UNICAMP
Universidade Estadual de Campinas

FEEC
Faculdade de Engenharia Elétrica e de Computação

Aluno: Mateus Neves Barreto
R.A.: 142358
Disciplina: IA006
Professor: Ricardo R. Gudwin

Relatório – Aula 6

1 Atividade 1

Neste tópico será apresentado a execução do tutorial 7 do SOAR, onde é abordado aprendizagem por reforço. O aprendizado por reforço permite que os agentes do SOAR mudem o comportamento ao longo do tempo. Isso é feito através de um sistema de recompensas, que altera probabilisticamente o comportamento do agente, esse mecanismo contrasta com a aprendizagem *chunking*.

1.1 Reforço de aprendizagem em ação

Para iniciarmos, implementaremos um agente simples que pode apenas escolher duas opções, se mexer para a direita ou para a esquerda. Inicialmente o agente não possui preferência por nenhum sentido, mas assim que é tomada a decisão, o mesmo recebe um *feedback* de -1 se o sentido da esquerda foi tomado e 1 para a direita.

1.1.1 O agente *left-right*

Utilizando o aprendizado por reforço, o agente vai aprender rapidamente que o sentido para a direita é preferível. A Figura 1 apresenta o código que propõe o agente esquerda-direita. O código do projeto utilizado neste tópico está presente no diretório "*left-right_tutorial7*".

```
sp {propose*initialize-left-right
   (state <s> ^superstate nil
              -^name)
-->
   (<s> ^operator <o> +)
   (<o> ^name initialize-left-right)
}
sp {apply*initialize-left-right
   (state <s> ^operator <op>)
   (<op> ^name initialize-left-right)
-->
   (<s> ^name left-right
        ^direction <d1> <d2>
        ^location start)
   (<d1> ^name left ^reward -1)
   (<d2> ^name right ^reward 1)
}
```

Figura 1 – *Initialize* agente.

O agente armazena as direções e associa a recompensa com o estado.
O agente pode se mover para qualquer direção disponível, a direção escolhida é armazenada no estado, Figura 2.

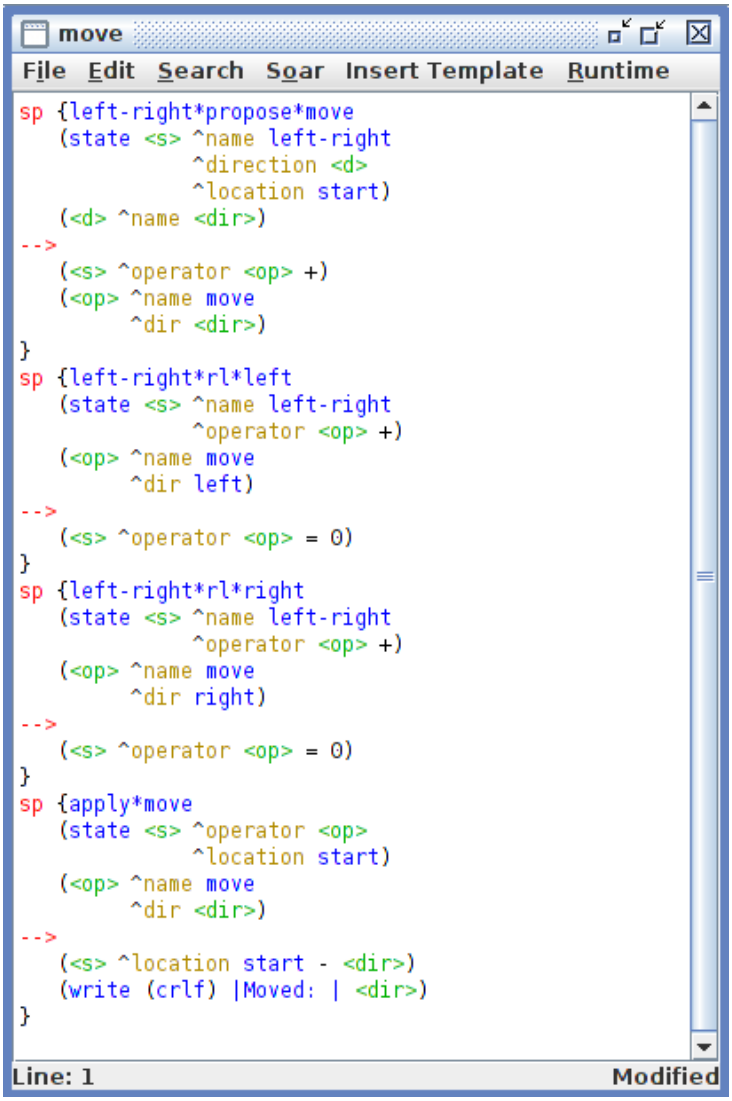

Figura 2 – Armazenando escolha de direção.

Assim que o agente escolhe a direção, ele recebe o *feedback* com a recompensa, podendo ser positiva ou negativa, Figura 3.

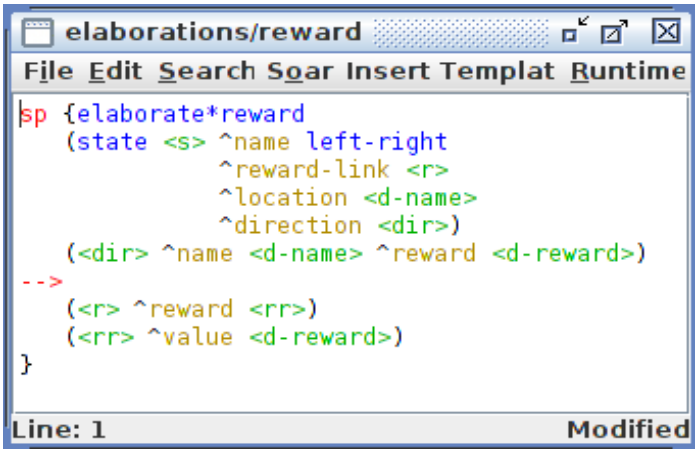

Figura 3 – Regra de recompensa.

Quando um agente escolhe a direção a tarefa acaba e o agente é pausado, Figura 4.

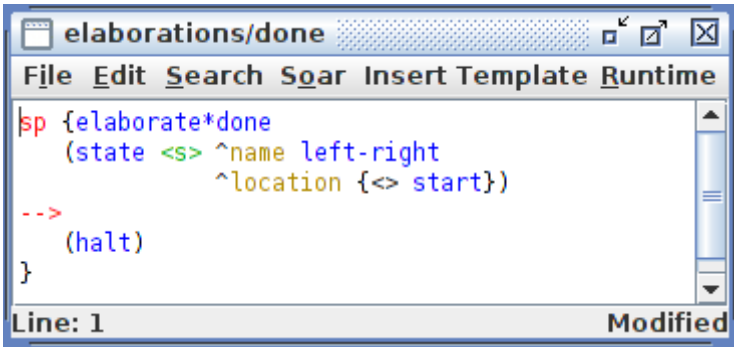

Figura 4 – Regra de conclusão.

