SPLIN: Una herramienta para el diseño, ejecución y evaluación de problemas de planificación[*]

Laura Sebastiá Tarín, Eliseo J. Marzal Calatayud

Dpto. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n 46022 VALENCIA
{lstarin,emarzal}@dsic.upv.es

Resumen. Este trabajo presenta el diseño e implementación de un Sistema de PLanificación INteligente (SPLIN) que se compone de dos módulos principales: un compilador para el lenguaje de planificación y un Planificador de Orden Parcial (POP). SPLIN está integramente desarrollado en C, dispone de un lenguaje de planificación de fácil utilización e incorpora la técnica de menor compromiso (*least commitment*) [Weld94] tanto en la ordenación de pasos durante el desarrollo del plan como en el tratamiento de variables. Este último aspecto permite obtener resultados altamente satisfactorios frente a otros planificadores de orden parcial. La versión de SPLIN que se presenta en este artículo es un primer prototipo que se ha realizado en el grupo GTI-IA de la Universidad Politécnica de Valencia.

1. Introducción

El planificador UCPOP [Penberthy92] es uno de los primeros prototipos que pone de manifiesto los beneficios que pueden obtenerse de una planificación de orden parcial [Barrett94]. Recientes trabajos [Gerevini96, Pollack97] muestran la importancia de emplear estrategias de búsqueda idóneas que permitan reducir la complejidad inherente a todo problema de planificación y obtener así el máximo beneficio fr la utilización de un planificador de orden parcial.

Sin embargo, el rendimiento idóneo de un POP no sólo se debe a la adecuada selección de estrategias de búsqueda para la resolución de un problema particular. Son muchos los factores que intervienen directa o indirectamente en la medida de eficiencia de resolución de un POP; entre ellos se encuentra la necesidad de un lenguaje suficientemente expresivo, sencillo y de fácil utilización que permita representar diferentes tipos de conocimiento, que posteriormente puedan ser gestionados eficientemente por el planificador. Otro de los aspectos que influyen considerablemente en el rendimiento de un POP es la gestión que se realiza en el manejo de las restricciones, lo cual obliga en muchos casos a incorporar métodos y técnicas de *Constraint Satisfaction Problems* (CSP).

SPLIN es un primer prototipo de una herramienta de planificación de fácil uso, flexible y que permite especificar fácilmente distintos dominios de aplicación. SPLIN incorpora la estrategia de menor compromiso en todas las gestiones y operaciones que se realizan en el tratamiento de variables. Estas funcionalidades permiten obtener resultados satisfactorios, los cuales se contrastan con los obtenidos por el planificador UCPOP [Penberthy92, Barrett95].

[*] Este trabajo ha sido subvencionado parcialmente por el proyecto CICYT TAP98-0345.

2. Componentes básicas de SPLIN

SPLIN es un planificador regresivo de orden parcial que realiza una búsqueda en un espacio de planes. En este caso se parte de un plan incompleto denominado ***plan parcial***, y se consideran todas las distintas formas de expandir este plan hasta conseguir un plan completo que resuelva el problema.

Hay dos elementos básicos en SPLIN, el *compilador* del lenguaje de planificación y el núcleo principal o *POP*. SPLIN dispone de una interfaz de usuario a través de un lenguaje de planificación propio que permite la generación de código intermedio, el cual posteriormente será utilizado por el POP. Este código intermedio contiene, principalmente, las estructuras de datos que se emplearán para gestionar los objetos creados a partir del dominio del problema.

