Universidad Carlos III de Madrid
Escuela Politécnica Superior

Manual de usuario PLTool

Leganés
Noviembre de 2007

Índice

1. Introducción

El presente documento describe el funcionamiento de la herramienta PLTool, una herramienta de iniciativa mixta para la planificación automática de tareas desarrollado por el Grupo de Planificación y Aprendizaje (PLG) de la Universidad Carlos III de Madrid. Esta herramienta corresponde a un planificador que es lanzado desde CLISP. En la actualidad, existen varios planificadores que se pueden utilizar mediante comandos por consola. Los planificadores a los que nos referimos concretamente son: PRODIGY, SAYPHI, LPG, SGLPAN y METRIC-FF. La interfaz desarrollada, nos ofrece un soporte gráfico a estos planificadores con el objetivo de hacer su utilización más sencilla e intuitiva.

La herramienta ofrecerá la posibilidad de trabajar con HAMLET, un sistema de aprendizaje de reglas para Prodigy 4.0 (ahora IPSS).

Como añadido a la nueva versión de la herramienta aparece la utilidad "Experimenter". Esta utilidad nos permitirá lanzar experimentos que pueden estar formados por varios dominios y problemas que se podrán aplicar a un conjunto de planificadores, con el fin de obtener resultados comparativos de cómo se comportan los distintos planificadores.

2. Instalación de la herramienta

En este apartado se hablará de los requisitos mínimos para la utilización de la herramienta, así como los pasos a realizar para la instalación y ejecución de la misma.

2.1. Requisitos

En este apartado se describen los requisitos necesarios para que la herramienta funcione correctamente:

- Sistema operativo LINUX.
- Compilador gcc versión 4.0 y superiores. Para poder ejecutar y compilar los Planificadores LPG, METRIC-FF y SGPLAN.
- Versión de Common LISP (CLISP) 2.39 y superiores. Para poder compilar y ejecutar los Planificadores PRODIGY y SAYPHI.
- Versión de la librería gtk 2.0 y superiores. Necesaria para la interfaz gráfica de la herramienta.

2.2. Instalación

Para instalar la herramienta deberemos seguir los siguientes pasos:

1. Crear el directorio donde se instalará la aplicación. Por ejemplo,

   ```
   % mkdir pltool
   ```

2. Dentro de este directorio almacenaremos los ficheros *pltool.tgz* e *install.sh*.

3. Añadiremos permisos de ejecución al fichero *install.sh*.

   ```
   % chmod +x install.sh
   ```

4. Ejecutaremos el script anterior que instalará la herramienta.

   ```
   % ./install.sh
   ```

2.3. Ejecución

Una vez instalado *PLTooL*, es posible arrancar el interfaz gráfico de la herramienta desarrollada. Para ello, hay que realizar los siguientes pasos:

1. Se habrá generado el script *pltool.sh*. Situarse en el mismo directorio donde se encuentra este script. Dar a este fichero permisos de ejecución:

   ```
   chmod +x pltool.sh
   ```

2. Para ejecutar la interfaz gráfica escribir:

   ```
   ./pltool.sh
   ```

El interfaz gráfico consta principalmente de cinco componentes o módulos que implementan la funcionalidad principal de la herramienta. Con cada uno de estos componentes es posible interactuar a través de un interfaz gráfico:

- La primera interfaz con la que se encuentra el usuario nada más arrancar el programa es la que le permite la elaboración de planes. El planificador por defecto es *IPSS* o *Prodigy 4.0*.

- La segunda interfaz es la que permite el aprendizaje de reglas mediante la interacción gráfica con *Hamlet*.
- La tercera interfaz se emplea para el aprendizaje de macro-operadores.
- La cuarta se utiliza para traducir entre los lenguajes PDDL2.1 e IPSS.
- La quinta y última se corresponde con el Experimenter y se emplea para realizar experimentos con varios dominios, problemas y planificadores al mismo tiempo.

2.4. Configuración del idioma

Por defecto, el idioma de la interfaz está establecido a español. Es posible modificar el valor de cierta variable lo que haría que se mostrara la aplicación en inglés. Esta variable se encuentra en el fichero *init.lisp* en el directorio *ipss* y su nombre es **language**. Si modificamos su valor a *'en* la interfaz se mostrará en inglés.

```
(defvar *language* 'en)
(setf *language* 'en)
```

3. Módulo planificación

En esta interfaz se presenta todo lo necesario para la carga de dominios y problemas que previamente habremos definido. Asimismo se facilita la opción de elegir el planificador que se desea utilizar para resolver el problema.

Además, se permiten la selección de múltiples opciones que le son proporcionadas al planificador elegido para construir el árbol de búsqueda. Cada planificador tendrá sus propias opciones que serán completamente independientes de las de los demás planificadores. La interfaz se muestra en la Figura 1.

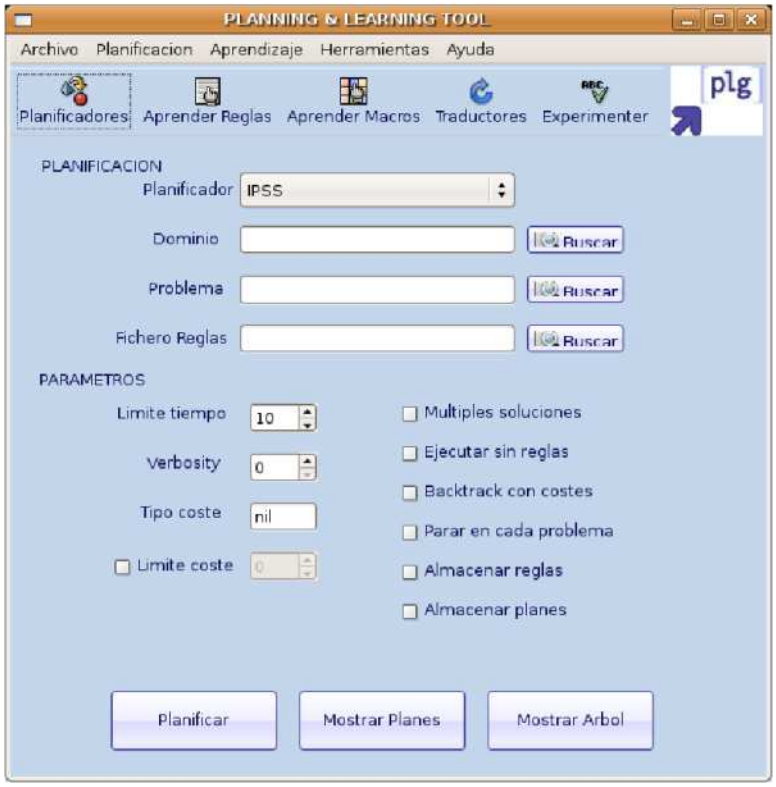

Figura 1: Interfaz Planificación

A continuación se describirá brevemente cada uno de los planificadores que se pueden elegir en la herramienta y cuáles son las diferentes opciones de cada uno. Un planificador no es más que un programa de ordenador que permite obtener planes para resolver problemas. Un plan no es más que la secuencia de operadores que debemos aplicar para dado un estado inicial de partida llegar al estado final deseado. Para construir el árbol de búsqueda con el planificador hay que realizar cuatro pasos fundamentales.

- Selección del planificador que se empleará.
- Suministrar a la herramienta el nombre del dominio.
- Suministrar a la herramienta el nombre del problema.
- Seleccionar el fichero que contiene las reglas de control que se aplicarán (opcional).
- Seleccionar los parámetros.

Las diferentes opciones que presenta cada planificador se explicarán por separado, puesto que cada planificador posee sus opciones propias.

3.1. Selección del planificador

La interfaz presenta al inicio un menú desplegable en el que es posible seleccionar el planificador que se después se empleará. Por defecto el planificador que aparece es *IPSS* o *Prodigy 4.0* como se puede observar en la Figura 1.

El resto de planificadores que se pueden elegir en la herramienta son: *SAYPHI*, *METRICFF*, *SGPLAN* y *LPG*.

A continuación pasamos a describir cada uno de los planificadores y sus opciones.

3.2. Planificadores

En esta sección se describirán detalladamente cada uno de los planificadores que forman parte de la herramienta.

3.2.1. Planificador IPSS (Prodigy 4.0)

Prodigy 4.0 es un planificador no lineal que utiliza heurísticas de forma que partiendo de las metas u objetivos que todavía no están resueltos selecciona los operadores adecuados que le permiten alcanzar esos objetivos [Veloso et al, 1995]. Su interfaz gráfica se puede consultar en la Figura 1.

Tanto el dominio como el problema que se le va a pasar a *IPSS* debe estar especificado en código lisp.

IPSS tiene además la opción de indicarle un fichero de reglas de control en el que se basará el planificador, con el fin de minimizar al máximo el árbol de busqueda.

Los parámetros que se observan en la Figura 1 del planificador *IPSS* son:

- **Límite tiempo** (valor nil o entero). Limita el máximo tiempo en segundos en que busca solución. Por defecto son 10 segundos.
- **Verbosity** (valor [0, 3]). Controla la información a imprimir por pantalla. 0, no se escribe nada; 1 sólo se escribe el plan resultante; 2 escribe información de como se expanden los nodos y el plan; 3 escribe todo lo anterior y las reglas de control que se disparan. Por defecto 0.
- **Tipo coste**. Función de coste a usar. Por defecto valor nil.
- **Límite coste**. Para especificar un límite de coste de la solución encontrada. Por defecto valor 0 y sin seleccionar esta opción.

- **Multiples soluciones**. Si se selecciona esta opción, el planificador no se detiene al encontrar una solución sino que buscará todas las posibles soluciones de problema. Por defecto no seleccionada.

- **Ejecutar sin reglas**. La ejecución se realizará sin cargar las reglas de control especificadas. Por defecto no seleccionada.

- **Backtrack con coste**. Si se selecciona esta opción conjuntamente con la opción *Multiples soluciones* se mostrarán todas las soluciones al problema con un coste menor o igual a la última solución encontrada. Por defecto no seleccionada.

- **Parar en cada problema**. Esta opción sólo tiene efecto si se especifica un conjunto de problemas y provoca que el planificador se detenga en cada problema. Por defecto no seleccionada.

- **Almacenar reglas**. La selección de esta opción hace que las reglas que se hayan disparado durante la planificación se almacenen en una estructura. Posteriormente, estas reglas almacenadas, podrán ser visualizadas cuando se muestre el árbol de búsqueda construido. Por defecto no seleccionada. (Para más información sobre las reglas de control ver sección 4).

- **Almacenar planes**. La selección de esta opción tiene resultados parecidos a la anterior. Cuando es seleccionada, los planes que se han construido son almacenados en una estructura. Por defecto no seleccionada. (Para más información sobre los planes ver sección 3.3).

3.2.2. Planificador SAYPHI

SAYPHI es un sistema de planificación y aprendizaje que intenta integrar diferentes técnicas para adquirir conocimiento de control que soporten la búsqueda de un planificador heurístico común. Su interfaz gráfica se puede consultar en la Figura 2.

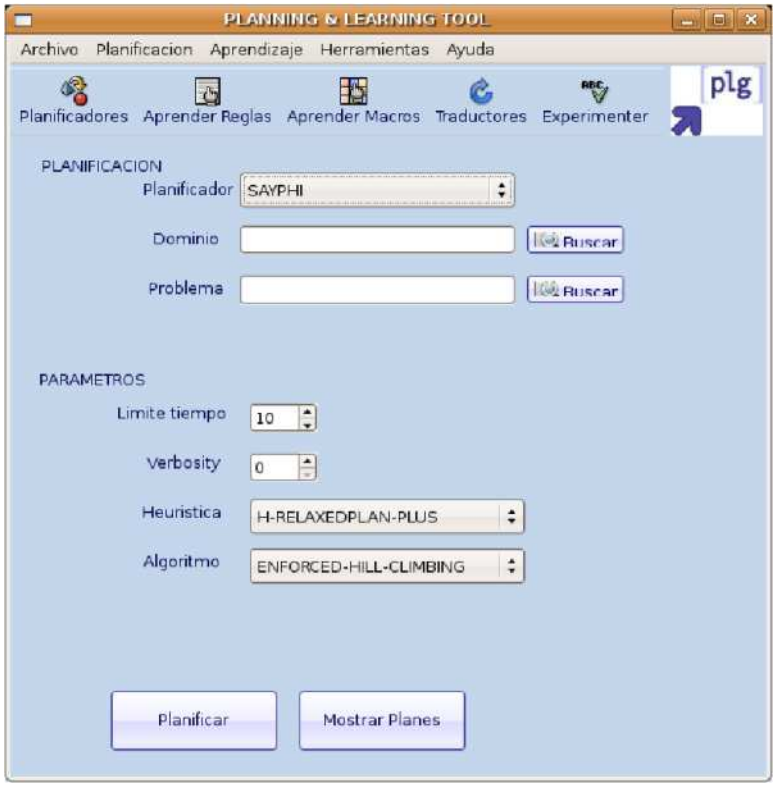

Figura 2: Interfaz SAYPHI

Tanto el dominio como el problema que se le va a pasar a *SAYPHI* debe estar especificado en lenguaje PDDL2.1.