A execução do código apresentado nas figuras irá gerar elementos inesperados, as rezões para estes comportamentos serão explicados nos próximos subtópicos. A Figura 5 apresenta a execução simples do código, sem a ativação de aprendizagem nenhuma, note que a saída foi aleatória.

```
Agent reinitialized.
11 productions excised.
source {/home/mateus/Dropbox/Mestrado/IA006/aula6/left-right_tutorial7/left-ri
**********
Total: 11 productions sourced.
run
     1: O: O1 (initialize-left-right)
     2: O: O2 (move)
Moved: left
Interrupt received.
This Agent halted.

An agent halted during the run.
init-soar
     0: ==>S: S1
Agent reinitialized.
run
     1: O: O1 (initialize-left-right)
     2: O: O3 (move)
Moved: right
Interrupt received.
This Agent halted.

An agent halted during the run.
```

Figura 5 – Execução simples do projeto *left-right*.

1.1.2 Executando o agente *left-right*

Para executar o código apresentado neste subtópico, é necessário abri-lo no SoarDebugger. Através da linha de comando, insira os seguintes comandos para habilitar os mecanismos de aprendizagem:

- *"rl --set learning on";*
- *"indifferent-selection –epsilon-greedy".*

O primeiro comando ativa o aprendizado, e o segundo define a política de exploração. Além do dois comandos que são utilizados no início, o comando *"print –rl"* é utilizado entre os *steps* (ciclos), observe a Figura 6. Note que inicialmente os valores de preferência possuem o valor zero após o segundo *step* (seguinte a inicialização), no terceiro *step* (segundo após inicialização) o valor 0,3 é associado para a regra com movimento a direita, setando como um a escolha da direita. Se a regra da esquerda fosse escolhido, o valor setado na regra para esquerda seria de 1 e a indiferença

numérica para -0,3; Figura 7.

```
source {/home/mateus/Dropbox/Mestrado/IA006/aula6/left-right_tutorial7/left-rig
***********
Total: 11 productions sourced.
rl --set learning on
indifferent-selection --epsilon-greedy
step
     1: O: O1 (initialize-left-right)
print --rl
left-right*rl*right  0, 0
left-right*rl*left  0, 0
step
     2: O: O3 (move)
print --rl
left-right*rl*right  0, 0
left-right*rl*left  0, 0
step
Moved: right
Interrupt received.
This Agent halted.

An agent halted during the run.
print --rl
left-right*rl*right  1, 0,3
left-right*rl*left  0, 0
```

Figura 6 – Decisão tomada para à direita.

```
source {/home/mateus/Dropbox/Mestrado/IA006/aula6/left-right_tutorial7/left-rig
***********
Total: 11 productions sourced.
rl --set learning on
indifferent-selection --epsilon-greedy
step
     1: O: O1 (initialize-left-right)
print --rl
left-right*rl*right  0, 0
left-right*rl*left  0, 0
step
     2: O: O2 (move)
print --rl
left-right*rl*right  0, 0
left-right*rl*left  0, 0
step
Moved: left
Interrupt received.
This Agent halted.

An agent halted during the run.
print --rl
left-right*rl*right  0, 0
left-right*rl*left  1, -0,3
```

Figura 7 – Decisão tomada para à esquerda.

Se o agente for inicializado novamente após a execução ("*Init-soar*"), e reexecutado ("*Run*"), o aprendizado da execução anterior se mantém. Com isso, a cada vez que se reexecuta o agente o mesmo adquire um conhecimento a mais. No problema tratado, o agente começa a perceber que se ele escolher a direita ao invés da esquerda ele terá uma maior recompensa. A Figura 8 apresenta o resultado da estrutura de aprendizado por reforço após 20 execuções seguidas.

```
Agent reinitialized.
run
     1: O: O1 (initialize-left-right)
     2: O: O3 (move)
Moved: right
Interrupt received.
This Agent halted.

An agent halted during the run.
init-soar
     0: ==>S: S1
Agent reinitialized.
run
     1: O: O1 (initialize-left-right)
     2: O: O3 (move)
Moved: right
Interrupt received.
This Agent halted.

An agent halted during the run.
print --rl
left-right*rl*right  18. 0.9983715864020896
left-right*rl*left  1. -0.3
init-soar
     0: ==>S: S1
Agent reinitialized.
run
     1: O: O1 (initialize-left-right)
     2: O: O3 (move)
Moved: right
Interrupt received.
This Agent halted.

An agent halted during the run.
print --rl
left-right*rl*right  19. 0.9988601104814627
left-right*rl*left  1. -0.3
```

Figura 8 – Resultado da aprendizagem por reforço.

Note que em 20 repetições, o agente escolheu apenas uma vez a opção da esquerda, e ao receber o *feedback* negativo passou a notar que a opção da direita a recompensa seria maior.

1.2 Construindo um agente com aprendizado

Para utilizar o aprendizado por reforço na maioria dos agentes são em três etapas: a primeira é a utilização da regra de aprendizagem por reforço; a segunda é a implementação de uma ou mais regras de recompensa; a terceira e última basta ativar a utilização do aprendizado por reforço no momento de execução ou via código (agente).
Como exemplo neste subtópico será atualizado o agente do problema jarro de água para ser utilizado com reforço de aprendizagem. O código referente a este item está presente no diretório "*water-jug_tutorial7*".

1.2.1 Regras de reforço de aprendizagem

A estrutura de sintaxe que uma regra de aprendizagem por reforço deve seguir está presente na Figura 9.

```
sp {my*proposal*rule
   (state <s> ^operator <op> +
              ^condition <c>)
-->
   (<s> ^operator <op> = 2.3)
}
```

Figura 9 – Estrutura de uma regra de aprendizado por reforço.