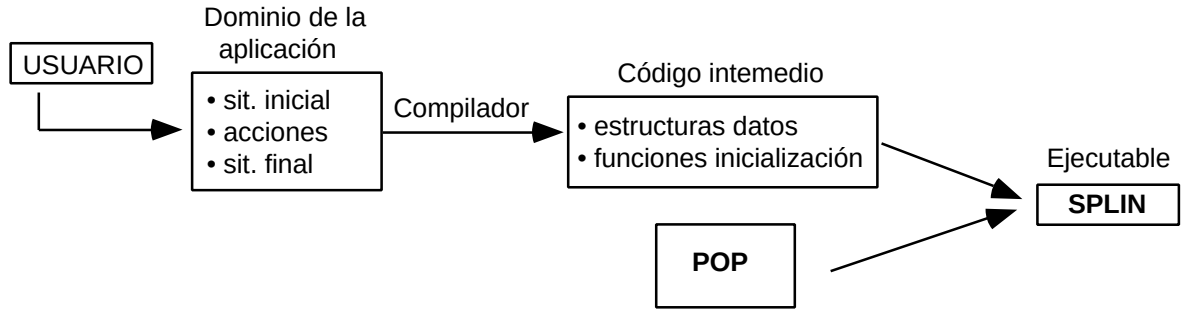

Figura 1. Componentes de SPLIN

El POP emplea un algoritmo básico de planificación de orden parcial que garantiza la completitud y correctitud del proceso de planificación. Dependiendo de la estrategia de búsqueda que se utilice en el proceso se podrá, adicionalmente, garantizar la obtención de la solución óptima.

3. Lenguaje de planificación

El lenguaje de planificación de SPLIN, basado en la aproximación clásica del lenguaje STRIPS [Fikes71], se emplea para especificar los dominios de los problemas. La descripción de un *estado del problema* se compone de un conjunto de literales positivos (se asume que los literales que no se mencionan explícitamente en la descripción del estado son falsos) y el *objetivo del problema* se describe igualmente mediante una conjunción de literales positivos.

A diferencia de la mayoría de los esquemas de representación empleados en planificación, los literales en SPLIN mantienen una estructura uniforme que se compone de cuatro elementos: `<clase objeto atributo valor>`. El lenguaje estructurado de SPLIN facilita la especificación del dominio de un problema al proporcionar un esquema de representación más natural e intuitivo. Para realizar la descripción de un problema es necesario definir los siguientes elementos:

- *clases*, donde se definen las propiedades o atributos que caracterizan a la clase. SPLIN acepta la especificación de slots no temporales y temporales (estos últimos se emplearán en una posterior versión del planificador donde se incluirá un módulo de razonamiento

temporal). Por el momento todos los atributos de una clase se especificarán como no-temporales.

• *objetos*, que representan cada una de las instancias de una clase determinada.

La sintaxis del lenguaje SPLIN guarda una estrecha relación con el lenguaje CLIPS [CLIPS97]. Por ejemplo, para representar el escenario de la anomalía de Sussman en el "mundo de bloques" se definirían los siguientes elementos:

```
(defclass BLOCK (is-a user)
    (non-temporal
        (slot ON (type BLOCK))
        (slot STATUS (type STRING))))

(defobject Table (is-a BLOCK))

(defobject BlockA (is-a BLOCK)
    (ON Table))

(defobject BloqueB (is-a BLOCK)
    (ON Table)
    (STATUS "libre"))

(defobject BloqueC (is-a BLOCK)
    (ON BlockA)
    (STATUS "libre"))

(defobjective GOAL
    (BLOCK BlockA on BlockB)
    (BLOCK BlockB on BlockC))
```

La definición de objetos junto con los valores asociados a sus slots configuran los literales que representan la situación inicial del problema. La tarea del compilador consiste en generar el código intermedio (Figura 1), donde se incluirán las funciones encargadas de crear la situación inicial, el objetivo, las clases y sus slots, los objetos con sus valores asociados y las acciones. Cuando SPLIN se ejecuta se invocan dichas funciones y se generan todos los elementos necesarios para realizar el proceso de planificación. En el ejemplo de la anomalía de Sussman, la situación inicial vendría representadas por los siguientes literales:

```
(BLOCK BlockA ON Table)
(BLOCK BlockB STATUS "libre")
(BLOCK BlockB ON Table)
(BLOCK BlockC ON BlockA)
(BLOCK BlockC STATUS "libre")
```

La representación de los operadores o acciones consta de los siguientes componentes:

• *nombre del operador y parámetros* de la acción: los parámetros del operador son constantes y variables que aparecen en las descripciones de las precondiciones y efectos del operador

• *precondiciones*: es una conjunción de literales positivos que determinan lo que debe ser cierto en el momento de aplicación del operador.

• *tests condicionales*: SPLIN admite la inclusión de un conjunto de tests condicionales sobre los valores que pueden asumir las variables. Estos tests se corresponden con las

restricciones de igualdad y desigualdad (restricciones de *codesignation* y *noncodesignation*) que se emplean en otros planificadores.