Los parámetros que se observan en la Figura 2 del planificador *SAYPHI* son:

- **Límite tiempo** (valor nil o entero). Limita el máximo tiempo en segundos en que busca solución. Por defecto son 10 segundos.
- **Verbosity** (valor [0, 3]). Controla la información a imprimir por pantalla. 0, no se escribe nada; 1 sólo se escribe el plan resultante; 2 escribe información de como se expanden los nodos y el plan; 3 escribe todo lo anterior y las reglas de control que se disparan. Por defecto 0.
- **Heurística**. Función de heurística a utilizar en la planificación. Por defecto H-relaxedplan.
  - **H-relaxedplan**. Heurística que coincide con la de Metric-FF. Heurística que cuenta el número de pasos del plan relajado, que

se obtiene al resolver un problema relajado planteado al no considerar los borrados en las acciones del dominio.

- **H-metric-rxplan**. Heurística que *calcula* el coste del plan relajado en lugar de la longitud del mismo. Utilizada especialmente para dominios con variables númericas en los que se intenta optimizar una métrica para el problema.
- **H-relaxedplan-plus**. Heurística que añade a la heurística del plan relajado una fracción entre 0 y 1 que representa la dificultad de las precondiciones de las acciones que aparecen en el plan relajado

- **Algoritmo**. Algoritmo que se utilizará en la planificación. Por defecto Enforced-hill-climbing.

  - **Enforced-hill-climbing**: Es un algoritmo basado en el algoritmo de búsqueda local *Hill-climbing*. Enforced-Hill-Climbing lo que hace es buscar una mejor solución en sus siguientes sucesores y en caso de que no encuentre ninguna solución mejor, vuelve a la solución óptima anterior para seguir buscando una mejor solución.
  - **Hill-climbing (Escalada)** Hill-climbing busca una solución y mediante sucesivos cambios en dicha solución intenta mejorar el valor heurístico de la misma. De este modo se va a cercando a la solución óptima como si fuera escalando una montáña de mejor en mejor solución, con este algortimo no se puede asegurar que la solución encontrada sea la ótima.
  - **A-star**: El algortimo A-estrella, es un algoritmo que tiene en cuenta el valor de cada nodo según la función f(n) = g(n) + h(n). La función g(n) es el coste de llegar hasta dicho nodo, mientras que h(n) es el coste de llegar desde este nodo a la meta según la heurística h. En todo momento con este algortimo se puede cambiar el valor de f(n) de un nodo si se encuentra el modo de llegar al mismo con un menor coste. Este algoritmo se suele usar para hayar el menor coste o camino entre dos puntos en un grafo.
  - **DFBNB**. Algoritmo que refina la solución encontrada por el Enforced-hill-climbing con una técnica de *Branch-and-Bound*.

3.2.3. Planificador METRIC-FF

METRIC-FF es un sistema de planificación independiente del dominio desarrollado por Jörg Hoffmann. Este sistema es una extensión del planificador FF (combinado con ADL) para variables de estado numéricas. Este

sistema esá implementado completamente en C. Ha participado en la tercera IPC en los dominios numéricos, demostrando un rendimiento muy competitivo. Su interfaz gráfica se puede consultar en la Figura 3.

Figura 3: Interfaz METRIC-FF

Tanto el dominio como el problema que se le va a pasar a *METRIC-FF* debe estar especificado en lenguaje PDDL2.1.

METRIC-FF tiene además la opción de indicarle un fichero de reglas de control en el que se basará el planificador, con el fin de minimizar al máximo el árbol de búsqueda.

Los parámetros que se observan en la Figura 3 del planificador *METRIC-FF* son:

- **Límite tiempo** (valor nil o entero). Limita el máximo tiempo en segundos en que busca solución. Por defecto son 10 segundos.
- **Max-tam**. Tamaño máximo de la lista abierta en el algoritmo WA*. Por defecto sin limite de tamaño. El algoritmo WA* esta basado en los mismos principios que el algoritmo A*, pero en este caso se le añade

un peso a cada una de las partes de la formula: heuristica y coste. De modo que se puede dar más importancia a una de las dos partes. Para más información sobre el algoritmo A* consultar la sección 3.2.2.

- **Peso g**. Fija el valor de wg en la formula wg*g(s) + wh*h(s). Por defecto valor 1.
- **Peso h**. Fija el valor de wh en la formula wg*g(s) + wh*h(s). Por defecto valor 5.
- **Optimización de métrica**. Tiene en cuenta la función de optimización, por defecto es la longitud del plan.
- **Heurística**. Función de heurística a utilizar en la planificación.
  - **Metric-ff**. Heurística que se corresponde a una variación del metodo de escalada, en la cúal en cada paso de búsqueda se realiza una resolución de planes relajados. Con los resultados de los planes relajados se elije el camino que menos distancia le quede hasta la solución final.
  - **Graphplan-with-fixed-cost-step**. Acción más barata del dominio utilizando Graphplan.
  - **Cheapest-aplicable-action-with-cost>0**. Acción más barata del dominio utilizando Graphplan cuyo coste sea mayor que 0.
  - **Cheapest-aplicable-action-with-cost>=0**. Acción más barata del dominio utilizando Graphplan cuyo coste sea mayor o igual que 0.
  - **Cheapest-aplicable-action-with-cost>0-including-delayed-actions**. Acción más barata del dominio utilizando Graphplan cuyo coste sea mayor que 0 incluyendo las acciones retrasadas.
  - **Cheapest-aplicable-action-with-cost>0-including-delayed-actions**. Acción más barata del dominio utilizando Graphplan cuyo coste sea mayor o igual que 0 incluyendo las acciones retrasadas.
- **Algoritmo**. Algoritmo que se utilizará en la planificación.
  - **Best-first**. Con esta opción no se intenta Enforced-hill-climbing sino que se intenta "mejor primero" antes.
  - **Enforced-hill-climbing**. Con esta opción se fuerza al planificador a utilizar "Enforced hill climbing".

- **Anytime**. Número de soluciones a encontrar por el planificador y de qué modo.
  - **Stop-at-first-solution**. Parar en la primera solución encontrada.
  - **Anytime-WA*-prunning-by-g-values**. En cualquier momento con el algoritmo WA*, podando por valores de g.
  - **Anytime-WA*-prunning-by-g-values-and-using-helpful-actions-prune**. En cualquier momento con el algoritmo WA*, podando por valores de g y usando acciones útiles para la poda.
  - **Anytime-WA*-prunning-by-f-values**. En cualquier momento con el algoritmo WA*, podando por valores de f.
  - **Anytime-WA*-prunning-by-f-values-and-using-helpful-actions-prune**. En cualquier momento con el algoritmo WA*, podando por valores de f y usando acciones útiles para la poda.

3.2.4. Planificador SGPLAN

SGPLAN es un planificador que quedó el primero en la IPC del 2006 en la parte determinística de la competición. La interfaz gráfica de *SGPLAN* dentro de la herramienta se puede consultar en la Figura 4.

Figura 4: Interfaz SGPLAN

Tanto el dominio como el problema que se le va a pasar a *SGPLAN* debe estar especificado en lenguaje PDDL3.

Los parámetros que se observan en la Figura 4 del planificador *SGPLAN* son:

- **Límite tiempo** (valor nil o entero). Limita el máximo tiempo en segundos en que busca solución. Por defecto son 10 segundos.

3.2.5. Planificador LPG

LPG es un planificador que participó en la IPC en 2004. Es un planificador estocástico basado en un algoritmo de mejor primero similar al que usa FF. SGPLAN es recomendado para dominios que tienen cantidades numéricas y duraciones. La interfaz gráfica de *LPG* dentro de la herramienta se puede consultar en la Figura 5.

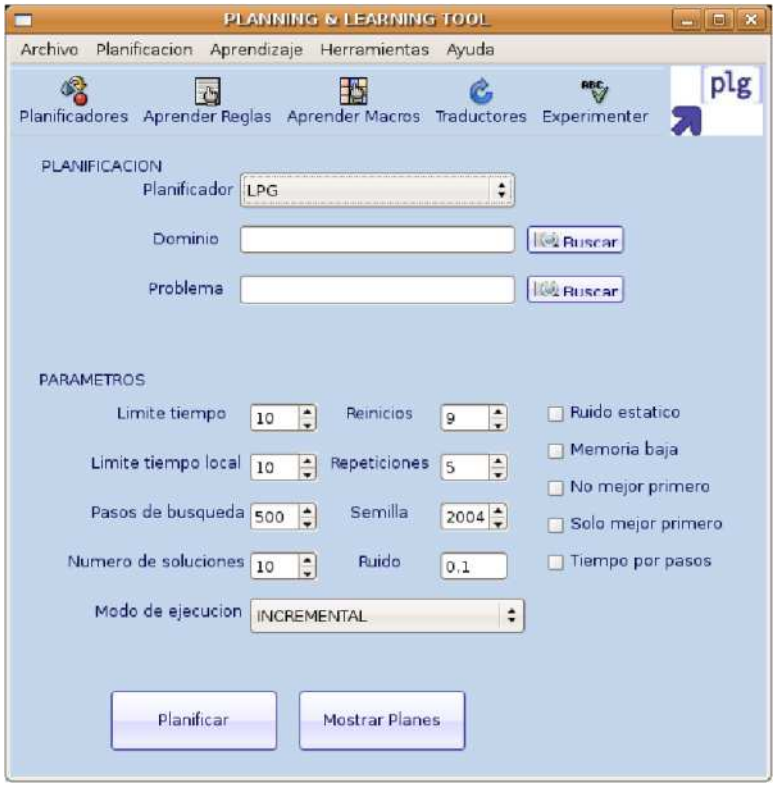

Figura 5: Interfaz LPG

Tanto el dominio como el problema que se le va a pasar a *LPG* debe estar especificado en lenguaje PDDL2.2.

Los parámetros que se observan en la Figura 5 del planificador *LPG* son:

- **Límite tiempo** (valor nil o entero). Limita el máximo tiempo en segundos en que busca solución. Por defecto son 10 segundos.
- **Límite tiempo local**. Especifica el máximo tiempo que el planificador usará para el procedimiento de la búsqueda local. Por defecto son 10 segundos.
- **Pasos en la búsqueda**. Indica el número de pasos antes del primer reinicio en la busqueda local. El valor por defecto es 500.
- **Número de soluciones**. Número de soluciones a encontrar como máximo en el modo de ejecución incremental. Por defecto tomará valor 10. Variable que no se tiene en cuenta excepto en el modo incremental.
- **Número de reinicios**. Número máximo de reinicios. Por defecto valor 9.

- **Semilla inicial**. Fija el valor de la semilla del generador aleatorio de números. Por defecto tomará el valor 2004.
- **Valor del ruido**. Indica el valor inicial del ruido del Walkplan. Walkplan es la heurística básica en la que se funda LPG, esta heurística se define como un procedimiento de busqueda local estocástica que se usa para encontrar la planificación final. Valores entre cero y uno. Por defecto tiene el valor 0.1.
- **Fijar el ruido como estático**. Fija el valor del ruido a un valor fijo y estático.
- **Utilizar en modo de memoria baja**. Computa las relaciones de mutex entre distintas acciones en tiempo de ejecución.
- **No usar modo de mejor primero**. Apaga la busqueda de mejor primero y activa la busqueda local estocástica en grafos de acción.
- **Sólo usar modo de mejor primero**. Ejecuta inmediatamente la busqueda por mejor primero.
- **Estudiar el coste por pasos**. Fija la calidad del plan segun métrica como el número de pasos del plan.
- **Modo de ejecución**.
  - **Incremental**. Busca soluciones hasta que se llega al máximo número de soluciones buscadas o se cumplen las restricciones impuestas en la planificación.
  - **Speed**. Busca la solución del modo más rapido, muestra la primera solución que encuentra.
  - **Quality**. Busca la solución cuyo plan sea el que nos de la mejor calidad.