O objetivo deste item é apresentar como utilizar a aprendizagem por reforço no problema do jarro da água. Para realizar esta tarefa, os operadores de "*empty*", "*fill*" e "*pour*" serão modificados. Para modificar o agente *water-jug* para suportar aprendizagem por reforço são necessárias duas etapas: a primeira etapa é a modificação das regras existentes; a segunda é a criação das regras de aprendizagem por reforço.
A primeira modificação é simples, basta remover os operadores de indiferença das regras, Figura 10.

```
sp {water-jug*propose*empty
   (state <s> ^name water-jug
              ^jug <j>)
   (<j> ^contents > 0)
-->
   (<s> ^operator <o> +)
   (<o> ^name empty
        ^empty-jug <j>)}

sp {water-jug*propose*fill
   (state <s> ^name water-jug
              ^jug <j>)
   (<j> ^empty > 0)
-->
   (<s> ^operator <o> +)
   (<o> ^name fill
        ^fill-jug <j>)}

sp {water-jug*propose*pour
   (state <s> ^name water-jug
              ^jug <i> { <> <i> <j> })
   (<i> ^contents > 0 )
   (<j> ^empty > 0)
-->
   (<s> ^operator <o> +)
   (<o> ^name pour
        ^empty-jug <i>
        ^fill-jug <j>)}
```

Figura 10 – Removendo operadores de indiferença.

A segunda modificação pode ser um pouco mais elaborada. Pois será necessário fornecer um *feedback* para cada estado do problema. No problema tratado, o estado pode ser representado pelo volume dos jarros e as ações *empty, fill* e *pour.* A Figura 11 apresenta o código gerado para o estado: "esvaziar o jarro de 3 litros (que contém 2 litros) e ao mesmo tempo o jarro de 5 litros contém 4 litros".

```
sp {water-jug*empty*3*2*4
   (state <s> ^name water-jug
              ^operator <op> +
              ^jug <j1> <j2>)
   (<op> ^name empty
         ^empty-jug.volume 3)
   (<j1> ^volume 3
         ^contents 2)
   (<j2> ^volume 5
         ^contents 4)
-->
   (<s> ^operator <op> = 0)
}
```

Figura 11 – 1ª regra de *feedback*.

Para agentes mais simples, como o *left-rigth*, é possível representar todos os estados possíveis. Porém para o problema do jarro da água, o número de possibilidades é alto (3*2*4*6 =144 regras), gerando a necessidade de 144 regras de reforço de aprendizagem a serem escritas, para cobrir todos os estados possíveis. Mas, estas regras podem ser representadas de forma combinatória. Para gerar estas regras de forma combinatória são utilizados os comandos *SOAR gp,* Figura 12.

```
gp {rl*water-jug*empty
   (state <s> ^name water-jug
              ^operator <op> +
              ^jug <j1> <j2>)
   (<op> ^name empty
         ^empty-jug.volume [3 5])
   (<j1> ^volume 3
         ^contents [0 1 2 3])
   (<j2> ^volume 5
         ^contents [0 1 2 3 4 5])
-->
   (<s> ^operator <op> = 0)
}
gp {rl*water-jug*fill
   (state <s> ^name water-jug
              ^operator <op> +
              ^jug <j1> <j2>)
   (<op> ^name fill
         ^fill-jug.volume [3 5])
   (<j1> ^volume 3
         ^contents [0 1 2 3])
   (<j2> ^volume 5
         ^contents [0 1 2 3 4 5])
-->
   (<s> ^operator <op> = 0)
}
gp {rl*water-jug*pour
   (state <s> ^name water-jug
              ^operator <op> +
              ^jug <j1> <j2>)
   (<op> ^name pour
         ^empty-jug.volume [3 5])
   (<j1> ^volume 3
         ^contents [0 1 2 3])
   (<j2> ^volume 5
         ^contents [0 1 2 3 4 5])
-->
   (<s> ^operator <op> = 0)
}
```

Figura 12 – Regras de aprendizagem por reforço – *utilizando regra gp*.

1.2.2 Regras de recompensas

Regras de aprendizagem são como qualquer outra regra em SOAR, a única especifidade é que as mesmas modificam a estrutura *reward-link* do estado, refletindo nas decisões do agente. Os valores de recompensa devem ser armazenados no *reward-link*. O SOAR não remove ou modifica esta estrutura de recompensa, é responsabilidade do agente manter a estrutura de recompensa para ser utilizada no mecanismo de aprendizagem por reforço.
Para o problema jarro da água (*water-jug*) o agente só ganha a recompensa se atingir a meta. Então a modificação na regra de meta é necessária, conforme a Figura 13.

```
sp {water-jug*detect*goal*achieved
   (state <s> ^name water-jug
              ^jug <j>
              ^reward-link <rl>)
   (<j> ^volume 3 ^contents 1)
-->
   (write (crlf) |The problem has been solved.|)
   (<rl> ^reward.value 10)
   (halt)}
```

Figura 13 – Regra de meta com recompensa.

Agora, basta rodar o agente ativando a aprendizagem por reforço. Para facilitar, basta inserir os comandos: *"rl --set learning on"* e *"indifferent-selection –epsilon-greedy"* no projeto, pelo *VisualSOAR* no arquivo *_firstload*. A Figura 14, apresenta a execução do agente no *SoarJavaDebugger*. Especificamente na Figura 14, o agente foi executado poucas vezes, conseguindo convergir para a solução ótima, devido a utilização da aprendizagem por reforço.

```
init-soar
     0: ==>S: S1
Agent reinitialized.
run
     1: O: O1 (initialize-water-jug)
 5:0 3:0
     2: O: O2 (fill)
  FILL(3)
 5:0 3:3
     3: O: O4 (pour)
  POUR(3:3,5:0)
  POUR(3:0,5:3)
 5:3 3:0
     4: O: O8 (fill)
  FILL(3)
 5:3 3:3
     5: O: O10 (pour)
  POUR(3:3,5:3)
  POUR(3:1,5:5)
 5:5 3:1
The problem has been solved.
Interrupt received.
This Agent halted.

An agent halted during the run.
```

Figura 14 – Execução do agente com aprendizagem por reforço.

A Figura 15 contém o resultado da estrutura de recompensa após a execução da Figura 14.