Se puede especificar un test entre una variable y una constante o entre dos variables. Las funciones de evaluación que se pueden definir son: `<`, `<=`, `>`, `>=`, `=`, `<>`. Un test actúa como un filtro para determinar la veracidad de las condiciones de la acción. Si todos los tests condicionales de la instancia de un operador se satisfacen entonces se podrá aplicar dicha acción.

• *efectos*: es una conjunción de literales, positivos y negativos, que determinan lo que será cierto tras la aplicación del operador. La especificación de los literales que representan los efectos en SPLIN se pueden dividir en efectos ***primarios*** y efectos ***secundarios o consecuencias***. Esta clasificación permitirá, en futuras versiones de SPLIN, incorporar un módulo de jerarquías de abstracción [Knoblock94], para lo cual resulta conveniente distinguir entre los efectos primarios y secundarios de una acción a efectos de conseguir un mayor grado de refinamiento en la jerarquía del problema. Los efectos primarios constituyen los objetivos principales que se consiguen con la aplicación de la acción; los efectos secundarios o consecuencias son aquellos cambios indirectos que se consiguen con dicha acción. Los efectos negativos de una acción se especifican como consecuencias mientras que los efectos positivos pueden describirse como primarios o consecuencias. En la actualidad se puede especificar todos los efectos positivos de una acción como primarios o como primarios/consecuencias, ya que SPLIN realizará un tratamiento similar de ambos tipos de efectos.

La acción "*desapilar un bloque x de un bloque y*" en el dominio del mundo de bloques se especificaría del siguiente modo:

```
(defaction DESAPILAR (?x ?y)
    (preconds
        (BLOCK ?x ON ?y))
        (BLOCK ?x STATUS "libre"))
    (test (<> ?y Table))
    (effects
        (primary
            (BLOCK ?x STATUS "cogido")
            (BLOCK ?y STATUS "libre"))
        (consequences
            (add)
            (delete
                (BLOCK ?x STATUS "libre")
                (BLOCK ?x ON ?y)))))
```

4. Diseño de SPLIN

El núcleo principal de SPLIN lo constituye un Planificador regresivo de Orden Parcial (POP) [Weld 94, Yang97]. El POP construye un plan de orden parcial añadiendo incrementalmente todos los componentes del plan. En cada iteración del POP se selecciona una precondición de un paso y se identifican los pasos del plan y los operadores que pueden resolver dicha precondición. La precondición se resuelve entonces estableciendo un nuevo enlace causal en el plan. A continuación se buscan los pasos que amenazan dicho enlace causal así como los enlaces causales que pueden verse amenazados por la adición del nuevo paso. Este proceso se repite hasta que todas las precondiciones de cada paso del plan tienen asociado un enlace causal y todas las amenazas del plan se han eliminado.

4.1 Características generales del POP

Las principales características del planificador SPLIN son:

a) *Proceso de Unificación.* En SPLIN las variables sólo pueden aparecer en el campo 'objeto' o en el campo 'valor' de un literal. Cada una de las variables puede tomar valores dentro de un rango definido para su clase o según el tipo de atributo. Existen dos formas de asignar dominios iniciales a las variables dependiendo del tipo de éstas. Cuando la variable es de tipo *clase*, su dominio consiste en todos los objetos pertenecientes a la clase. Si la variable pertenece a un tipo estándar (entero, real, boolean, string) el dominio consiste en dos valores, límite inferior y límite superior que representan el rango de posibles valores. La determinación, a priori, de los valores de los dominios para los tipos estándar es uno de los principales problemas, ya que dichos dominios son infinitos.

El proceso de unificación en SPLIN es bastante sencillo debido a la uniformidad de la estructura que se utiliza para los efectos y precondiciones (literales). Dos literales pueden 'descartarse' rápidamente si sus campos 'clase' o 'slot' no coinciden. El proceso de unificación propiamente dicho sólo se aplica a los campos 'objeto' y 'valor' de los dos literales, ya que son los únicos que puden contener variables. El proceso de unificación de los campos 'valores' de dos literales se especifica del siguiente modo:

- Dos referencias constantes a elementos pertenecientes a una clase definida por el usuario unifican si referencian al mismo objeto.
- Dos tipos estándar unifican si coinciden los valores
- Una referencia a un objeto y una variable unifican si el objeto pertenece al dominio de la variable. En caso de unificar se restringe el dominio de la variable al objeto.
- Un tipo estándar y una variable unifican si el valor del tipo está en el rango de valores del dominio. En caso de unificar se restringe el dominio de la variable al valor.
- Dos variables unifican si la intersección de los dominios no es vacía. En caso que las variables unifiquen, el dominio de ambas variables será el conjunto intersección.

b) Utilización del Criterio de menor compromiso [Chapman87, Weld94]. Esta estrategia es utilizada en la mayoría de los planificadores de orden parcial como mecanismo para retrasar la decisión sobre la ordenación de pasos del plan. SPLIN extiende la aplicación de esta estrategia al tratamiento de variables, permitiendo así que cada variable tenga asociado un conjunto de posibles valores que se pueden ligar a la misma. La utilización de este criterio en el tratamiento de variables se aplica en cuatro situaciones:

b.1) en la resolución de una precondición `p` mediante la adición de un nuevo paso `s` (planificación con acciones parcialmente instanciadas [Weld94]). En este caso, la acción que representa el paso `s` puede contener una o varias variables cuyos valores aún no han sido determinados. Las decisiones siguientes podrán añadir más restricciones sobre el valor de dichas variables, hasta que éstas tomen un único valor. Este tipo de restricciones se refieren básicamente a restricciones de igualdad y desigualdad (*codesignation* y *noncodesignation*).

b.2) en la detección y resolución de amenazas. Dado que las variables tienen ahora asociado un conjunto de posibles valores, las amenazas se dividen en dos categorías: amenazas *necesaria*s (cuando las variables del enlace causal y del paso están completamente instanciadas) y amenazas *posibles*. Al igual que UCPOP 4.0

[Barrett95], SPLIN retrasa la resolución de las amenazas posibles hasta el final [Peot93], ya que durante el proceso de planificación muchas de estas amenazas desaparecen y otras se pueden convertir en amenazas necesarias a medida que se restringen los valores del dominio de las variables.

b.3) en el proceso de unificación. Si una precondición p_i que contiene una variable `?x` puede unificar con `n` posibles literales de un paso (asociando `n` valores distintos a la variable), SPLIN genera un único nodo, asignando los `n` valores al dominio de la variable `?x`.

b.4) en el proceso de resolución de tests condicionales. Otra posible aplicación de la estrategia de menor compromiso hace referencia a la resolución de las restricciones de igualdad y desigualdad de los operadores. Esta funcionalidad se detalla en el siguiente apartado.

c) *Tests condicionales.* Otra de las características que conviene resaltar del planificador es el procesamiento de los tests condicionales y la utilización que se hace de los mismos. Los tests se emplean en SPLIN como parte del propio proceso de planificación para retringir los valores de las variables. Por ejemplo, un test del tipo `(test(= ?x ?y))` obliga a unificar las variables `?x` e `?y` para que éstas tengan asociado un mismo dominio. Asumiendo que `?x={BloqueA}` e `?y={BloqueA, BloqueB}`, el test se satisface tras unificar ambas variables al mismo dominio `{BloqueA}`, lo que determina que se eliminará el valor `{BloqueB}` del dominio de la variable `?y`. Si la intersección de valores del dominio de las variables `?x` e `?y` es el conjunto vacío entonces el mencionado test falla y se poda la rama del árbol de búsqueda correspondiente al plan bajo estudio.

Una de las aportaciones de SPLIN en el tratamiento de variables y test condicionales consiste en "resolver" los tests aunque éstos no permitan a priori determinar el valor concreto que deben asumir las variables. En lugar de generar un plan para cada posible combinación de valores de dos variables involucradas en un test, SPLIN aplica una estrategia de menor compromiso que permite seguir asociando un conjunto de valores a las variables a la vez que se resuelve el test. Sea por ejemplo`(test (<> ?x ?y))`, y los valores asociados a las variables `?x={BloqueA, BloqueB, BloqueD}`, `?y={BloqueB, BloqueC}`. En la resolución del test, SPLIN generará dos planes sucesores; en el primero se elimina el conjunto conflicto de valores (`BloqueB`) del dominio de la primera variable, y en el segundo hijo, se elimina el valor conflicto del dominio de la segunda variable. De este modo, los valores asociados a cada uno de los planes sucesores serían:

```
P1: ?x={BloqueA, BloqueD}, ?y={BloqueB, BloqueC}
P2: ?x={BloqueA,BloqueB,BloqueD}, ?y={BloqueC}.
```