3.3. Visualizar el plan construido

Una vez que el planificador termine la ejecución se mostrará al usuario un mensaje en el cual se le indicará si la planificación ha sido o no terminada con exito. En caso de que la planifación haya sido un exito se podra consultar el plan generado por el planificador para solucionar el problema. Este plan se consulta pulsando el botón *Mostrar Planes*.

En la Figura 6 se puede observar la interfaz donde se muestran los planes.

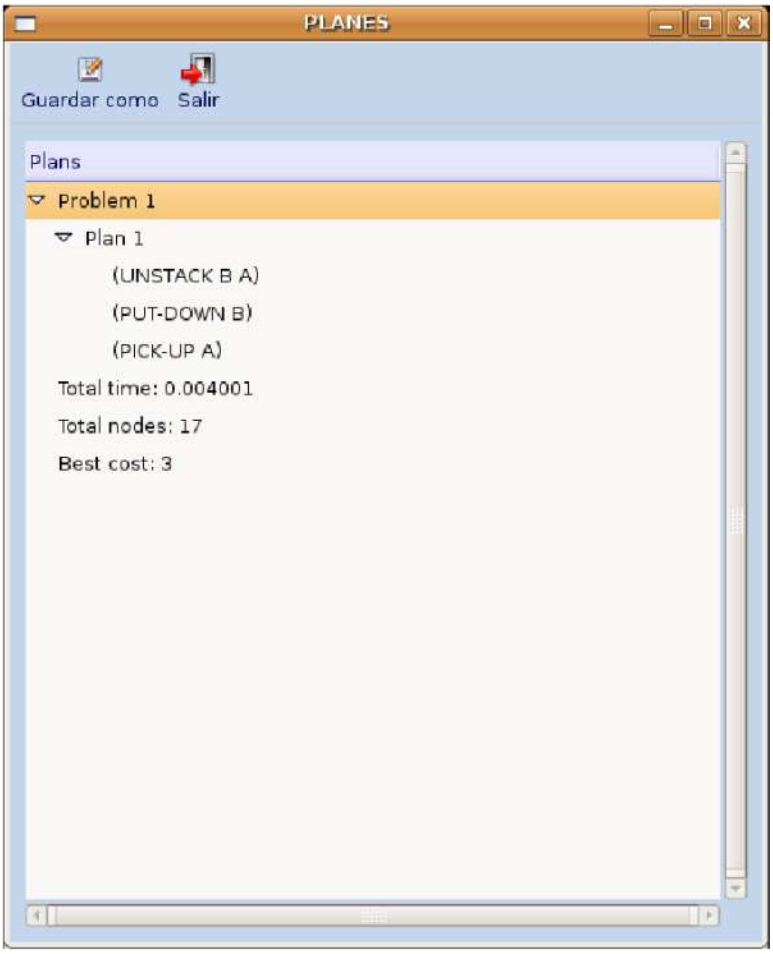

Figura 6: Ejemplo de visualización de plan

En esta interfaz, aparece un botón arriba a la izquierda *Guardar como*, qué nos permitirá almacenar los planes obtenidos en un archivo con formato XML. Este archivo se podrá abrir en cualquier navegador web y podrá ser almacenado en la carpeta que el usuario desee y con el nombre que el usuario indique.

3.4. Visualizar el árbol de búsqueda

Prodigy nos da la opción de consultar cual ha sido el árbol de búsqueda construido para llegar a la solución del problema. Una vez que el planificador ha terminado con exito una planificación se puede pulsar el botón de *Mostrar Arbol*.

En la Figura 7 se muestra la interfaz donde se seleccionaran las opciones

con las que se mostrará el árbol de búsqueda.

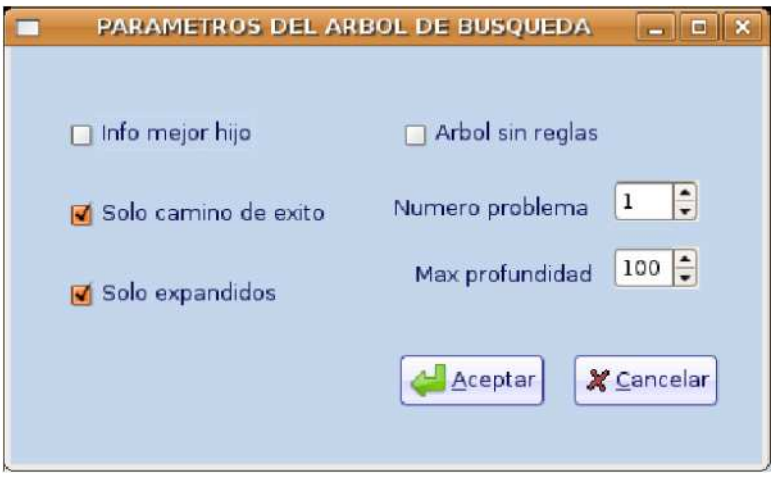

Figura 7: Interfaz con las opciones de visualización del árbol de búsqueda

Las opciones que se pueden seleccionar son:

- **Info mejor hijo**: En la posterior visualización del árbol mostrará información sobre el mejor hijo.
- **Solo camino de éxito**: Si se selecciona esta opción posteriormente sólo se visualizarán los nodos de éxito y los nodos de fallo que divergen en un paso de los caminos de éxito.
- **Solo expandidos**: Si esta opción se marca sólo se mostrarán los nodos expandidos.
- **Árbol sin reglas**: Se muestra el árbol construido sin mostrar las reglas de control disparadas.
- **Número problema**: Permite seleccionar el número de problema que se quiere visualizar.
- **Max profundidad**: Permite seleccionar hasta qué profundidad queremos visualizar el árbol de búsqueda.

Un ejemplo de arbol de búsqueda con todas las opciones sin seleccionar se corresponde con la Figura 8

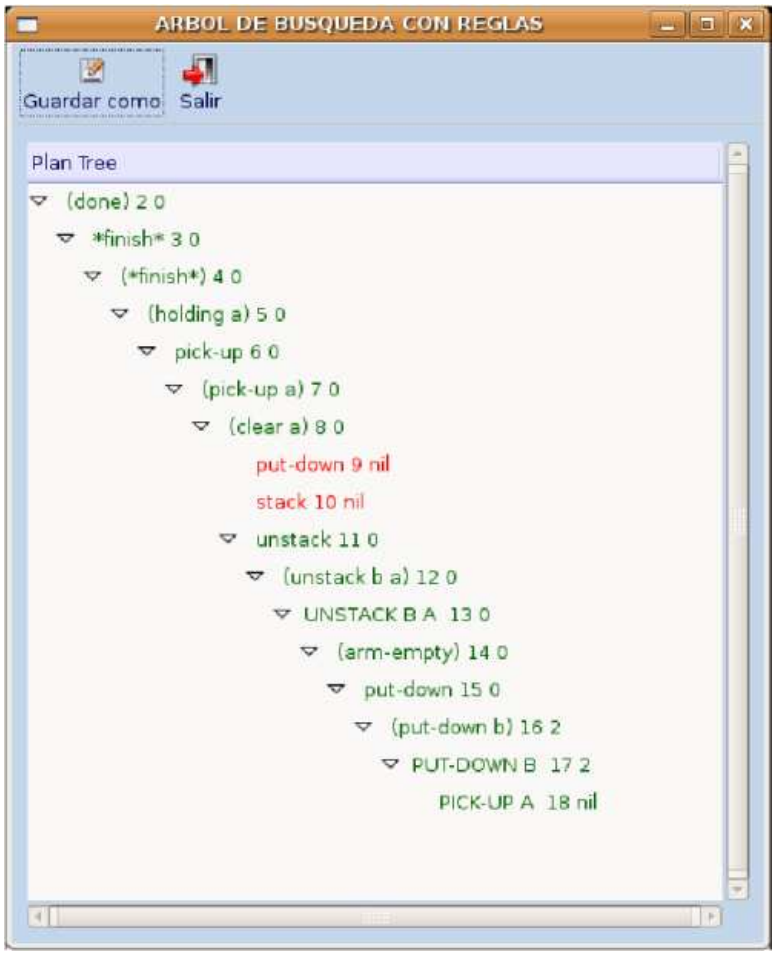

Figura 8: Ejemplo de árbol de búsqueda

En la Figura 8 se puede ver el árbol construido por Prodigy 4.0. Los nodos con algo entre paréntesis indican un objetivo que Prodigy trata de resolver, los nodos con algo entre < ... > indican un posible operador instanciado que Prodigy quiere utilizar para alcanzar el objetivo. Si además estos últimos nodos están en mayúsculas indican que Prodigy 4.0 ha aplicado el operador y por tanto ha cambiado el estado. Las flechas indican el orden en que Prodigy 4.0 avanza por los nodos. Los nodos en rojo nos indican un camino que ha sido recorrido por el algoritmo y mediante el cual no se llega al objetivo

En esta interfaz, aparece un botón arriba a la izquierda *Guardar como*, qué nos permitirá almacenar el árbol obtenido en un archivo con formato XML. Este archivo se podrá abrir en cualquier navegador web y podrá ser almacenado en la carpeta que el usuario desee y con el nombre que el usuario indique.

4. Módulo de aprendizaje de reglas de control

Este módulo de la herramienta se apoya en el sistema *Hamlet* que permite el aprendizaje de reglas de control (heurísticas) para *Prodigy 4.0* [Veloso et al, 1996]. *Hamlet* está integrado en el planificador no lineal *Prodigy 4.0*. Las entradas de *Prodigy 4.0* como se pudo comprobar anteriormente son:

- La teoría del dominio D (o simplemente dominio).
- El problema, especificado en términos de una configuración inicial del mundo (o estado inicial, S) y un conjunto de metas que se pretenden conseguir (G).
- Un conjunto de reglas de control (C) que guía las decisiones en el proceso de obtención del plan.

Hamlet, por tanto, está integrado en el planificador *Prodigy 4.0*. Las entradas que se proporcionan a *Hamlet* son:

- El dominio, D.
- Un conjunto de problemas de entrenamiento, P.
- Una medida de calidad, Q.
- Un modo de aprendizaje, L.
- Un parámetro de optimalidad, O.

La salida es un conjunto de reglas de control, C. *Hamlet* tiene dos módulos principales: el módulo Bounded-Explanation y el módulo de refinamiento. En la Figura 9 se muestra el funcionamiento de Hamlet y como se integra con el planificador *Prodigy 4.0*.

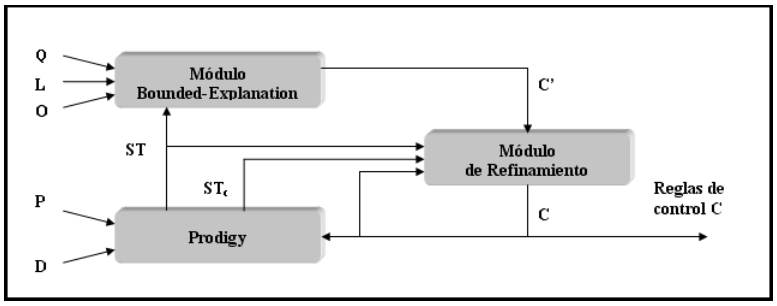

Figura 9: Hamlet.