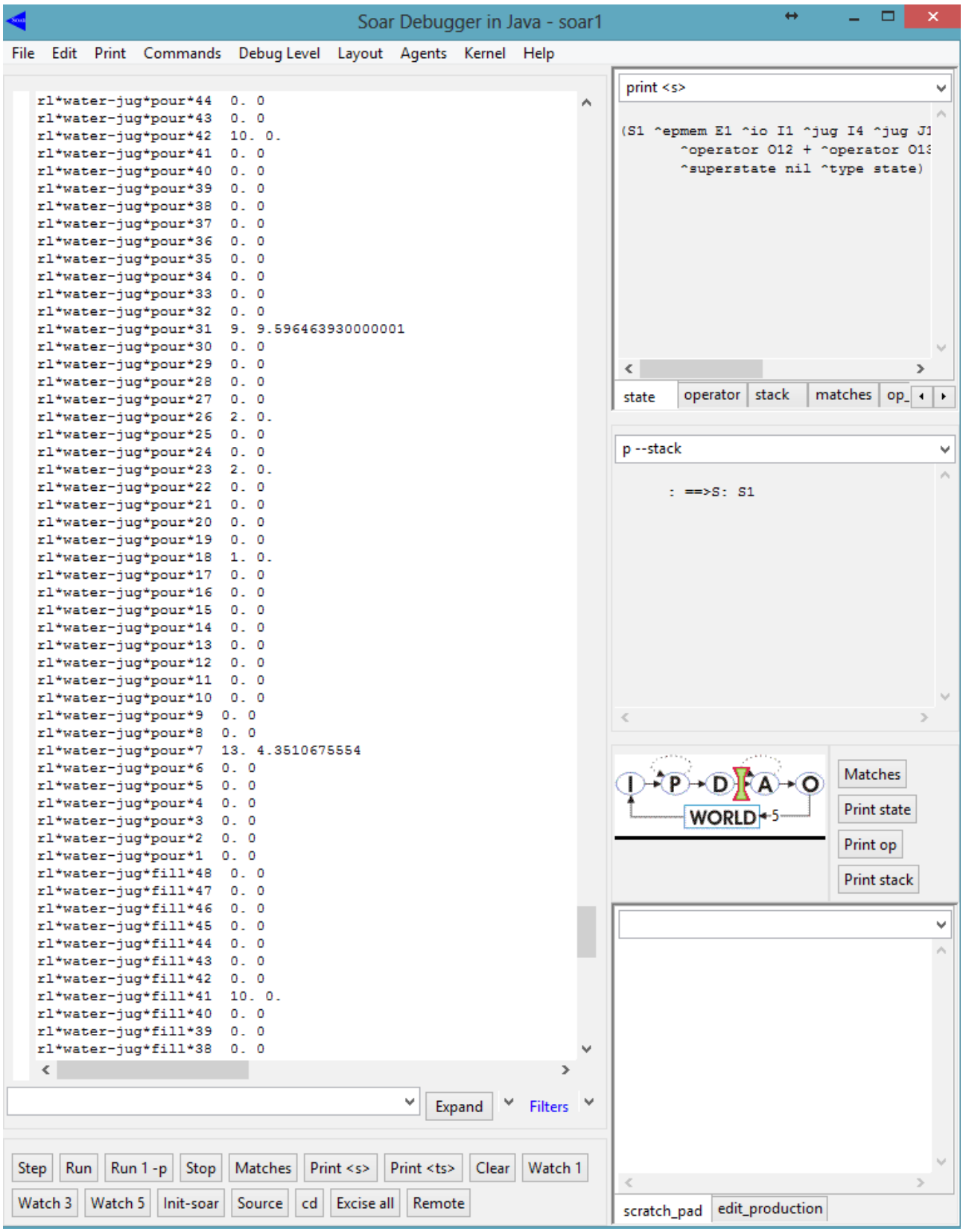

Figura 15 – Estrutura de aprendizagem por reforço.

Na página de apresentação de atividade possui um video contendo a execução da Figura 14 e 15.

1.3 Além da exploração

Voltando ao problema do agente *left-right,* agora será explicado porque casos presentes na Figura 16 ocorrem. Mesmo já sabendo que a escolha para a direita é uma boa escolha, porque foi escolhido

a direção da esquerda na quinta execução? Essa resposta está relacionada com as políticas de exploração. Existem momentos na exploração que consideram o teste do menos preferido, verificando se esse estado pode levar a um caminho útil. Através do comando *indifferent-selection* é possível alterar os níveis de exploração.

```
source {D:\mateus\Dropbox\Mestrado
***********
Total: 11 productions sourced.
watch 0
run
Moved: right
Interrupt received.
This Agent halted.

An agent halted during the run.
init-soar
Agent reinitialized.
run
Moved: right
Interrupt received.
This Agent halted.

An agent halted during the run.
init-soar
Agent reinitialized.
run
Moved: right
Interrupt received.
This Agent halted.

An agent halted during the run.
run

An agent halted during the run.
init-soar
Agent reinitialized.
run
Moved: right
Interrupt received.
This Agent halted.

An agent halted during the run.
init-soar
Agent reinitialized.
run
Moved: left
Interrupt received.
This Agent halted.

An agent halted during the run.
print --rl
left-right*rl*right  4. 0.7599
left-right*rl*left  1. -0.3
```

Figura 16 – Exploração.

Utilizando o comando "*indifferent-selection –stats*" no *SoarJavaDebugger* obtém-se o resultado presente na Figura 17.

```
indifferent-selection --stats
Exploration Policy: epsilon-greedy
Automatic Policy Parameter Reduction: off

epsilon: 0.1
epsilon Reduction Policy: exponential
epsilon Reduction Rate (exponential/linear): 1/0

temperature: 25
temperature Reduction Policy: exponential
temperature Reduction Rate (exponential/linear): 1/0
```

Figura 17 – Seleção indiferente.

Este comando retorna a política de exploração e parâmetros. Existem cinco tipos de políticas: *boltzmann, epsilon-greedy, softmax, first,* e *last*. Para alterar a política de exploração, basta usar o seguinte comando: "*indifferent-selection –nomePolitica*", onde "*nomePolitica*" deve ser uma dos cinco tipos de política de exploração.
No tutorial só é discutida a política *epsilon-greedy,* onde o parâmetro *epsilon* escolhe o operador mais preferido. Então, com (1-*epsilon*) de probabilidade o operador mais preferido é selecionado, e com probabilidade de *epsilon* é selecionado um operador todos os operadores restantes.
Quando o SOAR é inicializado, a política padrão de exploração é a *softmax,* porém a primeira vez que se utiliza aprendizagem por reforço, a arquitetura muda automaticamente para política de exploração *epsilon-greedy,* política mais adequada aos agentes de aprendizagem por reforço. Para alterar o valor de *epsilon* basta utilizar o comando: "*indifferent-selection --epsilon <value>*", onde "*<value>*" é o valor desejado para *epsilon* e lembrando que o valor deve estar entro 0 e 1. Sendo o valor zero, desativando a exploração e 1 resultará em uma seleção uniformemente aleatória.

2 Atividade 2

Neste tópico será apresentado a execução do tutorial 8 do SOAR, onde é abordada a memória semântica, que permite que o agente armazene ou recupere objetos persistentes.

2.1 O armazenamento semântico

Antes de detalhar um agente que usa memória semântica, será apresentado um exemplo de pré conhecimento e visualização da memória. Para iniciar, basta inserir o comando presente na Figura 18 no *SoarJavaDebugger.*

```
smem --add {(<a> ^name alice ^friend <b>)(<b> ^name bob ^friend <a>)(<c> ^name charley)}
```

Figura 18 – Comando - I.

