Planificación con Aprendizaje de Reglas Rebatibles para elección de Acciones utilizando Argumentación

Diego García Alejandro García

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur. Av. Alem 1253, (8000) Bahía Blanca, Argentina
{dgarcia,agarcia}@cs.uns.edu.ar

En este trabajo se presenta un planificador, que durante la búsqueda de un plan, genera conocimiento en la forma de reglas rebatibles, para representan información de cuales acciones lo llevaron a encontrar un plan y cuales no. De esta forma, cuando se desea encontrar un plan para un nuevo problema, en lugar de explorar el árbol de búsqueda exhaustivamente, las reglas rebatibles generadas en planificaciones anteriores son utilizadas para argumentar sobre que acción aplicar en cada estado.

Ejemplo 1 : Supongamos que en el conocido "mundo de bloques" se quiere obtener un plan para hacer una pila con 3 bloques que están inicialmemnte sobre la mesa. En este ejemplo, la representación interna del estado inicial I esta dada por: $I = \{clear(a), clear(b), clear(c), ontable(a), ontable(b), ontable(c)\}$ y el estado final o meta por: $F = \{on(b,a), on(c,b), clear(c), ontable(a)\}$.

I

F

Se asumirá por conveniencia un planificador que parte del estado inicial hacia el estado final, eligiendo en cada estado E de un conjunto de acciones posibles $PA(E)$, y explorando el árbol de búsqueda en profundidad. En el caso del Ejemplo 1 el conjunto de $PA(I)$ es $\{stack(b,a), stack(c,a), stack(a,b), stack(c,b), stack(a,c), stack(b,c)\}$. Ante la ausencia de experiencia (reglas rebatibles), el planificador explorará todo el árbol de búsqueda. Aunque el planificador llegue a obtener un plan, seguirá explorando todo el árbol de búsqueda con el fin de obtener el plan óptimo y generar la mayor cantidad de experiencia posible (positiva y negativa) en la forma de reglas rebatibles.

En el proceso de exploración del árbol de búsqueda, cuando se llega a un nodo hoja N, se produce una recompensa positiva $(+)$ o negativa $(-)$ para premiar o penalizar las acciones que llevaron hasta ese nodo N. Esta recompesa se propaga luego hacia sus ancestros en el árbol, y se utiliza para generar reglas rebatibles que compilan la experiencia que obtuvo el planificador al ejecutar una acción en una situación particular. Cada nodo del árbol será etiquetado con un par (R, N), donde R será la recompensa y N un número que representa un nivel del árbol. Cuando el nodo hoja corresponde al estado final F, se produce una recompensa positiva por haber encontrado un plan, y se etiqueta al nodo con $(+, N)$, donde N es el nivel de la hoja. Si la hoja, en cambio, es un nodo A ya visitado anteriormente (ancestro), se produce una recompensa negativa, ya que ese camino no conduce a un plan. En este caso se etiqueta con $(-, N)$, donde N es el nivel de dicho ancestro A.
Para cado nodo interno E, la etiqueta se calcula de la siguiente manera:
(a) Si existe al menos un hijo de E con recompensa positiva, entonces la etiqueta de E será $(+, N)$ donde N es el mínimo nivel, entre las etiquetas de sus hijos con recompensa positiva. En este caso N corresponde a la longitud del plan mas corto que pasa por E.

(b) Si todos sus hijos tienen recompensa negativa, entonces la etiqueta de E será $(-, N)$ donde N es el mínimo nivel entre las etiquetas de sus hijos.

En la Figura 1 puede verse el subárbol que se forma de elegir la acción $stack(c, b)$ a partir del estado inicial I del Ejemplo 1. Cada nodo del árbol muestra la configuración de los bloques y está etiquetado con un nombre E_i : y el par $(recompensa, nivel)$. Los arcos están etiquetados con la acción que permite pasar del nodo padre al hijo.