El módulo Bounded-Explanation genera reglas de control (C') a partir de los árboles de búsqueda generados por Prodigy (ST_C). Estas reglas de control pueden ser demasiado específicas o demasiado generales. El módulo de refinamiento soluciona el problema de las reglas demasiado específicas a partir del análisis de ejemplos positivos. Además, reemplaza las reglas demasiado generales con otras más específicas cuando encuentra situaciones en las cuales las reglas aprendidas conduzcan a decisiones incorrectas. Poco a poco Hamlet aprende las reglas de control convergiendo al conjunto de reglas correctas, es decir, aquellas reglas que no son ni demasiado generales ni demasiado específicas. ST y ST_C son planes generados por Prodigy en dos llamadas diferentes al planificador, C es el conjunto de reglas y C' es el nuevo conjunto de reglas aprendidas por el módulo Bounded-Explanation. Por cada problema $p \in P$ *Hamlet* realiza dos llamadas a *Prodigy 4.0*. En la primera de ellas se genera el árbol de búsqueda ST que es el resultado de resolver el problema p sin el empleo de ninguna regla de control buscando exahustivamente en el espacio de búsqueda hasta encontrar la solución óptima. Hamlet genera nuevos ejemplos positivos a partir de ST. En la segunda llamada que *Hamlet* realiza a *Prodigy 4.0*, se emplea el conjunto de reglas de control aprendidas C y Prodigy 4.0 genera el árbol de búsqueda ST_C. Hamlet identifica posibles ejemplos negativos al comparar al árbol de búsqueda que ha sido podado con el uso de reglas de control, ST_C, con el árbol completo de búsqueda ST. Los ejemplos positivos y negativos son usados para refinar las reglas de control aprendidas y de esta forma producir un nuevo conjunto de reglas, C.

La interfaz de este módulo de la herramienta puede verse en la Figura 10.

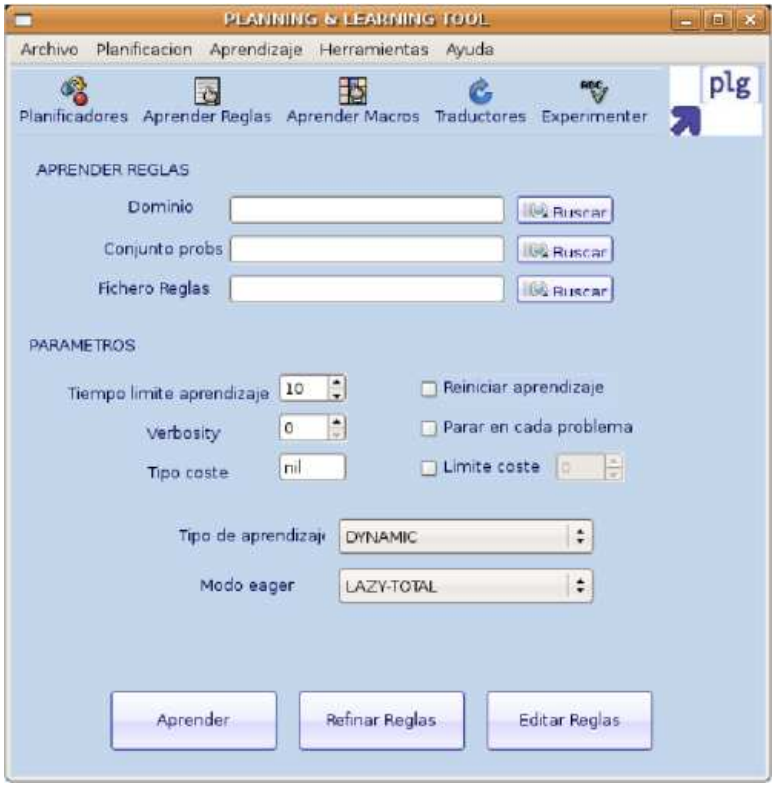

Figura 10: Módulo de aprendizaje de reglas.

Para realizar el aprendizaje de reglas es necesario:

- Proporcionar el dominio del que se quieren obtener las reglas.
- Proporcionar el conjunto de problemas de entrenamiento, P.
- Proporcionar un conjunto de reglas de partida (Opcional).
- Establecer los parámetros de la búsqueda.

4.1. Especificación del dominio

En este caso habrá que especificar a la herramienta cual es el dominio del que queremos extraer reglas. Para ello, pulsaremos el botón *Buscar* que se encuentra a la derecha del campo de texto etiquetado con *Dominio*.

4.2. El conjunto de problemas de entrenamiento

Es necesario proporcionar a *Hamlet* un conjunto de problemas de entrenamiento, P, con los cuales obtendrá el conjunto de reglas de control, C.

Para ello, habrá que crear un directorio llamado *probsets* en el mismo directorio donde tenemos almacenado nuestro dominio. Dentro del directorio *probsets* habrá que editar un fichero con la definición de los problemas del dominio que van a pasar a formar parte de nuestro conjunto de entrenamiento. De hecho, habrá que crear un fichero por cada conjunto de problemas que se quiera utilizar. En cada uno de estos ficheros habrá que escribir la definición de los problemas que conforman el conjunto de entrenamiento:

```
(setf *all-problems* '(def-problem1 def-problem2 ... def-problemn))
```

donde $def - problem_i$ es la definición del problema i que tiene la forma:

```
(setf (current-problem) ...)
```

Supongamos que tenemos tres problemas: pgh1, pgh2 y pgh3. Cada uno definido en su propio archivo. Con estos tres problemas crearemos el fichero *problemas.lisp* en el directorio *probsets* que contendrá

```
(setf *all-problems* '(

;; pgh1
(setf (current-problem)
      (create-problem
       (name pgh1)
       (objects
...
;; pgh2

(setf (current-problem)
      (create-problem
       (name pgh2)
       (objects
...
;; pgh3

(setf (current-problem)
      (create-problem
       (name pgh3)
       (objects
    ...
))
```

Con lo que nuestro conjunto de entrenamiento estará formado por tres problemas diferentes. Para informar a la herramienta de que éste será el conjunto de problemas de entrenamiento que se emplee habrá que pulsar el botón *Buscar* situado a la derecha del campo de texto etiquetado como *Conjunto de probs* y seleccionar el fichero donde hemos definido el conjunto de entrenamiento, en nuestro caso seleccionaremos el fichero *problemas.lisp*.

4.3. Especificación de las reglas de control

Es posible introducir al inicio del algoritmo un conjunto de reglas de control en vez de partir de un conjunto de reglas vacio. Para ello, pulsaremos sobre el botón *Buscar* que se encuentra a la derecha del campo de texto *Fichero reglas* y seleccionaremos el fichero que almacena las reglas de control.

4.4. Parámetros

Es posible indicar al algoritmo cómo debe actuar mediante el establecimiento de ciertos parámetros.

- **Limite tiempo aprendizaje**(valor nil o entero). Limita el máximo tiempo en segundos en que busca solución. Por defecto son 10 segundos.
- **Verbosity** (valor [0, 3]). Controla la informacion a imprimir por pantalla. 0, no se escribe nada; 1 sólo se escribe el plan resultante; 2 escribe informacion de como se expanden los nodos y el plan; 3 escribe todo lo anterior y las reglas de control que se disparan. Por defecto 0.
- **Tipo coste**. Función de coste que se empleará.
- **Reiniciar aprendizaje**. Si se quiere seguir aprendiendo de experiencias pasadas o se quiere comenzar a aprender desde el principio.
- **Parar en cada problema**. Si se selecciona esta opción se detendrá la ejecución del aprendizaje de reglas en cada problema.
- **Limite coste**. Para especificar un límite de coste en las soluciones.
- **Tipo aprendizaje**. Es posible seleccionar entre diferentes tipos de aprendizaje.
  - **Dynamic**. Aprendizaje dinámico.
  - **Deduction**. Sólo aprende mediante deducción.
  - **Deduction-Induction**. Deducción seguida de inducción.

- **EBL**. Aprendizaje EBL.
- **Active**. Se llama a un generador aleatorio en cada ciclo para obtener un nuevo problema.

- **Modo eager**. El sistema puede operar en dos modos de aprendizaje: *eager* y *lazy*. En el modo *eager* se genera un ejemplo positivo en cada decisión que conduce a una solución. En el modo *lazy*, se genera un ejemplo positivo si la decisión conduce a una de las mejores soluciones globales. Además, dependiendo de cómo se realice la búsqueda sobre el espacio de estados ésta puede ser total o parcial. Es total cuando se requiere la expansión total del espacio de búsqueda y parcial cuando se realiza una expansión del espacio de búsqueda limitada. La combinación de estos conceptos produce cuatro modos de búsqueda diferentes: *Lazy-total, Lazy-partial, eager-total y eager-partial*.

4.5. Aprendizaje de las reglas de control

Para obtener las reglas de control una vez que se ha establecido el dominio y el conjunto de problemas de entrenamiento que se empleará se pulsa el botón *Aprender*.

Tras la ejecución del algoritmo se generarán varios ficheros con estadísticas en el directorio del dominio que se haya ejecutado y entre los cuales se encuentra un fichero que contiene las reglas de control aprendidas.

4.6. Refinamiento de las reglas de control

El refinamiento de las reglas de control se realiza con el fin de eliminar las reglas supérfluas que se hayan generado.

Para acceder a esta función se pulsará el botón *Refinar Reglas*.

4.7. Edición de las reglas de control

Es posible editar las reglas de control aprendidas pulsando el botón *Editar Reglas* de la interfaz. A continuación se muestra la interfaz en la cual se pueden editar las reglas de control aprendidas en la Figura 11 y en la Figura 12.

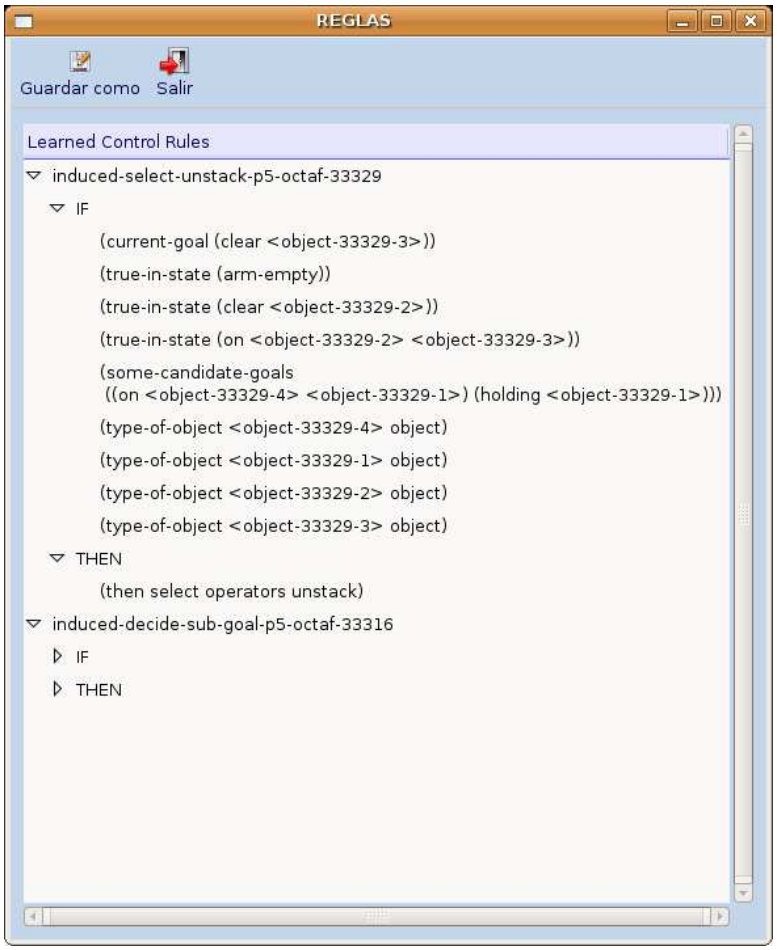

Figura 11: Ejemplo de edición de reglas de control: regla 1

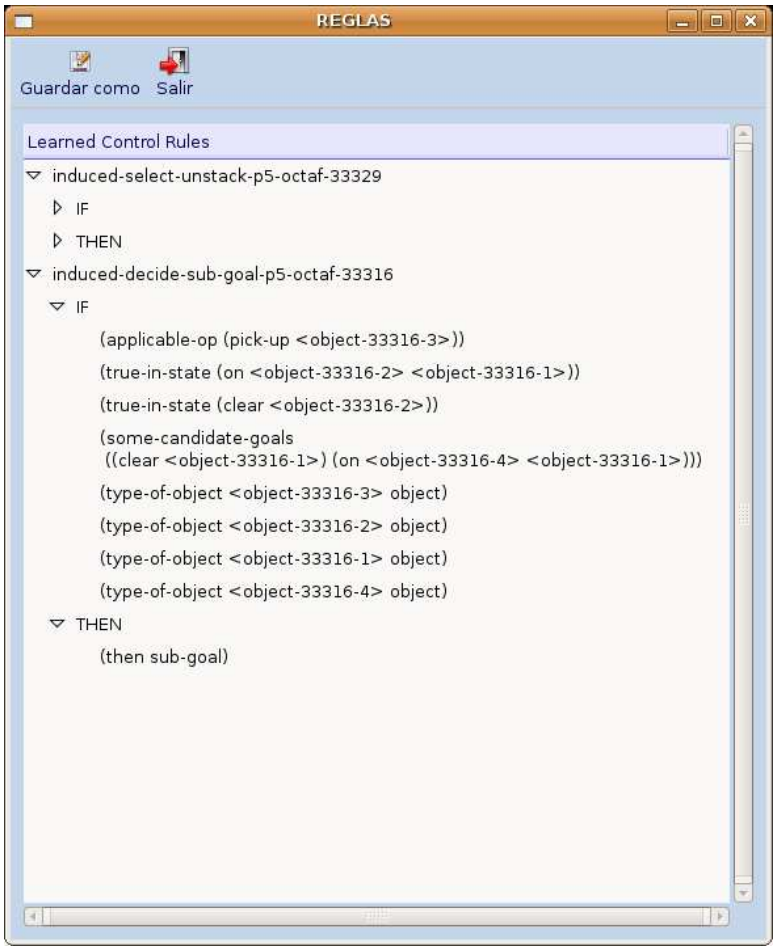

Figura 12: Ejemplo de edición de reglas de control: regla 2

Como se puede ver se han generado dos reglas de control. En la Figura 11 se observa desplegada la primera de las reglas de control, la segunda regla se encuentra desplegada en la Figura 12. Estas dos reglas de control son de dos tipos distintos:

- En la primera regla de control que se muestra *induced-select-unstack-p5-octaf-333329*, es un ejemplo de regla de control que elige un operador de entre todos los existentes para conseguir una meta. Con esta regla se esta modificando el modo de expansión por defecto de Prodigy, que es ejecutar el primer operador, según el orden en el que se han definido en el dominio, que consigue llegar a una meta.

- En la segunda regla de control *induced-decide-sub-goal-p5-octaf-33316*, la regla nos indica que hay que elegir la sub-meta siguiente de las pen-

dientes. No nos dice que operador ejecutar, si no que cambiemos de meta a cumplir primero por la sub-meta siguiente.

Como se puede observar el efecto que genera sobre la planificación cada una de las dos reglas anteriores es completamente distinto una de la otra.

5. Módulo de aprendizaje de *macro-operadores*

Este módulo dentro de la aplicación permite el aprendizaje de *macro-operadores* a partir de planes que se hayan aprendido previamente con el planificador *IPSS (Prodigy 4.0)*. Las distintas opciones que componen esta interfaz pueden verse en la Figura 13.

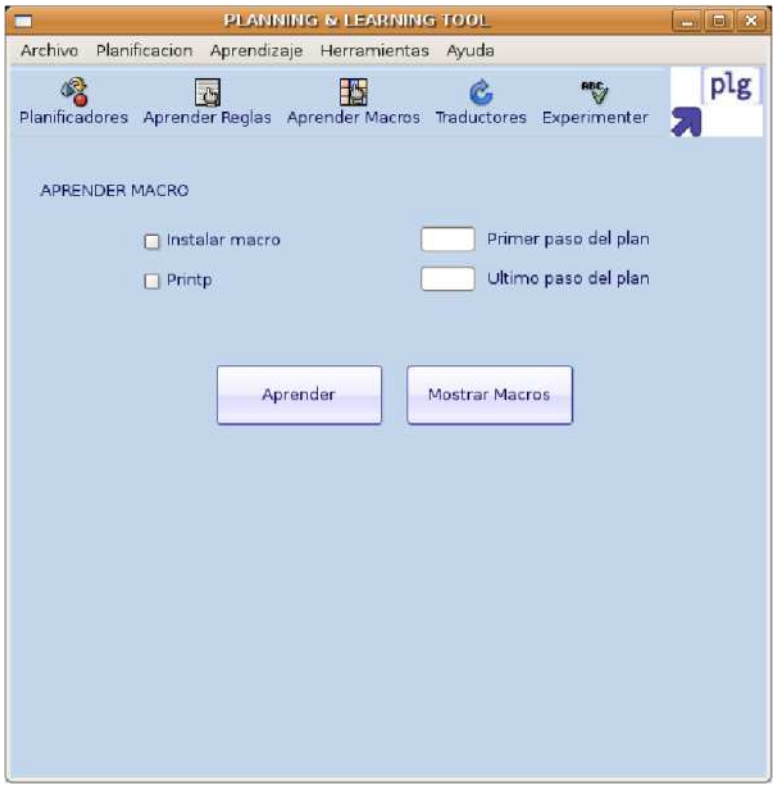

Figura 13: Interfaz para el aprendizaje de macro-operadores

La idea de los *macro-operadores* es memorizar secuencias de acción útiles aplicables a la planificación. Un *macro-operador* es por tanto una secuencia de acciones o de operadores que pueden ser tratados en conjunto.

Se pueden obtener *macro-operadores* mediante la utilización de tablas triangulares y planificación. Las tablas triangulares son una forma de representación de planes como se muestra en la Figura 14.

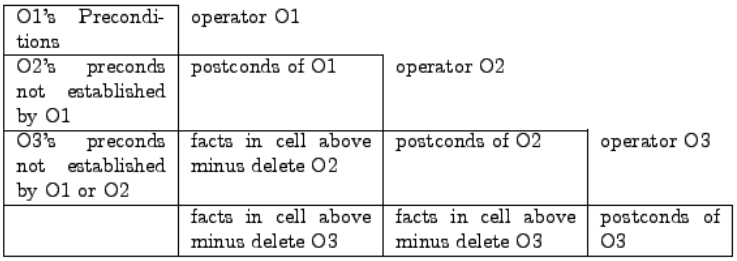

Figura 14: Tabla triangular de operadores

Por ejemplo, en el mundo de los bloques hay dos operadores, UNSTACK y PUT-DOWN, cuyas definiciones pueden verse en el fichero *domain.lisp* del directorio de prodigy *blocksworld*. Si se tiene un plan en el que aparece la secuencia de operadores:

```
UNSTACK(A, B)
PUT-DOWN(A)
```

La tabla triangular correspondiente a esta secuencia sería la que se muestra en la Figura 15.

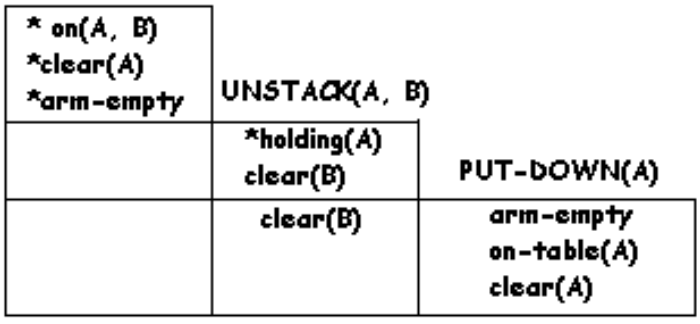

Figura 15: Tabla triangular de operadores

La tabla triangular generalizada para este ejemplo es la que se muestra en la Figura 16.

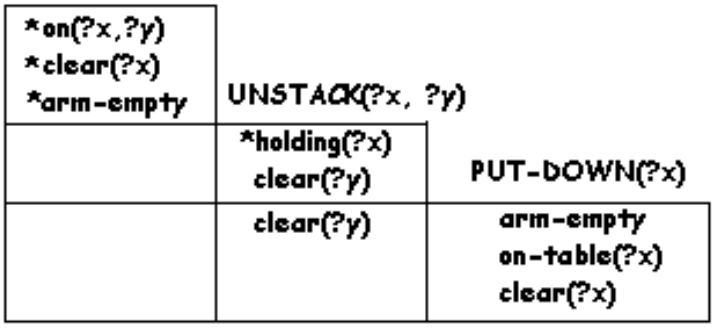

Figura 16: Tabla triangular de operadores

El macro-operador *UNSTACK-&-PUT_DOWN* resultante sería:

- **Precondiciones**: *on(?x,?y), clear(?x), arm-empty.*
- **adds**: *clear(?y), on-table(?x).*
- **dels**: *on(?x, ?y).*

5.1. Parámetros

La interfaz permite especificar los pasos del plan entre los cuales se va a contruir la tabla triangular correspondiente. Estos pasos pueden ser introducidos en los campos de texto etiquetados con *Primer paso plan* y *Ultimo paso plan*. Por defecto, si no se introduce nada en estos campos de texto se construye la tabla triangular sobre la totalidad de los operadores que constituyen el plan.

Cuando se ejecuta el programa pulsando el botón *Aprender*, en el directorio del dominio se crea el fichero *macros.lisp* que contiene los *macro-operadores* aprendidos. Si se tiene la opción *Instalar macro* seleccionada entonces se crearán dos ficheros más, el fichero *domain-without-macros.lisp* que se corresponde con el fichero del dominio original y el fichero *domain.lisp* que se corresponde con el fichero original del dominio al que se le han añadido los *macro-operadores* aprendidos.

Por último, si se marca la opción *printp* por el terminal se podrá visualizar la tabla triangular construida.

6. Módulo de traducción

Este módulo se emplea para traducir entre PDDL2.1 e IPSS. En la Figura 17 se muestra los componentes de este nuevo módulo de la aplicación.

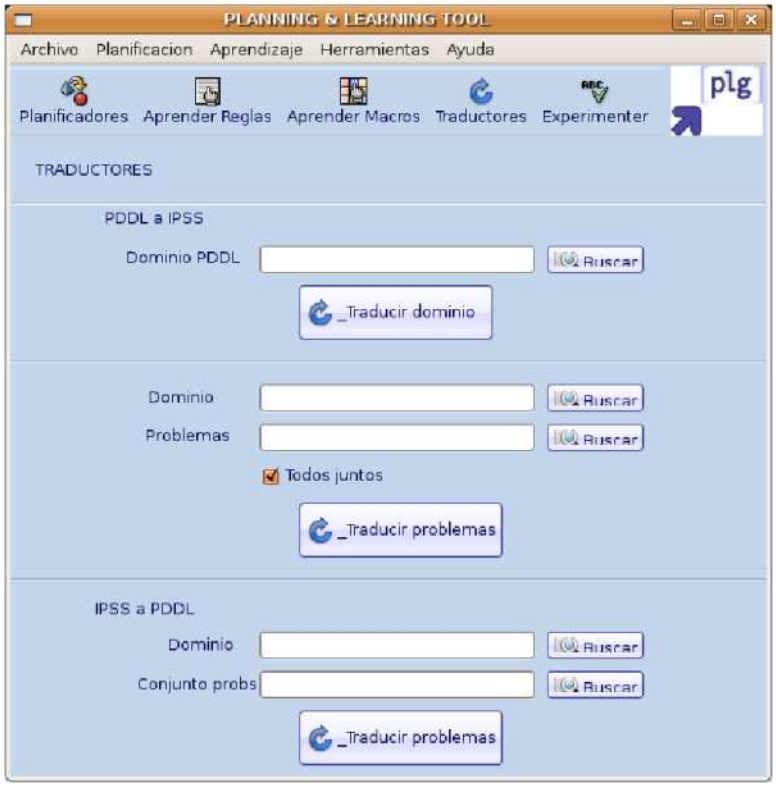

Figura 17: Interfaz correspondiente a la traducción entre PDDL2.1 e IPSS

La interfaz se divide en dos partes. La primera de ellas está destinada a la traducción de PDDL a IPSS y la segunda está destinada a la traducción de IPSS a PDDL2.1.

6.1. PDDL2.1 a IPSS

Se permite la traducción de dominios y problemas a IPSS. El primer campo de texto permite la traducción de un dominio en PDDL2.1 a IPSS mientras que los dos siguientes sirven para especificar el dominio y el conjunto de problemas en PDDL que se desean convertir a IPSS.

La variable **ipss-temporal-dir** definida en el fichero *init.lisp* en el directorio *ipss* tiene como valor por defecto *directorio-instalación/ipss/tmp*. Si se quiere modificar su valor habría que editar dicho fichero y cambiar su valor por defecto.

Para convertir un dominio PDDL2.1 a IPSS se han de seguir los siguientes pasos:

- Almacenar el fichero que contiene el dominio *pddl* en el directorio **ipss-*

*temporal-dir**. Concretamente se creará la siguiente estructura de directorios **ipss-temporal-dir*/domain-name/domain.pddl*.

