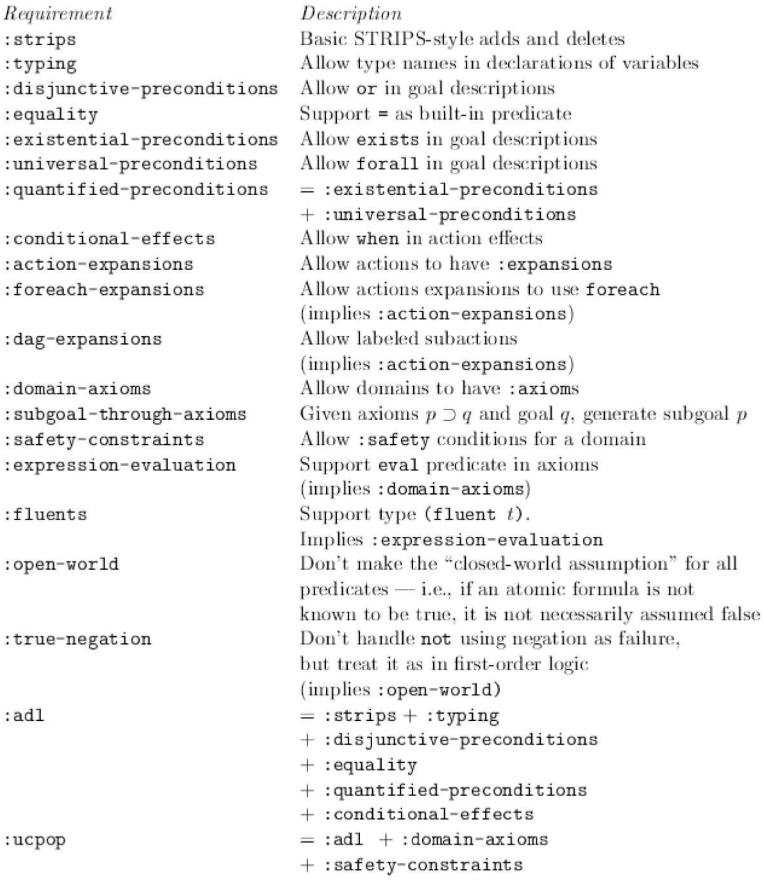
dominio

```
(define (domain hello)

(:requirements … )

(:types … )

(:predicates … )

(:action … )
)
```

problema

```
(define (problem hello-world)

(:domain hello)

(:objects … )

(:init … )

(:goal
    (and
        ...
    )
)
)
```