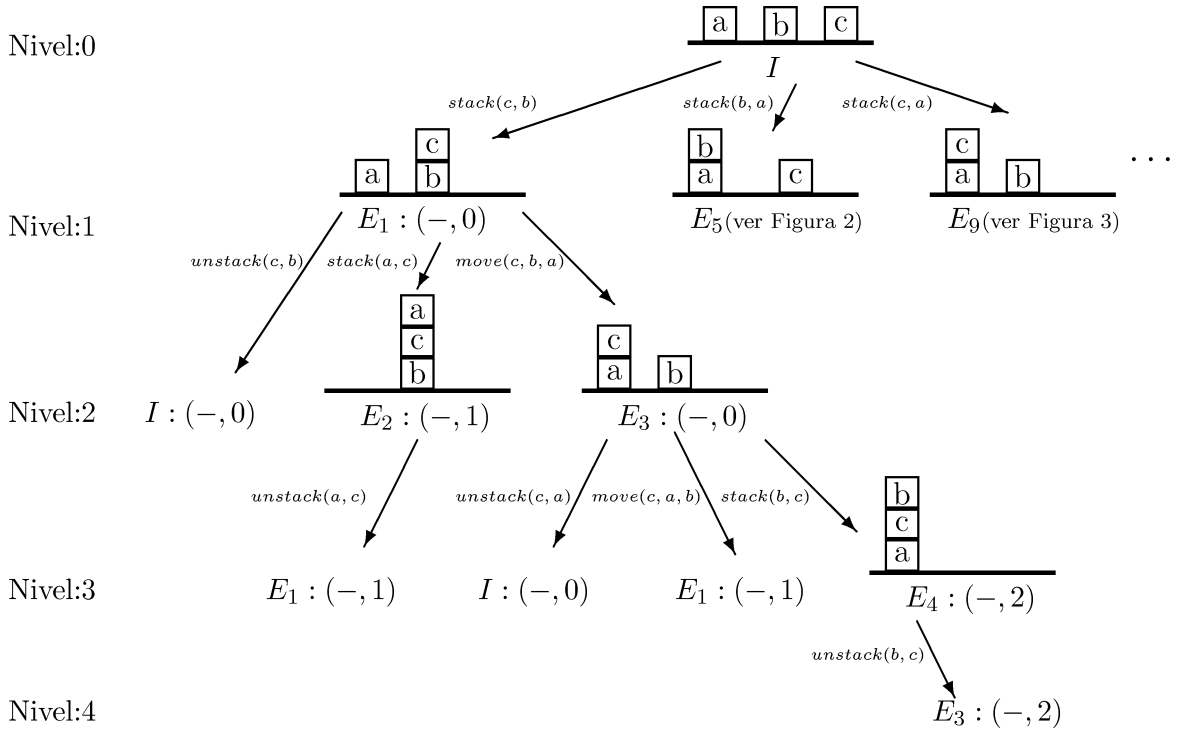


Figura 1: Parte del árbol de búsqueda del Ejemplo 1

En algunos nodos se generan reglas rebatibles que representan la experiencia de haber aplicado una acción Ac. Si la recompensa fue positiva se generará una regla de la forma "$do(Ac) -\prec G \cup E$", y si fue negativa "$\sim do(Ac) -\prec G \cup E$". Durante la exploración del subárbol de la figura 1, se generan las siguientes reglas:

(R_1) ~do(stack(a,c)) $-\prec$ $goal(on(b,a)), ontable(a), ontable(b), clear(a), clear(c), on(c,b)$

(R_2) ~do(stack(b,c)) $-\prec$ $goal(on(b,a)), goal(on(c,b)),$
$ontable(a), ontable(b), clear(a), clear(b), on(c,a)$

(R_3) ~do(stack(c,b)) $-\prec$ $goal(on(b,a)), goal(on(c,b)), ontable(a), ontable(b),$
$ontable(c), clear(a), clear(b), clear(c)$