- Posteriormente se pulsará el botón *Buscar* que se encuentra al lado del campo de texto etiquetado como *Dominio PDDL* y se seleccionará el fichero *domain.pddl* o bien el directorio que lo contiene */domain-name*.
- Si todo se ha realizado correctamente en el campo de texto de la interfaz aparecerá el nombre del dominio.
- La traducción se realizará pulsando el botón *Traducir Dominio*.
- Aparecerá una nueva ventana en la que se muestra el dominio original *.lisp* y el dominio *.pddl* al que se ha traducido.

Para convertir un conjunto de problemas escritos en PDDL2.1 a IPSS se han de llevar a cabo los siguientes pasos:

- Almacenar el fichero que contiene el dominio *pddl* en el directorio **ipss-temporal-dir**. Concretamente se creará la siguiente estructura de directorios **ipss-temporal-dir*/domain-name/domain.pddl*.
- El fichero del dominio se cargará de igual forma que en el caso anterior.
- También es necesario almacenar los problemas *.pddl* que se desean traducir en el directorio temporal **ipss-temporal-dir**. Concretamente se creará la siguiente estructura de directorios **ipss-temporal-dir*/domain-name/probs/probxxx.pddl*
- Se puede realizar la traducción de un único problema o de un conjunto de problemas.
  - En el primer caso se pulsará en el botón *Buscar* al lado del campo de texto *Problemas* y se seleccionará el problema que se quiere traducir.
  - En el caso que se quiera traducir un conjunto de problemas se escribirá directamente en el campo de texto *prob**. Para que funcione correctamente se ha supuesto que todos los problemas comienzan con *prob...* y el * es el carácter comodín.
- El problema o los problemas traducidos se almacenarán en el mismo directorio donde se encontraban los problemas .pddl.
- Si se selecciona la opción *Todos juntos* todos los problemas traducidos se almacenarán en un mismo fichero.
- Pulsando el botón *Traducir Problemas* se llevará a cabo la traducción.

6.2. IPSS a PDDL2.1

Los dos últimos campos de texto sirven para especificar el dominio y el conjunto de problemas en IPSS que se desean traducir a PDDL2.1.

La variable **ipss-temporal-dir** definida en el fichero *init.lisp* en el directorio *ipss* tiene como valor por defecto *directorio-instalación/ipss/tmp*. Si se quiere modificar su valor habría que editar dicho fichero y cambiar su valor por defecto.

Para realizar la traducción de un conjunto de problemas en IPSS a PDDL2.1 hay que realizar los siguientes pasos:

- Hay que construir la siguiente estructura de directorios **ipss-temporal-dir*/domain-name/domain.lisp* donde el fichero *domain.lisp* es el fichero con la especificación del dominio.
- Se pulsará el botón *Buscar* y se seleccionará el directorio donde se encuentra *domain.lisp*, es decir, */domain-name*.
- Después hay que construir el conjunto de problemas que se quieren traducir como se especificaba en la sección 3.2.2. Guarda este conjunto de problemas en el directorio **ipss-temporal-dir*/domain-name/probsets/probset.lisp*.
- Se seleccionará el fichero con la definición del conjunto de problemas pulsando el botón que aparece al lado del campo de texto etiquetado como *Conjunto de Problemas*.
- Pulsaremos el botón *Traducir Problemas* para traducir los problemas.

7. Módulo del experimenter

7.1. Introducción

Esta interfaz de la aplicación se utiliza para definir experimentos con varios dominios y problemas distintos, a la vez que varios planificadores con distintos parámetros cada uno.

Como se puede observar en la Figura 18 la interfaz esta dividida en tres zonas:

- **Área de control**: Zona donde se encuentran los botones de control sobre el experimento al completo.
- **Área de dominios y problemas**: Zona donde se podrá añadir y quitar nuevos dominios y problemas al experimento.

- **Área de planificadores**: Zona donde se podrá añadir y quitar nuevos planificadores con sus parametros propios al experimento.

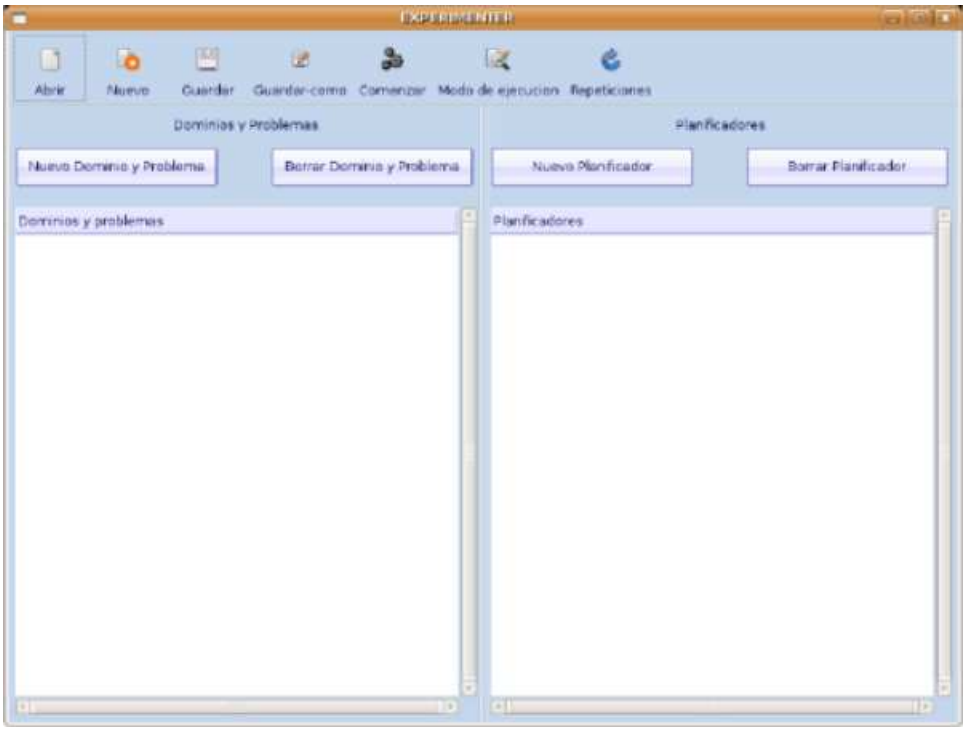

Figura 18: Interfaz del experimenter

A continuación se contará más en detalle cada una de estas zonas y para que sirve cada botón que se encuentre en ellas.

7.2. Área de control

La zona superior es la zona de control del experimenter, se van a describir uno a uno todos los botones dela parte superior que aparecen en la Figura 18.

- **Abrir**: Este botón se utiliza para abrir un archivo de experimento ya creado anteriormente. Al pulsar este botón se abrirá una interfaz de selección de archivo para que se pueda elegir el experimento a abrir. Una vez abierto la interfaz del *Experimenter* se actualizará mostrando los dominios y planificadores que forman parte del experimento.
- **Nuevo**: Este botón se utiliza para crear un nuevo experimento. Al pulsar este botón se abrirá una interfaz para indicar en qué carpeta y

con qué nombre se desea guardar el experimento. Se recomienda que todos los experimentos se almacenen con extensión ".exp".

- **Guardar**: Al pulsar este botón se guardarán los cambios realizados en el experimento.
- **Guardar como**: Este botón nos sirve para almacenar el experimento en un archivo que no se corresponde con el inicial. Cuando se pulse se abrirá una interfaz para que el usuario indique la carpeta y el nombre del archivo donde almacenar el experimento.
- **Comenzar**: Al pulsar este botón el experimento comenzará a ejecutarse.
- **Modo de ejecución**: Con este botón se elegirá el modo de ejecución del experimento. La interfaz de selección de modo se corresponde a la Figura 19. En esta interfaz se deben seleccionar dos opciones:
  - **Dominios o planificadores primero**: Con esta opción se le indica al *Experimenter* si se desea que se ejecute el experimento por planificadores o por dominios.

    Por planificadores significa que se coge el primer dominio y problema y se soluciona con todos los planificadores, al terminar se sigue con el siguiente dominio y problema con todos los planificadores, así hasta terminar con todos los dominios y problemas.

    Por dominios significa que se elige el primer planificador y se aplica a todos los dominios y problemas, al terminar se sigue con el siguiente planificador y se aplica a todos los dominios y problemas, asi hasta terminar con todos los planificadores.

    Esta opción es muy importante si existe algún dominio y problema o planificador que se sabe que va a tardar mucho en ejecutarse, se deja para el final para ir obteniendo resultados previos del experimento.
  - **Almacenar o no los planes**: Con esta opción se indica al *Experimenter* si se desea o no almacenar los planes en formato XML de los problemas resueltos. Hay que tener en cuenta que esta opción ralentizará más el experimento si se elige almacenar los planes. Se aconseja elegir no almacenar planes si sólo se quiere obtener un estudio estadístico de cómo se comportan ciertos planificadores ante varios dominios y problemas.

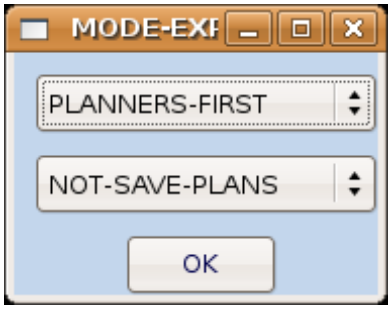

Figura 19: Interfaz para modo de ejecución en el Experimenter

- **Repeticiones**: Con este botón aparecerá la interfaz de la Figura 20 en la cual se puede indicar el número de veces que se quiere realizar el experimento. Esta opción se usará sobre todo en planificadores estocásticos como LPG, que pueden dar distintas soluciones en cada ejecución. Para el resto de planificadores, como se obtendrá la misma solución en las distintas ejecuciones no es necesario tener en cuenta esta opción.

Figura 20: Interfaz para número de repeticiones en el Experimenter

7.3. Área de dominios y problemas

Tal y como se ha comentado anteriormente, la zona inferior izquierda es la que se utilizará para añadir nuevos dominios al experimento. Esta zona no estará operativa hasta que no se haya creado un nuevo archivo de experimento o abierto uno ya existente.

Para añadir un nuevo dominio y problema al experimento se debe pulsar el botón *Nuevo Dominio y Problema*. Al pulsar este botón se abrirá una interfaz como la de la Figura 21, en esta interfaz se nos permite elegir un dominio, un problema y un archivo de reglas, el fichero de reglas se usará solo en los planificadores que lo soporten.

Se pueden añadir tantos dominios y problemas al experimento como se deseen sin limite alguno. Para añadir varios problemas del mismo dominio,

se deberá volver a elegir el dominio al añadir el nuevo problema como si fuera el primero que se añade.

Figura 21: Interfaz para añadir dominios y problemas

Para añadir un nuevo dominio y problema, sólo hay que pulsar el botón buscar que se encuentra al lado de cada campo de texto y seleccionar el dominio y problema que se desea resolver en el experimento. Sólo se permitirá elegir un problema si antes se ha elegido un dominio y si se pulsa el botón de añadir antes de haber especificado un dominio y problema se mostrará un mensaje de error y no se añadirá al conjunto de dominios del experimento.

Una vez se ha añadido un dominio y problema al experimento, se mostrará en la parte inferior izquierda de la pantalla a modo de árbol, con el path completo del dominio, el problema y el fichero de reglas, tal y como se puede observar en la Figura 22.

Si se quiere eliminar algún domino y su problema asociado del experimenter se debe pulsar el botón *Borrar Dominio y Problema*. Al pulsar este botón parecerá una interfaz como la de la Figura 23, en la cual se debe indicar la posición numérica del dominio y problema que se quiere eliminar del experimento tal y como se muestran en pantalla. Una vez elegido el dominio y problema a borrar se pulsa el botón *Borrar* y se actualizará automáticamente la parte de los dominios para mostrar el resto de dominios no eliminados.