PDDL STRIPS : Elementos del dominio

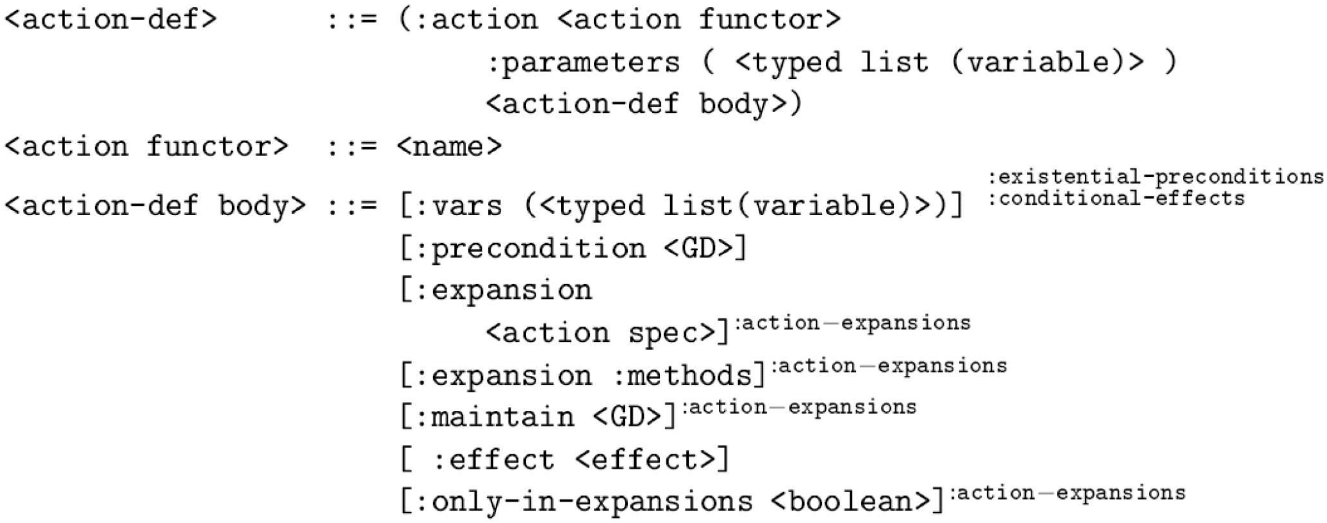


Requirement	*Description*
`:strips`	Basic STRIPS-style adds and deletes
`:typing`	Allow type names in declarations of variables
`:disjunctive-preconditions`	Allow `or` in goal descriptions
`:equality`	Support `=` as built-in predicate
`:existential-preconditions`	Allow `exists` in goal descriptions
`:universal-preconditions`	Allow `forall` in goal descriptions
`:quantified-preconditions`	= `:existential-preconditions` + `:universal-preconditions`
`:conditional-effects`	Allow `when` in action effects
`:action-expansions`	Allow actions to have `:expansions`
`:foreach-expansions`	Allow actions expansions to use `foreach` (implies `:action-expansions`)
`:dag-expansions`	Allow labeled subactions (implies `:action-expansions`)
`:domain-axioms`	Allow domains to have `:axioms`
`:subgoal-through-axioms`	Given axioms $p \supset q$ and goal q, generate subgoal p
`:safety-constraints`	Allow `:safety` conditions for a domain
`:expression-evaluation`	Support `eval` predicate in axioms (implies `:domain-axioms`)
`:fluents`	Support type `(fluent` t`)`. Implies `:expression-evaluation`
`:open-world`	Don't make the "closed-world assumption" for all predicates — i.e., if an atomic formula is not known to be true, it is not necessarily assumed false
`:true-negation`	Don't handle `not` using negation as failure, but treat it as in first-order logic (implies `:open-world`)
`:adl`	= `:strips` + `:typing` + `:disjunctive-preconditions` + `:equality` + `:quantified-preconditions` + `:conditional-effects`
`:ucpop`	= `:adl` + `:domain-axioms` + `:safety-constraints`

Fuente: Ghallab, Malik & Knoblock, Craig & Wilkins, David & Barrett, Anthony & Christianson, Dave & Friedman, Marc & Kwok, Chung & Golden, Keith & Penberthy, Scott & Smith, David & Sun, Ying & Weld, Daniel. (1998). PDDL - The Planning Domain Definition Language.

PDDL STRIPS : Elementos del dominio

Actions:

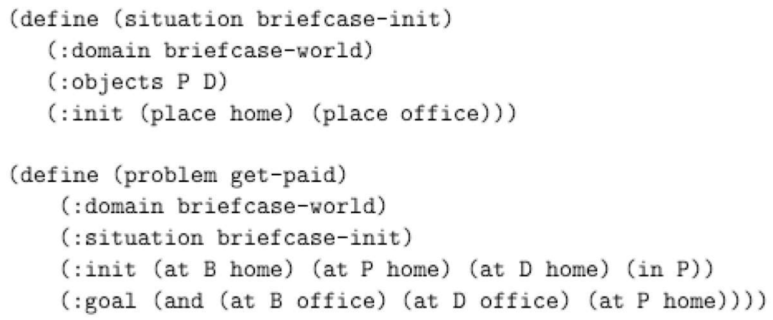


```
<action-def>        ::= (:action <action functor>
                              :parameters ( <typed list (variable)> )
                              <action-def body>)
<action functor>    ::= <name>
                                                                :existential-preconditions
<action-def body> ::= [:vars (<typed list(variable)>)]  :conditional-effects
                         [:precondition <GD>]
                         [:expansion
                              <action spec>]:action−expansions
                         [:expansion :methods]:action−expansions
                         [:maintain <GD>]:action−expansions
                         [ :effect <effect>]
                         [:only-in-expansions <boolean>]:action−expansions
```

Fuente: Ghallab, Malik & Knoblock, Craig & Wilkins, David & Barrett, Anthony & Christianson, Dave & Friedman, Marc & Kwok, Chung & Golden, Keith & Penberthy, Scott & Smith, David & Sun, Ying & Weld, Daniel. (1998). PDDL - The Planning Domain Definition Language.