O comando da Figura 18, carrega 3 objetos na memória semântica. Este comando é útil para pré carregar o conteúdo de grandes bases em SOAR. Para imprimir a memória semântica, basta utilizar o comando: "*smem –print*", que gera a saída presente na Figura 19.

```
smem --add {(<a> ^name alice ^friend <b>)(<b> ^name bob ^friend <a>)(<c> ^name charley)}
smem --print
(@A1 ^friend @B1 ^name alice [+1.000])
(@B1 ^friend @A1 ^name bob [+2.000])
(@C3 ^name charley [+3.000])
```

Figura 19 – Memória semântica.

Note que os identificadores da memória semântica possuem @ como prefixo, são persistentes e por isso geralmente chamados de identificadores *long-term.* Ao contrário das regras da memória de trabalho o conhecimento da memória semântica não precisa estar relacionado com nenhum estado.
É possível visualizar o conteúdo da memória semântica graficamente, isso é feito através da

combinação dos comandos presentes na Figura 20. O comando utilizado nesta Figura, gera como saída um arquivo no diretório corrente, que pode ser verificado pelo comando *pwd.*

```
command-to-file smem.gv smem --viz
pwd
D:/mateus/Dropbox/Mestrado/IA006/win64
```

Figura 20 – Visualização gráfica.

O comando: "*command-to-file smem.gv smem --viz*" gera um arquivo padrão do programa *Graphviz (http://graphviz.org)* que renderiza o código para um diagrama, Figura 21.

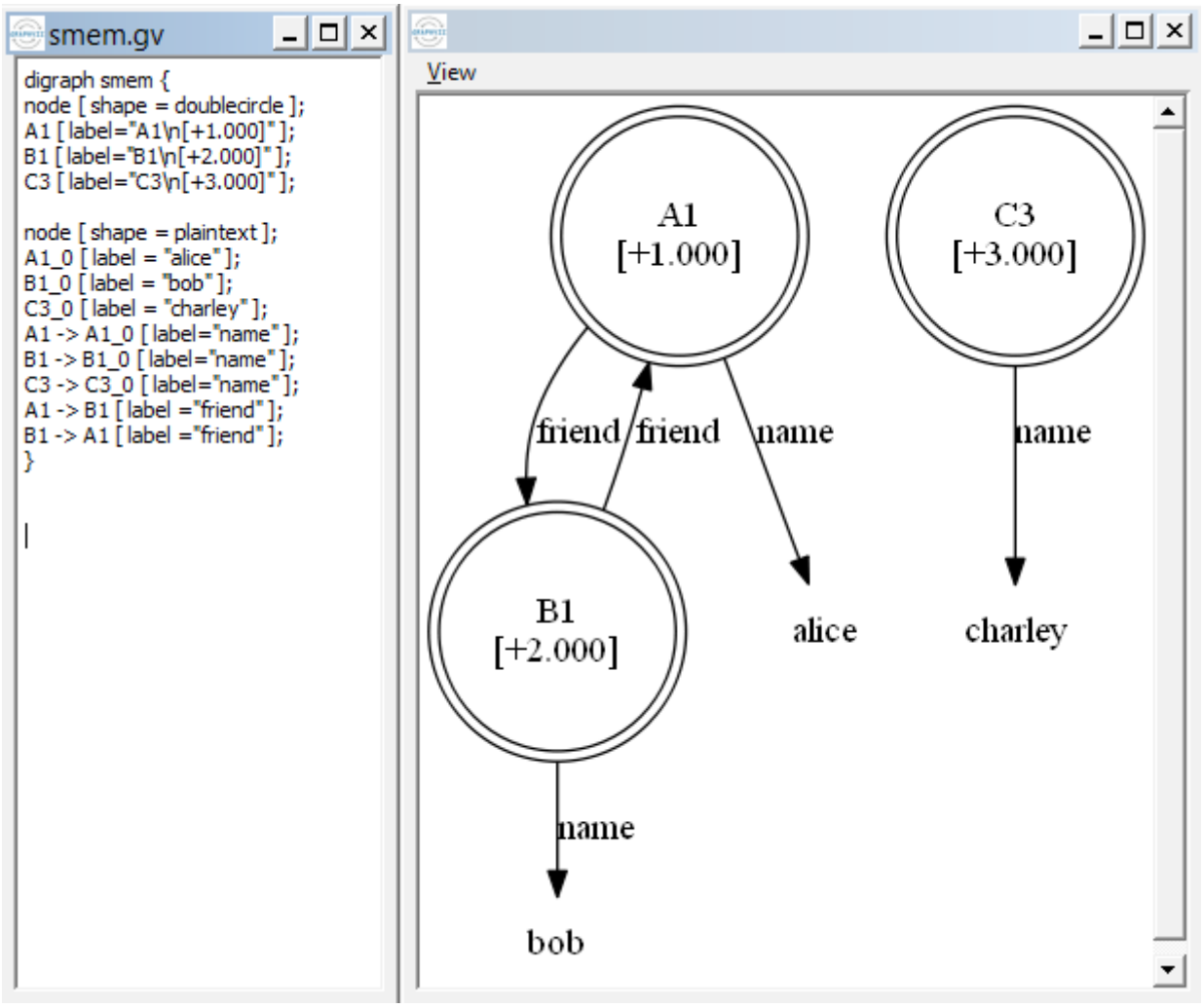

Figura 21 – Diagrama gerado.

Na janela à esquerda presente na Figura 21, está o código gerado pelos comandos presentes na Figura 20. Basta abrir o arquivo gerado no diretório corrente com o programa *Graphviz.*
Para provar que nenhum destes conhecimentos estão em outras memórias, procedural e/ou de trabalho, basta verificar o conteúdo das memórias através dos comandos: "*print*" e "*print --depth 100 <s>*" respectivamente, Figura 22.

```
print --depth 100 <s>
(S1 ^epmem E1 ^io I1 ^reward-link R1 ^smem S2 ^superstate nil ^type state)
  (E1 ^command C1 ^present-id 1 ^result R2)
  (I1 ^input-link I2 ^output-link I3)
  (S2 ^command C2 ^result R3)
print
```

Figura 22 – Conteúdo memória de trabalho e procedural.

Com estas apresentações fica provado que o conteúdo armazenado semanticamente não tem ligação com outras memórias. Para limpar o armazenamento semântico, devermos usar o comando: *"smem –init",* Figura 23.

```
smem --init
Agent reinitialized.
0 productions excised.
print --depth 100 <s>
(S1 ^epmem E1 ^io I1 ^reward-link R1 ^smem S2 ^superstate nil ^type state)
  (E1 ^command C1 ^present-id 1 ^result R2)
  (I1 ^input-link I2 ^output-link I3)
  (S2 ^command C2 ^result R3)
print
smem --print
```

Figura 23 – Limpando memória semântica.