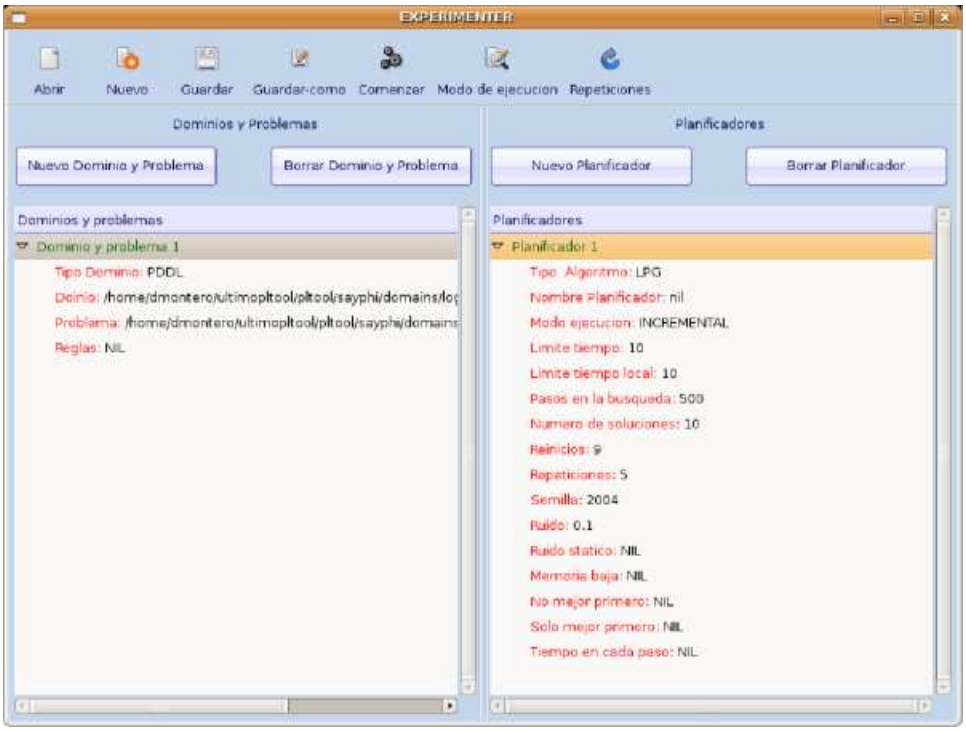

Figura 22: Interfaz experimenter con un dominio y un planificador añadidos

Figura 23: Interfaz para borrar un dominio y problema

7.4. Área de planificadores

La zona inferior derecha es la que se utilizará para añadir nuevos planificadores al experimento. Esta zona no estará operativa hasta que no se haya creado un nuevo archivo de experimento o abierto uno ya existente igual que en el caso anterior.

Para añadir un nuevo planificador al experimento se debe pulsar el botón *Nuevo Planificador*. Al pulsar este botón se nos abrira una interfaz como la de la Figura 24. En esta interfaz se nos permite elegir un planificador de entre

los que están añadidos en la herramienta: *IPSS, SAYPHI, LPG, SGLPAN y METRIC-FF*.

Cada uno de los planificadores tiene sus propias opciones tal y como se puede observar en las Figuras 24, 25, 26, 27 y 28. Debido a esto, cada uno tendrá su propia representación en esta parte una vez se añada el planificador al experimento. Para entender los parámetros de cada planificador ver capitulo 3.

Figura 24: Interfaz para añadir nuevo planificador al experimenter (IPSS)

Figura 25: Interfaz para añadir nuevo planificador al experimenter (SAYPHI)

Figura 26: Interfaz para añadir nuevo planificador al experimenter (METRIC-FF)

Figura 27: Interfaz para añadir nuevo planificador al experimenter (SGPLAN)

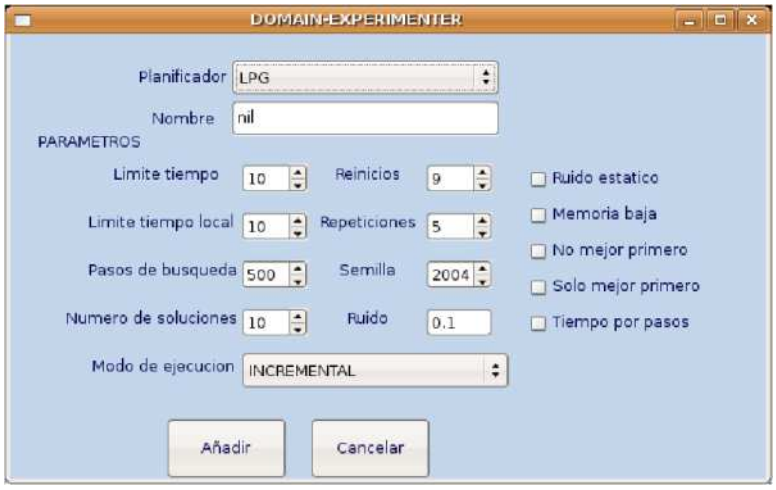

Figura 28: Interfaz para añadir nuevo planificador al experimenter (LPG)

El planificador será añadido a la parte inferior derecha con el nombre del planificador seguido del número de posición en la lista de planificadores añadidos o con el nombre que el usuario haya especificado al añadirlo. Este nombre no podrá contener espacios.

En cada planificador añadido se mostrarán en modo de árbol todas las opciones en concreto del planificador y el valor que toma cada una de ellas. Si la opcion es de tipo númerico se mostrará el valor al lado del nombre de la opción, si es de tipo String se mostrará la cadena de caracteres al lado de la opción y si es de tipo booleana se mostrará *T* si esta activa y *NIL* en caso contrario. Se puede observar un planificador ya añadido en la Figura 22.

Si se quiere eliminar algún planificador asociado del *Experimenter* se debe pulsar el botón *Borrar Planificador*. Al pulsar este botón parecerá una interfaz como la de la Figura 29, en la cual se debe indicar la posición numérica del planificador que se quiere eliminar del experimento tal y como se muestran en pantalla. Una vez elegido el planificador a borrar se pulsa el botón *Borrar* y se actualizará automáticamente la parte de los planificadores para mostrar el resto de planificadores no eliminados.

Figura 29: Interfaz para borrar un planificador

7.5. Obtener resultados del experimento

Dentro de todo el árbol de directorios por el que está compuesto *PLTool*, hay una carpeta que se corresponde exclusivamente para los ficheros generados por el *Experimenter*. Esta carpeta se encuentra situada dentro del directorio pltool/experimenter.

Cada vez que se pulsa el botón comenzar en la interfaz del *Experimenter* se creará una nueva carpeta en el directorio indicado arriba. El nombre de esta carpeta tendrá la siguiente sintaxis: "nombre del experimento"-"dia"-"mes"-"año"-"hora"-"minutos"-"segundos".

Dentro de esta carpeta se generará el archivo de resultados del experimento. Este archivo tendrá por nombre: "resultados"-"nombre del experimento".csv. En este archivo, se guardarán todos los resultados del experimento realizado. A continuación se indica cual es el formato exacto de este archivo.

Este archivo tendra una fila por cada pareja "dominio-problema"-"planificador". De modo que en esta linea se indiquen los resultados que se han obtenido para un dominio y problema con un planificador en concreto.

Cada una de estas filas tendrá un total de catorce valores separados por un espacio en blanco. Los datos que se indican en cada fila son:

- *Path* completo del dominio que se ha utilizado.
- *Path* completo del problema a solucionar.
- Nombre del planificador que ha intentado solucionarlo. En este caso el valor será o bien el nombre que le ha dado el usuario al añadir el planificador al *Experimenter* o el nombre del planificador y el número de posición dentro del experimento, por ejemplo "LPG1".
- Valores de las opciones del planificador. Para indicar las opciones del planificador se indicará el nombre de la opción seguido de dos puntos y el valor de la misma. Para separar todas las opciones se utilizará la barra vertical |.

- Tiempo que ha tardado el planificador en encontrar la solución al problema.
- Tiempo acumulado que lleva el planificador para resolver todos los problemas.
- Nodos totales expandidos por el planificador para resolver el problema.
- Nodos acumulados totales expandidos por el planificador para resolver todos los problemas.
- Coste del plan encontrado por el planificador para el problema actual.
- Coste acumulado de todos los planes que ha encontrado el planificador para el total de problemas.
- Longitud del plan encontrado por el planificador para el problema actual.
- Longitud acumulada de todos los planes que ha encontrado el planificador para el total de problemas.
- Si se ha resuelto el problema o no. Valores: si o no.
- Número de problemas totales que ha resuelto el planificador hasta el momento.

En caso de que el planificador no encuentre solución al problema o el planificador no soporte el dominio y problema indicado, se pondrá a valor "na" los valores de: tiempo, nodos expandidos, coste del plan y longitud del plan.

Si además se eligió la opción de almacenar planes, se almacenará un fichero XML en la carpeta de resultados del experimento para cada pareja "dominio-problema"-"planificador" que encuentre una solución al problema.

EL nombre de cada uno de estos archivos tendrá la siguiente sintaxis: "nombre del dominio"-"nombre del problema"-"nombre del planificador"-"numero de repeticion".xml. Con esto conseguimos que cada problema y planificador tenga un nombre propio y no se realicen sobreescrituras de ningún archivo a no ser que el usuario haya llamado a dos planificadores igual o haya incluido el mismo dominio y problema varias veces en el experimento.

8. Cómo definir dominios y problemas en Prodigy 4.0 y PDDL

En esta sección explicaremos cómo definir un dominio y problemas para dicho dominio para los lenguajes Prodigy 4.0 y PDDL.

8.1. Definición de dominios en Prodigy 4.0

En este apartado se va a explicar como definir correctamente un dominio en *Prodigy*.

Para la descripción de un dominio en Prodigy es necesario la especificación de tipos de elementos y operadores. Como ejemplo tomaremos el dominio *logistics*. En este dominio existen cuatro tipos de elementos: objetos, transportes, localizaciones y ciudades. Los transportes a su vez pueden ser: camiones o aviones. Las localizaciones dentro de la ciudad pueden ser: aeropuertos o oficinas postales.

Para definir los objetos en Prodigy se utiliza la siguiente sentencia: *(ptype-of NOMBRE :tipo-elemento)*

Si seguimos la sentencia anterior para definir los elementos del dominio *logistics* tendriamos que escribir:

```
(ptype-of OBJECT :top-type)
(ptype-of CARRIER :top-type)
(ptype-of TRUCK  CARRIER)
(ptype-of AIRPLANE CARRIER)
(ptype-of LOCATION :top-type)
(ptype-of AIRPORT LOCATION)
(ptype-of POST-OFFICE LOCATION)
(ptype-of CITY :top-type)
```

Si observamos los distintos tipos de elemento, existe el tipo *:top-type*, este tipo se corresponde con el tipo superior y de él dependeran aquellos elementos que no tengan ningún nivel superior, en nuestro ejemplo: *OBJECT, CARRIER, LOCATION y CITY*.

Una vez que se han definido los elementos del dominio debemos definir los operadores del mismo. Los operadores representan los posibles cambios que puede haber entre estados. Toda ejecución de un operador conlleva un cambio de estado.

El modo de definir los operadores es de la forma *SI condiciones ENTONCES acciones*. Las condiciones establecen cuando se puede ejecutar un operador, mientras que las acciones indican que acciones realizar sobre el

estado para transformarlo a uno nuevo una vez se haya aplicado el operador (añadiendo o eliminando hechos). En el ejemplo en el que nos encontramos existen varios operadores uno de ellos es *LOAD-TRUCK*. Este operador permite cargar un objeto en un camión. El modo de definirlo en Prodigy sería:

```
(OPERATOR LOAD-TRUCK
  (params <obj> <truck> <loc>)
  (preconds
   ((<obj> OBJECT)
    (<truck> TRUCK)
    (<loc>  (and LOCATION (in-truck-city-p  <truck> <loc>))))
   (and (at-obj <obj> <loc>)
(at-truck <truck> <loc>)))
  (effects
   ()
   ((add (inside-truck <obj> <truck>))
    (del (at-obj <obj> <loc>)))))
```

Lo primero de todo al definir un operador es indicar los parametros del mismo. En nuestro caso tenemos tres parámetros implicados: $< obj >$, $< truck >$ y $< loc >$.

Una vez tenemos los parametros definidos pasamos a las precondiciones que deben cumplir dichos parámetros. El modo de indicar que se comienzan a describir las precondiciones es con la palabra *preconds*. En nuestro caso y por orden de definición en el operador tenemos:

- **<obj> OBJECT**: el parámetro $< obj >$ debe ser del tipo OBJECT.
- **<truck> TRUCK**: el parámetro $< truck >$ debe ser del tipo TRUCK.
- **<loc> (and LOCATION (in-truck-city-p <truck> <loc>))**: el parámetro $< loc >$ debe ser del tipo LOCATION y el camión $< truck >$ debe estar en dicha localización.
- **at-obj <obj> <loc>**: el objeto $< obje >$ debe estar en la localización $< loc >$.
- **at-truck <truck> <loc>**: el camión $< truck >$ debe estar en la localización $< loc >$.