Esta estrategia evita generar tantos nodos como posibles combinaciones se puedan formar con los valores de las variables en el árbol de búsqueda actual (en el ejemplo propuesto serían 5), y permite ir restringiendo los dominios de las variables a medida que prosigue el proceso de planificación. Otra de las ventajas adicionales de la aplicación de esta estrategia en la resolución de tests es que el número de puntos muertos que aparecen en el árbol de búsqueda es menor, o al menos éstos se encuentran antes, evitando así la exploración de ramas que no conducen a una solución. Otra ventaja, respecto a los planificadores que emplean algoritmos CSP para la resolución de restricciones de asignación, es que en el

caso de SPLIN, una vez resuelto un test en un plan no se ha de considerar nuevamente en ninguno de los nodos sucesores de dicho plan.

4.2 Búsqueda en un espacio de planes

El proceso de obtención del plan que resuelve un determinado problema se puede ver como un proceso de búsqueda en un espacio de planes. El nodo raíz del árbol se corresponde con el plan inicial p_0; en cada paso del proceso de búsqueda se generan todos los sucesores de un nodo dado n_i, los cuales representan las distintas formas de resolver una precondición p, o una amenaza a, de un paso s del plan p_i que representa el nodo n_i. La búsqueda tendrá éxito si se encuentra un nodo que contenga un plan correcto. El problema del control de la búsqueda ocurre en dos niveles:

a) En primer lugar hay que decidir el nodo, entre el conjunto de nodos frontera, que se selecciona para su expansión. A este problema se denomina 'problema de la selección del plan'.

b) El segundo problema se define como el problema de seleccionar el siguiente 'objetivo' a estudiar, es decir, la siguiente precondición o amenaza que se tratará de resolver del plan seleccionado en el nivel anterior.

Selección de Planes. La estrategia que se sigue habitualmente es la de ordenar los nodos frontera del árbol de búsqueda por menor coste del plan representado en cada nodo. Para determinar el mejor nodo a expandir se aplica una función de evaluación a cada nodo (f(n)=g(n)+h(n)), donde g(n) representa el coste de la solución desde el nodo raíz hasta el plan representado en el nodo n, y h(n) es la estimación del coste de la solución desde el nodo n hasta el objetivo. El número de pasos (S) de un plan de un nodo n se puede considerar como una medida del factor g(n), y el número de precondiciones sin resolver (OC -open conditions-) y de amenazas pendientes del plan (UC -unsafe conditions-) se pueden considerar como medidas estimativas del coste de la solución restante, es decir como medidas del factor h(n) [Gerevini96]. Esta es la combinación que emplea UCPOP por defecto; sin embargo en [Gerevini96] se argumenta en favor de utilizar f(n)=S(n)+OC(n), omitiendo el número de amenazas existentes.

Selección de Objetivos. A lo largo de la literatura sobre planificación se han propuesto distintas estrategias para la selección del objetivo de un plan. Entre las más relevantes destaca la estrategia *Delay Unforced Threats* -DUnf- [Peot93], la estrategia *Least Cost Flaw Repair* -LCFR- [Joslin94] o la propuesta ZLIFO [Gerevini96]. En concreto, el planificador UCPOP permite incorporar la estrategia ZLIFO, cuyos resultados mejoran notablemente la estrategia LIFO utilizada en UCPOP por defecto. La estrategia ZLIFO ha demostrado ser bastante eficiente para diferentes dominios de problemas.

5. Pruebas de evaluación

En este apartado se muestran algunos resultados obtenidos con SPLIN y se comparan con los obtenidos con la versión 4.0 de UCPOP [Barrett95]. Para realizar la comparativa se han analizado únicamente el número de nodos creados (NC) y el número de nodos expandidos (NE), esto es, el número total de planes creados/explorados en cada prueba. El número de

planes es una medida del tamaño del espacio de búsqueda requerido por el planificador, y depende, fundamentalmente, de las características del lenguaje de representación y del algoritmo de búsqueda empleado, pero no de la implementación.