Estas reglas se utilizarán para podar el árbol de búsqueda que se está explorando (Figura 3), y luego serán generalizadas como experiencia para construir planes en otros problemas similares (ver Ejemplo 2). La regla (R_1) se genera en el nodo E_1, y representa el conocimiento adquirido al explorar el subárbol con raíz E_2. La regla debe leerse de la siguiente forma: "si la búsqueda se encuentra en el estado E_1, definido por $ontable(a), ontable(b), clear(a), clear(c), on(c, b)$, y la meta restante es $goal(on(b, a))$, entonces hay razones para no apilar a sobre c". Esto se debe a que la ejecución de $stack(a, c)$ en E_1 lleva a la exploración de un subárbol cuyas hojas son todos nodos ya visitados, y descendientes de E_1. Esta situación se distingue por medio de la etiqueta $(-, 1)$ del nodo E_2. El "$-$" en $(-, 1)$ representa que todas las hojas del subarbol con

raíz E_2 son nodos ya visitados, y el "1" representa que todos los nodos visitados se encuentran del nivel 1 hacia abajo, por lo tanto son todos descendientes de E_1, ya que E_1 se encuentra en el nivel 1. La reglas R_2 y R_3 se generan en los estados E_3 e I respectivamente, por razones análogas a las explicadas para R_1.

En general, si una acción Ac lleva de un estado S a un estado E, y E tiene una etiqueta $(-, N)$, entonces la acción Ac recibe una recompensa negativa ya que ningún descendiente de aplicar esa acción lleva a un plan. Si además N es menor o igual al nivel de S, entonces se generará una regla de la forma "$\sim do(Ac) -\prec Antecedente$", donde $Antecendente$ esta formado por las metas (goals) que restan por cumplir en S, unido a la representación de S. En la siguiente figura se muestra como se generan reglas rebatibles que representan experiencia *positiva*.

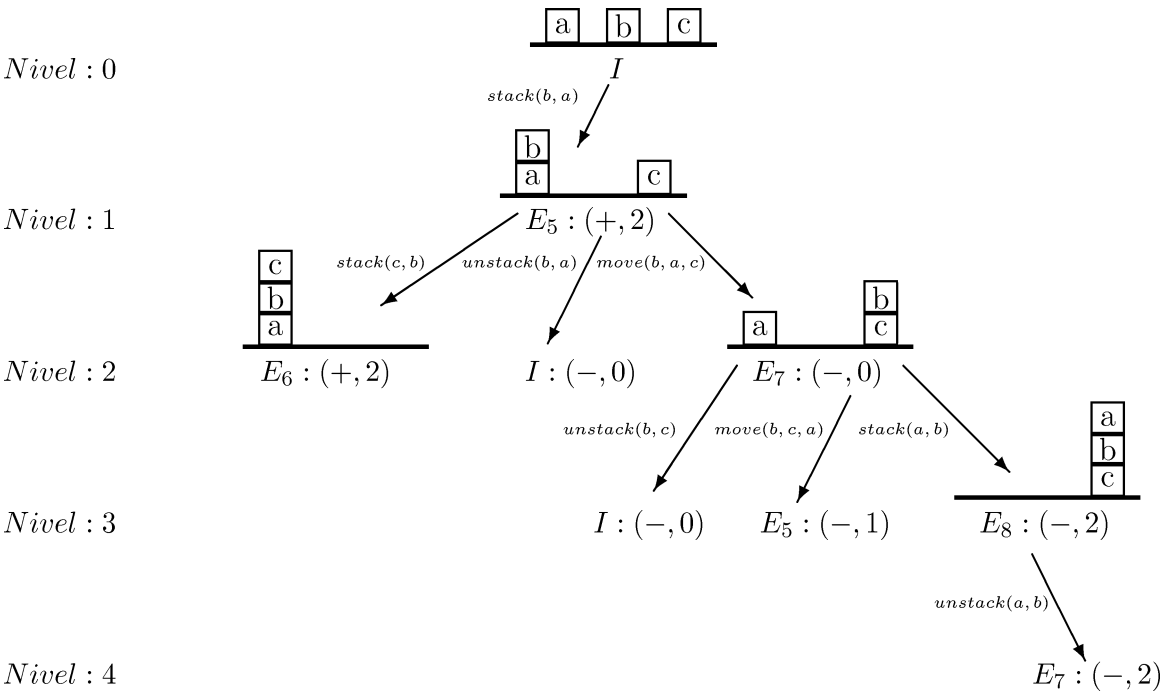


Figura 2: Parte del árbol de búsqueda del Ejemplo 1

A continuación se muestran las reglas generadas en el subarbol de la Figura 2:

(R_4) $do(stack(c,b)) \quad -\prec \quad goal(on(c,b)), ontable(a), ontable(c), clear(b), clear(c), on(b,a).$

(R_5) $\sim do(stack(a,b)) \quad -\prec \quad goal(on(b,a)), goal(on(c,b)),$
$ontable(a), ontable(c), clear(b), clear(c), on(b,c).$

(R_6) $do(stack(b,a)) \quad -\prec \quad goal(on(b,a)), goal(on(c,b),$
$ontable(a), ontable(b), ontable(c), clear(a), clear(b), clear(c).$

La regla (R_4) se genera en el nodo E_5, y representa el conocimiento adquirido al explorar el subárbol con raíz E_6, que es una hoja por ser el estado final F. A diferencia de las reglas generadas antes, esta regla representará conocimiento positivo, ya que este camino del árbol corresponde a un plan posible. La regla debe leerse de la siguiente forma: "si la búsqueda se encuentra en el estado E_5, definido por $ontable(a), ontable(c), clear(b), clear(c), on(b,a)$, y la meta restante es $goal(on(c,b))$, entonces hay razones para apilar c sobre b $(stack(c,b))$". Esto se debe a que la ejecución de $stack(c,b)$ en E_5 lleva a concretar un plan. La regla (R_5) se genera en el nodo E_7 de forma análoga a lo explicado para (R_1). La regla (R_6) en cambio, se genera en el nodo I de forma análoga a (R_4).

La Figura 3 muestra otra parte del árbol de búsqueda del Ejemplo 1, donde se genera en el nodo I la regla (R_7) = ∼do(stack(c,a)) −≺ goal(on(b,a)), goal(on(c,b)), ontable(a), ontable(b), ontable(c), clear(a), clear(b), clear(c). Además, el conocimiento generado hasta el momento (reglas R_1 a R_6) permiten podar el árbol de búsqueda. Observese que en el estado E_9, la acción $stack(b, c)$ no es elegida porque tiene un argumento para no ejecutarla, generado con R_2.