Una vez terminadas de definir las precondiciones se definen los efectos del operador. Estos efectos comienzan a definirse con la palabra *efects*. Dentro de los efectos existen dos tipos de sentencias, la sentencia *add*: para añadir nuevos hechos al estado y la sentencia *del*: para eliminar hechos del estado. En los efectos de este operador se puede observar:

- **add (inside-truck <obj> <truck>)**: Se añade el hecho de que el objeto $< obj >$ se encuentra dentro del camión $< truck >$.
- **del (at-obj <obj> <loc>)**: Se elimina de la localizacion $< loc >$ el objeto $< obj >$.

Como se puede observar, en los operadores aparecen una serie de predicados *at-obj, inside-truck....* Para cada uno de estos predicados es el usuario el que debe tener muy claro lo que significa cada uno de ellos. No existe un modo de definición especifica de predicados en *Prodigy 4.0*, lo que los predicados nos indican es como se encuentra el problema en un momento determinado. Más adelante veremos como al definir un problema en realidad estamos definiendo unos predicados para el estado inicial y unos predicados que queremos que se cumplan en el estado final deseado.

Se puede consultar la totalidad del dominio *logistics*, en la carpeta *prodigy/domains/logistics*, el fichero tiene por nombre *domain.lisp*.

8.2. Definición de problemas de planificación en Prodigy 4.0

Los problemas en *Prodigy* se especifican como un par *(estado-inicial, estado-final)* y lo que queremos con la planificación es encontrar la sucesión de operadores que debemos aplicar para llegar del estado inicial al final.

Para definir un problema hay que:

- Definir el nombre del problema.
- A continuación se declaran los tipos de objetos que pueden intervenir.
- Definición del estado inicial.
- Definición del estado final al que se quiere llegar.

Con el fin de que se entienda mejor la definición de problemas continuaremos con nuestro ejemplo en el dominio *logistics* y definiremos un problema para dicho dominio.

Supongamos que definimos un problema en el que en la situación inicial existe un objeto (*m1*) que se quiere transportar desde una agencia de transporte (*agencia1*) a otra agencia de transporte (*agencia2*) mediante la utilización de un camión (*truck1*). Además, todos estos elementos se encuentran en la localización (*París*).

Para comenzar a definir estos objetos se comienza con la palabra *objects* y se continúa como se muestra a continuación:

```
(objects
        (m1 OBJECT)
        (Paris CITY)
        (camion1 TRUCK)
        (agencia1 agencia2 LOCATION)
        )
```

Una vez tenemos definidos todos los objetos de nuestro problema comenzamos a definir el estado inicial del mismo. Para definir el estado inicial del problema se comienza con la palabra *state*. Lo que sabemos de nuestro estado inicial es:

- Todos los objetos se encuentran en la localización (*París*).
- El objeto (*m1*) se encuentra en la (*agencia1*).
- El camión (*camion1*) se encuentra en la (*agencia1*).

Estos tres puntos se definen en el lenguaje *Prodigy* como:

```
  (state
        (and
         (at-truck camion1 Paris)
         (at-obj m1 Paris)
         (loc-at agencia1 Paris)
         (loc-at agencia2 Paris)
         (at-obj m1 agencia1)
         (at-truck camion1 agencia1)
         (part-of camion1 Paris)
         )
)
```

Como se comentó antes el cometido es llevar el objeto *m1* a la agencia de transporte *agencia2*. El estado final del problema se comienza a describir con la palabra *goal* y se especifica como se muestra a continuación:

```
 (goal
   (and
     (at-obj m1 agencia2)
    )
 )
```

A continuación mostramos como quedaría este problema al completo:

```
(setf (current-problem)
      (create-problem
       (name problema1)
       (objects
        (m1 OBJECT)
        (Paris CITY)
        (camion1 TRUCK)
        (agencia1 agencia2 LOCATION)
        )

       (state
        (and
         (at-truck camion1 Paris)
         (at-obj m1 Paris)
         (loc-at agencia1 Paris)
         (loc-at agencia2 Paris)
         (at-obj m1 agencia1)
         (at-truck camion1 agencia1)
         (part-of camion1 Paris)
         )
        )
       (goal
        (and
         (at-obj m1 agencia2)
        )
       )
    )
)
```

8.3. Definición de dominios en PDDL

En este apartado se va a explicar como definir correctamente un dominio en *PDDL*.

Para la descripción de un dominio en PDDL genérico es necesario la especificación de tipos de elementos, predicados y operadores. Como ejemplo tomaremos el dominio *logistics* igual que en el caso anterior. En este dominio existen cuatro tipos de elementos: objetos, transportes, lugares y ciudades. Los transportes a su vez pueden ser: camiones o aviones. Los lugares dentro de la ciudad pueden ser: aeropuertos u oficinas postales.

Para definir los objetos en PDDL se utiliza la siguiente sentencia: *(:types Nombre - tipo ....)*

Si seguimos la sentencia anterior para definir los elementos del dominio *logistics* tendriamos que escribir:

```
(:types truck airplane - vehicle
package vehicle - physobj
airport location - place
city place physobj - object)
```

Si observamos los distintos tipos de elemento, existe el tipo *:object*, este tipo se corresponde con el tipo superior y de él dependeran aquellos elementos que no tengan ningún nivel superior, en nuestro ejemplo: *city, place y physobj.*

Del resto de elementos tenemos que: *airport y location* son de tipo *place*, *package y vehicle* son de tipo *physobj* (objeto físico) y *truck y airplane* son de tipo *vehicle.*

Una vez que se han definido los elementos del dominio debemos definir los predicados que existen en el mismo. Para el dominio *logistics* tenemos un total de tres predicados:

- En qué ciudad se encuentra un lugar.
- En qué lugar se encuentra un objeto físico.
- En qué vehiculo se encuentra un paquete.

Vamos a definir estos tres predicados en lenguaje PDDL, la sentencia para definir predicados es: *(:predicates (predicado objeto1 ... objetoN) ....).* Siguiendo esta sentencia la definición quedaría:

```
(:predicates  (in-city ?loc - place ?city - city)
(at ?obj - physobj ?loc - place)
(in ?pkg - package ?veh - vehicle))
```

Una vez que se han definido los predicados se deben definir los operadores. El modo de definir los operadores es muy parecido a *Prodigy*: se deben definir los parámetros necesarios en el operador, las precondiciones para utilizar el operador y los efectos que se producirían sobre el estado actual. Con el fin de seguir el mismo ejemplo que en el caso de *Prodigy*, vamos a definir el operador *LOAD-TRUCK* en *PDDL*:

```
(:action LOAD-TRUCK
  :parameters   (?pkg - package ?truck - truck ?loc - place)
  :precondition (and (at ?truck ?loc) (at ?pkg ?loc))
  :efect  (and (not (at ?pkg ?loc)) (in ?pkg ?truck)))
```

Lo primero de todo al definir un operador es indicar los parámetros del mismo, se comienza con la palabra *:parameters*. En nuestro caso tenemos tres parámetros implicados: ?pkg, ?truck y ?loc. Además deberemos indicar con que tipo de objeto se corresponde cada uno de los parámetros.

Una vez tenemos los parámetros definidos pasamos a las precondiciones que deben cumplir dichos parámetros. Para comenzar a definir las precondiciones se comienza con la palabra *:preconditions*. En nuestro caso y por orden de definición en el operador tenemos:

- **(at ?truck ?loc)**: el camión "truck" debe estar en el lugar "loc".
- **(at ?pkg ?loc)**: el paquete "pkg" debe estar en el lugar "loc".

Una vez terminadas de definir las precondiciones se definen los efectos del operador. Estos efectos comienzan a definirse con la palabra *effects*. Dentro de los efectos se debe indicar que predicados no deben estar ya en el estado y cuales se añaden al mismo. En los efectos de este operador se puede observar:

- **(not (at ?pkg ?loc))**: Eliminar del estado el predicado que nos indica que el paquete "pkg" se encuentra en el lugar "loc".
- **(in ?pkg ?truck)**: Se añade el predicado que nos indica que el paquete "pkg" se encuentra en el camión "truck".

Con esto y otros operadores más tendriamos definido nuestro dominio *logistics* en el lenguaje PDDL.

Se puede consultar la totalidad del dominio *logistics* en el lenguaje PDDL, en la carpeta *sayphi/domains/logistics*, el fichero tiene por nombre *domain.pddl*.

8.4. Definición de problemas de planificación en PDDL

Los problemas en *PDDL* se especifican como un par *(estado-inicial, estado-final)* igual que en *Prodigy*.

Para definir un problema en PDDL hay que:

- Definir el nombre del problema y decir a qué dominio se refiere.
- Declarar los tipos de objetos que pueden intervenir.
- Definición del estado inicial.
- Definición del estado final al que se quiere llegar.

Con el fin de que se entienda mejor la definición de problemas continuaremos con nuestro ejemplo en el dominio *logistics* y definiremos un problema para dicho dominio.

Partimos del mismo ejemplo que en el caso de *Prodigy*, en la situación inicial existe un objeto (*m1*) que se quiere transportar desde una agencia de transporte (*agencia1*) a otra agencia de transporte (*agencia2*) mediante la utilización de un camión (*truck1*). Además, todos estos elementos se encuentran en la localización (*París*).

Para comenzar a definir estos objetos se comienza con la palabra *:objects* y se continúa como se muestra a continuación:

```
(:objects
        m1 - package
        Paris - city
        camion1 - truck
        agencia1 agencia2 - location)
```

Una vez tenemos definidos todos los objetos de nuestro problema comenzamos a definir el estado inicial del mismo. Para definir el estado inicial del problema se comienza con la palabra *:init*. Lo que sabemos de nuestro estado inicial es:

- Todos los objetos se encuentran en la localización (*París*).
- El objeto (*m1*) se encuentra en la (*agencia1*).
- El camión (*camion1*) se encuentra en la (*agencia1*).

Estos tres puntos se definen en el lenguaje *PDDL* como:

```
  (:init
         (at camion1 agencia1)
         (at m1 agencia1)
         (in-city agencia1 Paris)
    (in city agencia2 Paris))
```

Como se comentó antes el cometido es llevar el objeto *m1* a la agencia de transporte *agencia2*. El estado final del problema se comienza a describir con la palabra *:goal* y se especifica como se muestra a continuación:

```
 (:goal
   (and
     (at m1 agencia2)))
```

A continuación mostramos como quedaría este problema al completo:

```
(define (problem logistics1)
      (:domain logistics)

      (:objects
        m1 - package
        Paris - city
        camion1 - truck
        agencia1 agencia2 - location)

      (:init
          (at camion1 agencia1)
          (at m1 agencia1)
          (in-city agencia1 Paris)
     (in city agencia2 Paris))

   (:goal (and
      (at m1 agencia2)))
)
```

Referencias

[Aler Mur, 2006] Ricardo Aler Mur (2006) *Tutorial de Prodigy4.0* http://scalab.uc3m.es/∼docweb/iasup/practicas/anteriores/prodigy.html Universidad Carlos III de Madrid

[Veloso et al, 1995] Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; Blythe, J. (1995) *Integrating Planning and Learning: the PRODIGY Architecture*

[Veloso et al, 1996] Veloso, M.; Borrajo, D. (1996) *Lazy Incrementar Learning of Control Knowledge for Efficienly Obtaining Quality Plans*

[Stefan Edelkamp, Jörg Hoffman, 2004] Stefan Edelkamp, Jörg Hoffman (2004) *PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition* http://ls5-web.cs.unidortmund.de/ edelkamp/ipc-4/DOCS/pddl2.2.ps.gz

[Jörg Hoffmann] *Página web de Metric-FF* http://members.deri.at/ joergh/metric-ff.html

[Alfonso Gerevini, Ivan Serina] *Página web de LPG-TD* http://zeus.ing.unibs.it/lpg/

[Chih-Wei Hsu, Benjamin W. Wah, Ruoyun Huang, Yixin Chen] *Página web de SGPLAN* http://manip.crhc.uiuc.edu/programs/SGPlan/index.html