PDDL STRIPS : Elementos del problema

Ejemplo de uso:



```
(define (situation briefcase-init)
   (:domain briefcase-world)
   (:objects P D)
   (:init (place home) (place office)))

(define (problem get-paid)
    (:domain briefcase-world)
    (:situation briefcase-init)
    (:init (at B home) (at P home) (at D home) (in P))
    (:goal (and (at B office) (at D office) (at P home))))
```

Fuente: Ghallab, Malik & Knoblock, Craig & Wilkins, David & Barrett, Anthony & Christianson, Dave & Friedman, Marc & Kwok, Chung & Golden, Keith & Penberthy, Scott & Smith, David & Sun, Ying & Weld, Daniel. (1998). PDDL - The Planning Domain Definition Language.

Puzzle-8

índice fila y columna	1	2	3
1	2	8	3
2	1	6	4
3	7	0	5

Estado Inicial

Objetos:

- Pieza y posición

Predicados:

- Localización de la pieza
- Identificación de la pieza blanca
- Indicar movimientos

Operadores:

- Mover-abajo
- Mover-arriba
- Mover-izquierda
- Mover-derecha

Puzzle-8

índice fila y columna	1	2	3
1	1	2	3
2	8	0	4
3	7	6	5

Estado Inicial

Indicar movimiento es valido
Identificación pieza blanca
Localización pieza

Mover-abajo

Negamos Identificación pieza blanca
Negamos Localización pieza
Nueva localización pieza
Nueva identificación pieza blanca

(inc ?filaActual ?filaDestino)
(blank ?filaDestino ?colActual)
(localizada ?pieza ?filaActual ?colActual)

Mover-abajo

(not (blank ?filaDestino ?colActual))
(not (localizada ?pieza ?filaActual ?colActual))
(blank ?filaActual ?colActual)
(localizada ?pieza ?filaDestino ?colActual))

Puzzle-8

índice fila y columna	1	2	3
1	1	2	3
2	8	0	4
3	7	6	5

Estado Final

Amazon

índice fila y columna	1	2	3	4
1	M1	#		M3
2		#		
3	M2		R	
4				

Estado Inicial

Objetos:

- Inventario
- Posición
- Robot

Predicados:

- Localización del robot
- Localización del inventario
- Caminos validos
- Tiene inventario

Operadores:

- Mover
- Cargar
- Descargar

Amazon

fila / columna	1	2	3	4
1		#		
2		#		
3				
4		M3	M2	M1

Estado Final

Getting Started – Visual Studio Code

1. open a blank folder in VS Code using *File > Open Folder...*,
2. create two blank files using *File > New File* named domain.pddl and problem.pddl, both files will show up in the *Explorer* pane, open them side by side in the editor,
3. open the *domain.pddl* file and type domain. The auto-completion suggests to insert the entire structure of the domain file. Use the Tab and Enter keys to skip through the placeholders and make your selections.
4. open the *problem.pddl* file and type problem. The auto-completion suggests to insert the entire structure of the problem file. Make sure that the (domain name) here matches the name selected in the domain file.
5. When prompted to install the VAL (i.e. Validator) tools, follow the instructions. This will bring a PDDL parser and plan validation utilities to your experience.
6. When you are ready to run the planner on your domain and problem files (both must be open in the editor), invoke the planner via context menu on one of the file text content, or via the Alt + P shortcut.
   The planning.domains solver will be used, so do not send any confidential PDDL code.
7. Configure your own PDDL planner by following instructions.

Gracias!