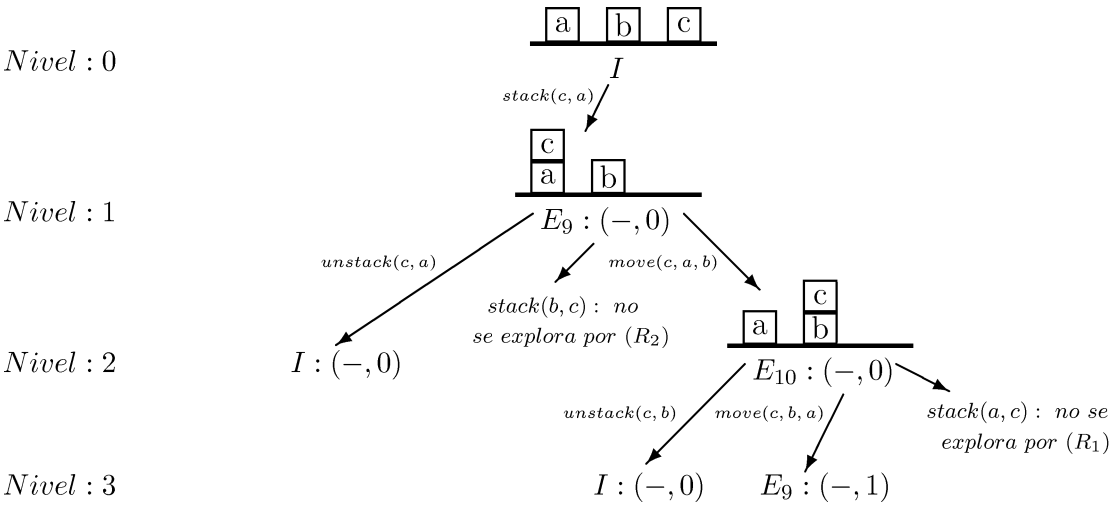


Figura 3: Parte del árbol de búsqueda del Ejemplo 1

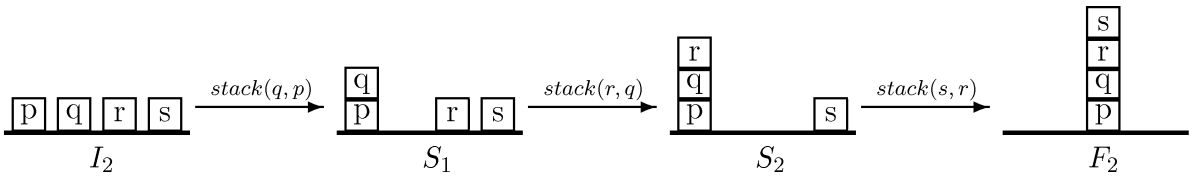
Una vez finalizada la exploración del árbol, las reglas rebatibles generadas son generalizadas y guardadas como experiencia para la búsqueda de futuros planes. Para generalizar una regla, se reemplaza cada atomo por una variable, y se elimina del cuerpo de la misma la descripción del estado, dejando solamente las metas. A continuación figuran las reglas obtenidas de generalizar las reglas R_1 a R_7:

(G_1) $\sim do(stack(X1, X3)) -\prec goal(on(X2, X1))$
(G_2) $\sim do(stack(X2, X3)) -\prec goal(on(X2, X1)), goal(on(X3, X2))$
(G_3) $\sim do(stack(X3, X2)) -\prec goal(on(X2, X1)), goal(on(X3, X2))$
(G_4) $do(stack(X3, X2)) -\prec goal(on(X3, X2))$
(G_5) $do(stack(X2, X1)) -\prec goal(on(X2, X1)), goal(on(X3, X2)$
(G_6) $\sim do(stack(X1, X2)) -\prec goal(on(X2, X1)), goal(on(X3, X2))$
(G_7) $\sim do(stack(X3, X1)) -\prec goal(on(X2, X1)), goal(on(X3, X2))$

Utilización de la experiencia para encontrar planes

El conjunto de reglas rebatibles generadas por el planificador (como por ejemplo G_1 a G_7) define un programa lógico rebatible. Por lo tanto, puede utilizarse un intérprete de DeLP para razonar y generar argumentos a favor y en contra de ejecutar acciones. De esta forma, antes de explorar el árbol de búsqueda para un nuevo problema, el planificador utilizará el intérprete de DeLP para elegir, dentro de las acciones posibles a realizar en un estado, una acción Ac que esté garantizada apartir del conocimiento generado en planes anteriores. Esto es, se utilizará DeLP para buscar un literal de la forma $do(Ac)$ que esté garantizado.

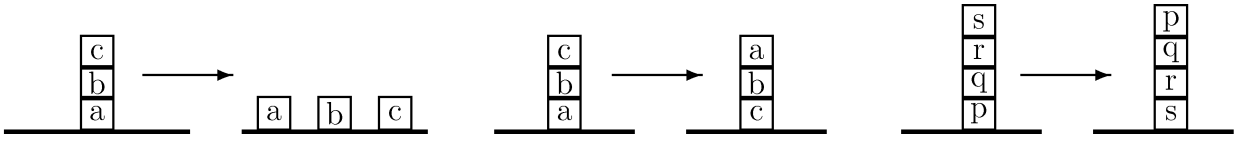
Ejemplo 2 : Supongamos que el planificador fue utilizado inicialmente para resolver el problema del Ejemplo 1, generando las reglas G_1 a G_7, y se desea obtener ahora un plan para hacer una pila con cuatro bloques p, q, r y s que están inicialmente sobre la mesa. Utilizando la experiencia de las reglas rebatibles, el plan es generado directamente expandiendo únicamente cuatro nodos como muestra la siguiente figura:



La representación interna del estado inicial esta dada por: $I_2 = \{clear(p), clear(q), clear(r), clear(s), ontable(p), ontable(q), ontable(r), ontable(s)\}$ y el estado final o meta por: $F_2 = \{on(s,r), on(r,q), on(q,p), clear(s), ontable(p)\}$. Las metas restantes a cumplir en el estado inicial, se obtienen como la diferencia entre el estado final y el inicial. Estas son: $G = \{goal(on(s,r)), goal(on(r,q)), goal(on(q,p))\}$. Sea Δ el conjunto de reglas rebatibles generadas como experiencia de planes anteriores. A partir del programa lógico rebatible $\Delta \cup G$, el único literal garantizado por DeLP es $do(stack(p,q))$ ya que el argumento que lo sustenta: $\{do(stack(q,p)) -\!\prec goal(on(q,p)), goal(on(r,q))\}$ que es generado usando la regla (G_5), no está derrotado. Esto significa que dadas las metas G hay un argumento no derrotado, basado en la experiencia del planificador, para aplicar la acción $stack(p,q)$. Todas las demás acciones posibles de aplicar en el estado I_2, y que tienen algún argumento, son derrotados por un argumento en contra. Por ejemplo, el argumento $\{do(stack(r,q)) -\!\prec goal(on(r,q)), goal(on(s,r))\}$ generado con (G_5) que sustenta $do(stack(r,q))$ es derrotado por $\{\sim do(stack(r,q)) -\!\prec goal(on(q,p)), goal(on(r,q))\}$ generado utilizando la regla (G_3). Una vez ejecutada la acción $stack(q,p)$ se llega al estado S_1 donde las metas restantes son $\{goal(on(s,r)), goal(on(r,q))\}$. Con estas nuevas metas como hechos el único literal garantizado por DeLP es ahora $do(stack(r,q))$ sustentado por el argumento no derrotado $\{do(stack(r,q)) -\!\prec goal(on(r,q)), goal(on(s,r))\}$ generado también con (G_5). Al aplicar esta acción se llega al estado S_2 donde la meta restante es $\{goal(on(s,r))\}$. Ahora el único literal garantizado es $do(stack(s,r))$ sustentado por el argumento $\{do(stack(s,r)) -\!\prec goal(on(s,r))\}$ generado con (G_4). La acción $do(stack(s,r))$ lleva directamente al estado final.

Resultados Preliminares

A continuacion se muestran algunos ejemplos y el desempeño del planificador. Después de ejecutar el planificador para el Ejemplo 1 y el Ejemplo 3, si se lo ejecuta para el Ejemplo 4, el planificador encuentra en tres pasos el plan óptimo, utilizando sólo la experiencia y sin necesidad de explorar el árbol de búsqueda. Si se lo ejecuta para el Ejemplo 5, encuentra dos planes (de los cuales uno es el óptimo), solamente expandiendo ocho nodos del árbol de búsqueda, y utilizando la experiencia en seis de los mismos.



Ejemplo 3 Ejemplo 4 Ejemplo 5

Referencias

[1] S. Džeroski, L. De Raedt, and H. Blockeel. *Relational Reinforcement Learning. The Eighth International* Conference ILP'98. Pages 11-22, Madison, Wisconsin, USA, 1998.

[2] A. García. *Programación en Lógica Rebatible, Definición, Semántica Operacional y Paralelismo.* Tesis Doctoral. Universidad Nacional del Sur. 2000.

[3] D. Kasakov and D. Kudenko. Machine Learning and Inductive Logic Programming for Multi-Agent Systems Summer School of MAS. Praga. 2001.