Para conseguir una homogeneización en las pruebas realizadas, se ejecutaron distintos problemas en UCPOP y SPLIN bajo los siguientes criterios:

a) Se ha empleado en todas las pruebas `S+OC` (número de pasos + precondiciones sin resolver) como estrategia para la selección de planes.
b) Se ha empleado en todas las pruebas la propuesta ZLIFO como estrategia para la selección de objetivos.
c) Se ha empleado en todas las pruebas la estrategia de búsqueda Best First Search (función **`bestf-search`** en UCPOP).

La tabla que se presenta a continuación muestra los resultados obtenidos para distintos dominios UCPOP.

	UCPOP 4.0		**SPLIN**	
	NG	NE	NG	NE
PROB. 1	47	28	33	24
PROB. 2	1128	780	237	195
PROB. 3 [1]	641	420	89	69
PROB. 4	107	71	69	57
PROB. 5	545	443	145	126
PROB. 6	31	17	271	151
PROB. 7	39	22	102	55
PROB. 8	96	57	76	52
PROB. 9	1092	742	137	96

PROB.1 se corresponde con el problema de las torres de Hanoi con un único operador y dos discos, PROB. 2 para tres discos y PROB. 3 para tres operadores y tres discos. La representación de SPLIN con un único operador (Apéndice A) es prácticamente equivalente a la de UCPOP. Se ha empleado una única clase (DISK) y se han incorporado los tests que comprueban que el disco a mover no es un objeto ficticio para equiparar las representaciones. El resultado para PROB. 1 es muy similar, mientras que para PROB. 2 las diferencias son considerables. Tras un análisis detallado de las ejecuciones de ambos planificadores, se ha comprobado que los buenos resultados obtenidos por SPLIN se deben a la utilización de la estrategia de menor compromiso en el proceso de unificación y resolución de tests condicionales. En UCPOP se genera un plan nuevo por cada valor que puede asumir la variable `?disk` del `operador MOVE-DISK`, lo que provoca que, para muchos nodos del árbol, se generen tres ramificaciones adicionales.

Los resultados de PROB. 3 se han extraído de [Gerevini96], donde los experimentos se han realizado con UCPOP 2.0. Esta es una versión de Hanoi con 3 operadores realizada por los autores del mencionado artículo y que no está disponible en UCPOP 4.0. En nuestro caso, se ha diseñado una versión similar a la que se puede encontrar en [Knoblock94].

[1] Resultados obtenidos con UCPOP 2.0

PROB. 4 corresponde al dominio *Ferry* y PROB. 5 al dominio *Fridge* (problema *Fixa*). El test del ferry es un problema simple que requiere mover dos coches de una localización A a otra localización B, utilizando para ello un único barco que puede realizar tres movimientos, cargar, navegar y desembarcar. *Fixa* es un problema donde se requiere cambiar el compresor de un frigorífico, teniendo que quitar los tornillos del panel, parar el frigorífico, quitar el panel trasero y cambiarlo.

PROB. 6 y PROB. 7 son dos problemas asociados al mundo de bloques. El primero corresponde con la anomalía de *Sussman* y el segundo con el problema *Tower-Invert3*. La diferencia de resultados se debe a que UCPOP implementa dicho dominio con un único operador, gracias a la utilización de efectos condicionales. Dado que SPLIN no dispone aún de esta funcionalidad, el dominio del mundo de bloques se ha representado con los cuatro clásicos operadores "coger un bloque", "dejar bloque", "apilar un bloque sobre otro" y "desapilar un bloque de otro". Esta diferencia en la representación del problema varía muchos los resultados ya que, en el caso de UCPOP, al disponer de un sólo operador el número de nodos sucesores que se pueden generar de un nodo particular es mucho menor.

Los dos últimos problemas se corresponden con el dominio *Monkey*. PROB. 8 es el *monkey-test1* y PROB. 9 el *monkey-test2*. Los sorprendentes resultados obtenidos para este dominio se acentúan aún más porque las representaciones utilizadas son "idénticas" en ambos planificadores: mismo número de operadores, mismo número de precondiciones y tests en cada operador, y mismo número literales en el estado inicial. Los resultados para PROB. 8 son similares; sin embargo, en *test2*, donde la diferencia respecto a *test1* radica en la existencia de una nueva localización y dos nuevos objetos, las diferencias de resultados son notables. Esto se debe, como en otros problemas, a la aparición de un mayor número de valores para las variables que representa las distintas localizaciones del problema, lo que favorece el tratamiento y gestión de las variables que realiza SPLIN.