2.2 Interação de agentes

Agentes interagem com a memória semântica através de estruturas específicas na memória de trabalho. O SOAR cria um *link* automaticamente da memória semântica para cada estado. Esse *link* tem subestruturas especializadas que enviam ações de inicialização para o agente e *feedbacks* da memória semântica. Se notar o resultado do comando "*print --depth 100 <s>*" na Figura 23, é possível observar que a memória de trabalho possui *links* para a memória semântica ("*smem S2*" e suas ramificações).
Através das regras, o agente interage com os *links* e manipula as arquiteturas, preenchendo e/ou limpando. Para o agente interagir com a memória semântica, o mecanismo dos *links* deve ser ativado, pois por *default* no SOAR estão desativados. Para ativar a memória semântica, e consequentemente todo o mecanismo dos *links,* basta utilizar o comando presente na Figura 24.

```
smem --set learning on
```

Figura 24 – Ativando memória semântica.

2.3 Armazenamento e modificação de agentes

Um agente armazena um objeto na memória semântica emitindo um comando *store,* com a sintaxe "*(<cmd> ^store <id>)*" *id* sendo o identificador e o *cmd* o *link* comando chegar a memória semântica (MS). Vale ressaltar que comandos múltiplos são permitidos.
O comando *store,* armazena o identificador que é o resultado do comando. Se o identificador, ainda não foi criado, ele é criado como *long-term.* Mas se ele já foi criado na MS, a expansão sobrescreverá o valor do identificador.
Agora, será apresentado um exemplo onde o código está presente no diretório "*smem_tutorial8*". A Figura 25 apresenta as regras do código exemplo.

```
sp {propose*init
    (state <s> ^superstate nil
               -^name)
 -->
    (<s> ^operator <op> +)
    (<op> ^name init)}

 sp {apply*init
    (state <s> ^operator.name init
               ^smem.command <cmd>)
 -->
    (<s> ^name friends)
    (<cmd> ^store <a> <b> <c>)
    (<a> ^name alice ^friend <b>)
    (<b> ^name bob ^friend <a>)
    (<c> ^name charley)}

sp {propose*mod
   (state <s> ^name friends
              ^smem.command <cmd>)
   (<cmd> ^store <a> <b> <c>)
   (<a> ^name alice)
   (<b> ^name bob)
   (<c> ^name charley)

-->
   (<s> ^operator <op> +)
   (<op> ^name mod)}

 sp {apply*mod
    (state <s> ^operator.name mod
               ^smem.command <cmd>)
    (<cmd> ^store <a> <b> <c>)
    (<a> ^name alice)
    (<b> ^name bob)
    (<c> ^name charley)
 -->
    (<a> ^name alice -)
    (<a> ^name anna
         ^friend <c>)
    (<cmd> ^store <b> -)
    (<cmd> ^store <c> -)}
```

Figura 25 – Exemplo utilização MS.

A Figura 26, apresenta a execução de 3 etapas (em botões), no *SoarJavaDebugger*: *step, watch 5* e *run 1 -p.*

```
source {D:\mateus\Dropbox\Mestrado\IA006\aula6\smem_tutorial8\smem-tutorial.soar}
**********
Total: 10 productions sourced.
step
     1: O: O1 (init)
watch 5
run 1 --phase
--- apply phase ---
--- Firing Productions (PE) For State At Depth 1 ---
Firing apply*init
-->
(C6 ^name charley + :O)
(B2 ^friend A2 + :O)
(B2 ^name bob + :O)
(A2 ^friend B2 + :O)
(A2 ^name alice + :O)
(C5 ^store C6 + :O)
(C5 ^store B2 + :O)
(C5 ^store A2 + :O)
(S1 ^name friends + :O)
--- Change Working Memory (PE) ---
=>WM: (25: C6 ^name charley)
=>WM: (24: B2 ^friend A2)
=>WM: (23: B2 ^name bob)
=>WM: (22: A2 ^friend B2)
=>WM: (21: A2 ^name alice)
=>WM: (20: C5 ^store A2)
=>WM: (19: C5 ^store B2)
=>WM: (18: C5 ^store C6)
=>WM: (17: S1 ^name friends)
--- Change Working Memory (PE) ---
=>WM: (28: R3 ^success @A2)
=>WM: (27: R3 ^success @B2)
=>WM: (26: R3 ^success @C6)
--- Firing Productions (IE) For State At Depth 1 ---
Firing propose*mod
-->
(O2 ^name mod +)
(S1 ^operator O2 +)
Retracting propose*init
-->
(O1 ^name init +)
(S1 ^operator O1 +)
--- Change Working Memory (IE) ---
=>WM: (30: S1 ^operator O2 +)
=>WM: (29: O2 ^name mod)
<=WM: (15: S1 ^operator O1 +)
<=WM: (16: S1 ^operator O1)
<=WM: (14: O1 ^name init)
--- Firing Productions (IE) For State At Depth 1 ---
--- Change Working Memory (IE) ---
```

Figura 26 – Execução exemplo - MS.

Observe que a regra *apply*init* foi selecionada e acrescentou 3 elementos na memória de trabalho, onde os identificadores são inicialmente não *long-term,* e as expansões refletem o conteúdo da memória semântica, item 2.1 deste relatório. Então no final da fase de elaboração, a MS processa o comando, transformando os identificadores em *long-term* e adicionando o valor para cada comando (*link*).

A Figura 27 possui a execução do comando "*smem --print*", note que o resultado da MS possui o mesmo conteúdo depois de usar o comando "*smem --add*" no item 2.1.

```
source {D:\mateus\Dropbox\Mestrado\IA006\aula6\smem_tutorial8\smem-tutorial.soar}
**********
Total: 10 productions sourced.
step
     1: O: O1 (init)
watch 5
run 1 --phase
--- apply phase ---
--- Firing Productions (PE) For State At Depth 1 ---
Firing apply*init
-->
(C3 ^name charley + :O)
(B1 ^friend A1 + :O)
(B1 ^name bob + :O)
(A1 ^friend B1 + :O)
(A1 ^name alice + :O)
(C2 ^store C3 + :O)
(C2 ^store B1 + :O)
(C2 ^store A1 + :O)
(S1 ^name friends + :O)
--- Change Working Memory (PE) ---
=>WM: (25: C3 ^name charley)
=>WM: (24: B1 ^friend A1)
=>WM: (23: B1 ^name bob)
=>WM: (22: A1 ^friend B1)
=>WM: (21: A1 ^name alice)
=>WM: (20: C2 ^store A1)
=>WM: (19: C2 ^store B1)
=>WM: (18: C2 ^store C3)
=>WM: (17: S1 ^name friends)
--- Change Working Memory (PE) ---
=>WM: (28: R3 ^success @A1)
=>WM: (27: R3 ^success @B1)
=>WM: (26: R3 ^success @C3)
--- Firing Productions (IE) For State At Depth 1 ---
Firing propose*mod
-->
(O2 ^name mod +)
(S1 ^operator O2 +)
Retracting propose*init
-->
(O1 ^name init +)
(S1 ^operator O1 +)
--- Change Working Memory (IE) ---
=>WM: (30: S1 ^operator O2 +)
=>WM: (29: O2 ^name mod)
<=WM: (15: S1 ^operator O1 +)
<=WM: (16: S1 ^operator O1)
<=WM: (14: O1 ^name init)
--- Firing Productions (IE) For State At Depth 1 ---
--- Change Working Memory (IE) ---
smem --print
(@A1 ^friend @B1 ^name alice [+1.000])
(@B1 ^friend @A1 ^name bob [+2.000])
(@C3 ^name charley [+3.000])
```