Conclusiones

SPLIN es un primer prototipo de planificación de orden parcial que ofrece las ventajas de un POP e incorpora funcionalidades adicionales en el tratamiento de las variables durante los procesos de unificación y resolución de tests condicionales. Estas dos características, unidas a la utilización de un lenguaje estructurado, flexible y de fácil manejo, permite obtener resultados satisfactorios en comparación con los obtenidos con el planificador UCPOP 4.0.

Actualmente se está trabajando para incluir nuevas funcionalidades en SPLIN, tales como un módulo de razonamiento temporal que utiliza un modelo basado en puntos de tiempo, y un módulo de abstracción que permite obtener una jerarquía de los datos del dominio de aplicación, pudiendo así realizar la planificación por niveles de la jerarquía del dominio. Esto permitirá abordar problemas más complejos y de aplicación a dominios reales.

Agradecimientos

Agradecemos a Eva Onaindía su inestimable ayuda tanto en la implementación de SPLIN como en los comentarios y sugerencias en la realización de este artículo.

Referencias

[Barrett94] Barrett A., Weld D.S. "Partial order planning: Evaluating possible efficiency gains". Artificial Intelligence, Vol. 67, No. 1, pp. 71-112, (1994).

[Barrett95] Barrett A., Christianson D., Friedman M., Kwok C., Golden K., Penberthy S., Sun Y., Weld D. "UCPOP User´s Manual (Version 4.0)". TR 93-09-06d, Dept. of Computer Science and Engineering. University of Washington, Seattle, WA 98105. Available via anonymous ftp from cs.washington.edu

[Chapman87] Chapman D. "Planning for Conjuntive goals". Artificial Intelligence Vol. 32, No. 3, pp. 333-377 (1987)

[CLIPS97] Lyndon B. Johnson Space Center (NASA). Software Technology Branch. "CLIPS Reference Manual": Vol. I "The Basic Programming Guide". Vol. II "The Adavanced Programming Guide". CLIPS Versión 6.05 (Octubre 1997).

[Fikes71] Fikes R.E., Nilsson N.J. "STRIPS: A new approach to the application of Theorem proving to Problem Solving". Artificial Intelligence, 2: 189-208 (1971).

[Gerevini96] Gerevini A., Schubert L. "Accelerating Partial-Order Planners: Some Techniques for Effective Search Control and Pruning". Journal of Artificial Intelligence Research, No. 5, p. 95-137, (1996).

[Joslin94] Joslin D., Pollack M.E. "Least-Cost Falw Repair: A Plan Refinement Strategy for Partial-Order PLanning". Proc. 12th National Conference on Artificial Intelligence, pp. 1004-1009 (1994)

[Knoblock94] Knoblock C.A. "Automatically Generating Abstractions for Planning". Artificial Intelligence, Vol. 68 , No. 2 , pp. 243-302 (1994).

[Peot93] Peot M.A., Smith D.E. "Threat-removal strategies for partial-order planning". In Proc. of AAAI-93, pp. 492-499, Washington DC, 1993.

[Penberthy92] Penberthy J.S., Weld D.S. "UCPOP: A Sound, Complete, Partial Order Planner for ADL". Proc. of the 1992 International Conference on Principles of Knowledge Representation and Reasoning, pp. 103-114, Morgan Kaufmann, Los Altos, CA, (1992).

[Pollack97] Pollack M.E., Joslin D., Paolucci M. "Flaw Selection Strategies for Partial-Order Planning". Journal of Artificial Intelligence Research, Vol. 6, pp. 232-262 (1997).

[Weld94] Weld Daniel S. "An Introduction to Least Commitment Planning". AI Magazine Vol.15, No.4, pp. 27-61 (Winter 1994).

[Yang97] Yang Q. "Intelligent Planning. A Decomposition and Abstraction Based Approach". Springer-Verlag. Berlin, Heidelberg, New York 1997. ISBN- 3-540-61901-1.