Figura 27 – *Print smem*- MS.

A Figura 28, apresenta a aplicação do próximo operador, em sequência a execução, novamente foi clicado em *step* e depois em *run 1 -p.* Note que o conteúdo da MS é sobrescrito.

```
step
--- output phase ---
--- input phase ---
--- propose phase ---
--- decision phase ---
=>WM: (31: S1 ^operator O2)
      2: O: O2 (mod)
run 1 --phase
--- apply phase ---
--- Firing Productions (PE) For State At Depth 1 ---
Firing apply*mod
-->
(C2 ^store @C3 - :O)
(C2 ^store @B1 - :O)
(@A1 ^friend @C3 + :O)
(@A1 ^name anna + :O)
(@A1 ^name alice - :O)
--- Change Working Memory (PE) ---
=>WM: (33: @A1 ^name anna)
=>WM: (32: @A1 ^friend @C3)
<=WM: (21: @A1 ^name alice)
<=WM: (18: C2 ^store @C3)
<=WM: (19: C2 ^store @B1)
--- Change Working Memory (PE) ---
<=WM: (26: R3 ^success @C3)
<=WM: (27: R3 ^success @B1)
--- Firing Productions (IE) For State At Depth 1 ---
Firing propose*ncb-retrieval
-->
(O3 ^friend @C3 +)
(O3 ^name ncb-retrieval +)
(S1 ^operator O3 =)
(S1 ^operator O3 +)
Firing propose*ncb-retrieval
-->
(O4 ^friend @B1 +)
(O4 ^name ncb-retrieval +)
(S1 ^operator O4 =)
(S1 ^operator O4 +)
Retracting propose*mod
-->
(O2 ^name mod +)
(S1 ^operator O2 +)
--- Change Working Memory (IE) ---
=>WM: (39: S1 ^operator O4 +)
=>WM: (38: S1 ^operator O3 +)
=>WM: (37: O4 ^friend @B1)
=>WM: (36: O4 ^name ncb-retrieval)
=>WM: (35: O3 ^friend @C3)
=>WM: (34: O3 ^name ncb-retrieval)
<=WM: (30: S1 ^operator O2 +)
<=WM: (31: S1 ^operator O2)
<=WM: (29: O2 ^name mod)
```

Figura 28 – Execução 2º *step* exemplo - MS.

É fácil verificar que os comandos *store* foram removidos (ex: C2 ^store @C3 -), e as expansões de @A1 são adicionados e removidos (ex: @a1 ^name anna + / alice -).
Agora novamente vale observar o conteúdo da memória semântica (MS), para notar as mudanças, Figura 29. Note que apenas o objeto @A1 sofreu alteração, condizente com a execução.

```
smem --print
(@A1 ^friend @B1 @C3 ^name anna [+4.000])
(@B1 ^friend @A1 ^name bob [+2.000])
(@C3 ^name charley [+3.000])
```

Figura 29 – *Print smem*- MS – 2º *step*.

2.4 Recuperação *Non-Cue-Based*

A primeira maneira que um agente pode recuperar conhecimento da memória semântica (MS) é utilizando a recuperação *non-cue-based.* O agente requisita da MS todos as expansões de *long-term,* utilizando ao comando coma seguinte sintaxe: "*(<cmd> ^retrieve <lti>)*" onde *lti* é o identificador do *long-term.* O exemplo apresentado a seguir, está seguindo a construção no mesmo código utilizado no item anterior, apenas completando as regras.

```
sp {propose*ncb-retrieval
   (state <s> ^name friends
              ^smem.command <cmd>)
   (<cmd> ^store <a>)
   (<a> ^name anna
        ^friend <f>)
-->
   (<s> ^operator <op> + =)
   (<op> ^name ncb-retrieval
         ^friend <f>)}

sp {apply*ncb-retrieval*retrieve
   (state <s> ^operator <op>
              ^smem.command <cmd>)
   (<op> ^name ncb-retrieval
         ^friend <f>)
   (<cmd> ^store <a>)
-->
   (<cmd> ^store <a> -
          ^retrieve <f>)

sp {apply*ncb-retrieval*clean
   (state <s> ^operator <op>
              ^smem.command <cmd>)
   (<op> ^name ncb-retrieval
         ^friend <f>)
   (<f> ^<attr> <val>)
-->
   (<f> ^<attr> <val> -)}
```

Figura 30 – Recuperando e removendo informações.

Estas regras recuperam um dos dois amigos (de modo aleatório) relacionados a @A1, e remove o mesmo da memória de trabalho. Em oposição aos comandos de *store*, todas as consultas são processadas durante a fase de saída do agente e apenas um comando de recuperação pode ser enviado por estado.
A Figura 31, apresenta a execução dos passos (botões) no *SoarJavaDebugger*: *step* e *run 1 -p, ... , run 1 -p* e *(print --depth 10 S2),* observe a Figura.

```
source {D:\mateus\Dropbox\Mestrado\IA006\aula6\smem_tutorial8\partII.soar}
**********
Total: 10 productions sourced.
step
     1: O: O1 (init)
run 1 --phase
smem --print
(@A1 ^friend @B1 ^name alice [+1.000])
(@B1 ^friend @A1 ^name bob [+2.000])
(@C3 ^name charley [+3.000])
step
     2: O: O2 (mod)
run 1 --phase
smem --print
(@A1 ^friend @B1 @C3 ^name anna [+4.000])
(@B1 ^friend @A1 ^name bob [+2.000])
(@C3 ^name charley [+3.000])
print --depth 10 s2
(S2 ^command C2 ^result R3)
  (C2 ^store @A1)
    (@A1 ^friend @C3 ^friend @B1 ^name anna)
      (@C3 ^name charley)
      (@B1 ^friend @A1 ^name bob)
  (R3 ^success @A1)
run 1 --phase
run 1 --phase
run 1 --phase
run 1 --phase
     3: O: O3 (ncb-retrieval)
print --depth 10 s2
(S2 ^command C2 ^result R3)
  (C2 ^store @A1)
    (@A1 ^friend @C3 ^friend @B1 ^name anna)
      (@C3 ^name charley)
      (@B1 ^friend @A1 ^name bob)
  (R3 ^success @A1)
print --depth 10 s2
(S2 ^command C2 ^result R3)
  (C2 ^store @A1)
    (@A1 ^friend @C3 ^friend @B1 ^name anna)
      (@C3 ^name charley)
      (@B1 ^friend @A1 ^name bob)
  (R3 ^success @A1)
run 1 --phase
run 1 --phase
run 1 --phase
run 1 --phase
run 1 --phase
     4: O: O5 (cb-retrieval)
print --depth 10 s2
(S2 ^command C2 ^result R3)
  (C2 ^retrieve @C3)
    (@C3 ^name charley)
  (R3 ^retrieved @C3 ^success @C3)
```