Apéndice A **Dominio Torres de Hanoi-1**

```
(defclass DISK (is-a user)          /* clase DISCO[2] */
      (non-temporal
            (ON (type DISK))              /* para indicar qué disco está encima de otro */
            (CLEAR (type INTEGER))
            (SIZE (type INTEGER))))       /* para indicar tamaño del disco */


(defobject D_NUL1 (is-a DISK)             /* representación del palo1 [2] */
      (SIZE 100))

(defobject D_NUL2 (is-a DISK)             /* representación del palo2*/
      (SIZE 100)
      (CLEAR "true"))

(defobject D_NUL3 (is-a DISK)             /* representación del palo3 */
      (SIZE 100)
      (CLEAR "true"))


(defobject D2 (is-a DISK)           /* disco2 */
      (ON D_NUL1)
      (SIZE 2))


(defobject D1 (is-a DISK)           /* disco1 */
      (ON D2)
      (SIZE 1)
      (CLEAR "true"))

/* para utilizar un tercer disco basta con especificar un nuevo objeto D3 que se sitúa directamente sobre
el palo1 y cuyo tamaño es 3. Modificar el slot ON del disco D2 para indicar que está sobre D3 */


(defaction MOVE_DISK (?disk ?below ?new_below)
      (preconds
            (DISK ?disk ON ?below)
            (DISK ?disk CLEAR "true")
            (DISK ?new_below CLEAR "true")
            (DISK ?disk SIZE ?s_disk)
            (DISK ?new_below SIZE ?s_new_below))
      (test
            (<> ?disk ?new_below)
            (<> ?new_below ?below)
            (<> ?below ?disk)
            (<> ?disk D_NUL1)
            (<> ?disk D_NUL2)
            (<> ?disk D_NUL3)
            (> ?s_new_below ?s_disk))
      (effects
            (primary
                  (DISK ?disk ON ?new_below)
                  (DISK ?below CLEAR "true"))
            (consequences
```

[2] Se ha utilizado una única clase DISK en la especificación de este problema para equiparar la representación de SPLIN a la utilizada en UCPOP 4.0.

[2] Es necesario crear objetos ficticios de tipo DISK para "simular" los palos donde se sitúan los discos, ya que se está empleando una única clase.

```
            (add)
            (delete
                (DISK ?disk ON ?below)
                (DISK ?new_below CLEAR "true")))))


(defobjective GOAL
    (DISK D2 ON D_NUL3)
    (DISK D1 ON D2))

/* para incluir un tercer disco modificar (DISK D2 ON D_NUL3) por (DISK D2 ON D3) e introducir el nuevo subbjetivo (DISK D3 ON D_NUL3) */
```

Apéndice B **Dominio Ferry**

```
(defclass PLACE (is-a user))

(defclass FERRY (is-a user)
    (non-temporal
        (slot LOCATION (type PLACE))
        (slot STATUS (type STRING))))

(defclass CAR (is-a user)
    (non-temporal
        (slot LOCATION (type PLACE))
        (slot ONFERRY (type FERRY))))

(defobject PA (is-a PLACE))

(defobject PB (is-a PLACE))

(defobject C1 (is-a CAR)
    (LOCATION PA))

(debobject C2 (is-a CAR)
    (LOCATION PA))

(defobject F1 (is-a FERRY)
    (LOCATION PA)
    (STATUS "empty"))

(defaction BOARD (?car ?place)
    (preconds
        (CAR ?car LOCATION ?place)
        (FERRY F1 LOCATION ?place)
        (FERRY F1 STAUTS "empty"))
    (effects
        (primary
            (CAR ?car ONFERRY F1))
        (consequencesNL
            (add)
            (delete
                (CAR ?car LOCATION ?place)
                (FERRY F1 STATUS "empty")))))


(defaction SAIL (?place1 ?place2)
    (preconds
        (FERRY F1 LOCATION ?place1))
```

```
    (test
        (<> ?place1 ?place2))
    (effects
        (primary
            (FERRY F1 LOCATION ?place2))
        (consequencesNL
            (add)
            (delete
                (FERRY F1 LOCATION ?place1)))))


(defaction DEBARK (?car ?place)
    (preconds
        (CAR ?car ONFERRY F1)
        (FERRY F1 LOCATION ?place))
    (effects
        (primary
            (CAR ?car LOCATION ?place)
            (FERRY F1 STATUS "empty"))
        (consequencesNL
            (add)
            (delete
                (CAR ?car ONFERRY F1)))))


(defobjective GOAL
    (CAR C1 LOCATION PB)
    (CAR C2 LOCATION PB))
```