Figura 31 – Recuperando e removendo informações de um amigo.

Note que para reproduzir o caso do tutorial foi necessário mais *run 1 --phase,* a repetição de escolhas nesse caso é um pouco complicada, devido a escolha do operador inicial.

2.5 Recuperação *Cue-Based*

A segunda maneira que um agente pode recuperar o conhecimento da memória semântica (MS) é chamado de recuperação *cue-based.* O agente requisita da MS todas as expansões de um identificador desconhecido de *long-term*. O comando utilizado para tal ação é "*(<cmd> ^query <cue>)*" onde os identificadores desejados correspondem a *<cue>*.

Como exemplo, as regras presentes na Figura 32, serão adicionadas as regras dos dois últimos itens, tendo o código presente no mesmo diretório.

```
sp {propose*cb-retrieval
   (state <s> ^name friends
              ^smem.command <cmd>)
   (<cmd> ^retrieve)
-->
   (<s> ^operator <op> + =)
   (<op> ^name cb-retrieval)}

sp {apply*cb-retrieval
   (state <s> ^operator <op>
              ^smem.command <cmd>)
   (<op> ^name cb-retrieval)
   (<cmd> ^retrieve <lti>)
-->
   (<cmd> ^retrieve <lti> -
          ^query <cue>)
   (<cue> ^name <any-name>
          ^friend <lti>)}
```

Figura 32 – Regras recuperação *cue-based*.

Estas regras recuperam um identificador que tenha duas restrições: a primeira é uma expansão que o atributo é do tipo *name,* mas o valor pode ser qualquer tipo; a segunda é uma expansão que o atributo é do tipo *friend* e o valor é *long-term*.
Vale lembrar que apenas um comando de recuperação pode ser emitido por estado e as consultas são processadas durante a fase de saída do agente.
A Figura 33, apresenta a execução do código com as novas regras, usando *steps* e *run 1 -p,* até a finalização do agente e utiliza o comando *print –depth 10 s2*.

```
source {D:\mateus\Dropbox\Mestrado\IA006\aula6\smem_tutorial8\smem-tutorial.soar}
**********
Total: 10 productions sourced.
step
     1: O: O1 (init)
run 1 --phase
step
     2: O: O2 (mod)
step
     3: O: O3 (ncb-retrieval)
step
     4: O: O5 (cb-retrieval)
step
Interrupt received.
This Agent halted.

An agent halted during the run.
print --depth 10 s2
(S2 ^command C2 ^result R3)
  (C2 ^query C4)
    (C4 ^friend @C3 ^name A2)
      (@C3 ^name charley)
  (R3 ^retrieved @A1 ^success C4)
    (@A1 ^friend @B1 ^friend @C3 ^name anna)
```

Figura 33 – Execução código com recuperação *cue-based*.

Note que a memória semântica tem recuperado e adicionado o identificador @A1 a memória de trabalho. Não possuiu nenhum identificador *long-term* na memória semântica que satisfez as

restrições do comando *query (consulta) cue*. Note também que os conhecimentos obtidos são limitados as expansões de identificadores *long-term*, como comando *store* a recuperação não é recursiva.
Se vários identificadores satisfazerem as restrições de *cue, cue* possivelmente terá apenas um elemento na memória de trabalho com tipo *name* e um identificador *short-term,* então o identificador *long-term* com maior valor será retornado. Também é possível bloquear um ou mais identificadores de *long-term* de serem recuperados.

3 Atividade 3

Neste tópico será apresentado a execução do tutorial 9 do SOAR, onde é abordado o tema de memória episódica. Em SOAR é um mecanismo que automaticamente: armazena, captura, indexa temporariamente o estado do agente e consegue através de mecanismos extrair proveitos destes recursos. Os códigos referentes a este item estão no diretório "*epmem_tutorial9*".

3.1 Uma breve demonstração

Antes de explorar a maneira de como um agente usa a memória episódica, será feito um exemplo simples de captura de um episódio e visualização do mesmo na memória. Todos os códigos referentes a este tópico estão presentes no diretório "aula6".
A Figura 34 apresenta três comandos que devem ser executados no *SoarJavaDebugger*.

```
epmem --set trigger dc
epmem --set learning on
watch --epmem
step
     1: ==>S: S3 (state no-change)
step
NEW EPISODE: 1
     2:    ==>S: S5 (state no-change)
```

Figura 34 – Comandos memória episódica - I.

Note que a penúltima linha de execução possui uma expressão de "*NEW EPISODE: 1*", isto significa que foi identificado um novo episódio e o mesmo foi armazenado na estrutura/arquitetura de armazenamento de episódios e possui o *id* igual a 1.
É possível visualizar o conteúdo da memória episódica através do seguinte comendo: "*epmem –print idMem*", onde *idMem* é o *id* do episódio desejado, Figura 35.

```
epmem --print 1
(<id0> ^io <id1> ^reward-link <id2> ^superstate nil ^type state)
(<id1> ^input-link <id4> ^output-link <id3>)
```

Figura 35 – *Print* episódio 1.

Como no tutorial anterior, também é possível visualizar a memória episódica graficamente, através de comandos presentes na Figura 36, o resultado da combinação é a saída, para o diretório corrente, de um arquivo que pode ser interpretado pelo programa *Graphviz (http://graphviz.org)* que renderiza o código para um diagrama, Figura 37. Para descobrir o diretório corrente, basta executar o comando "*pwd*".

```
command-to-file epmem.gv epmem --viz 1
```

Figura 36 – Geração de arquivo diagrama.

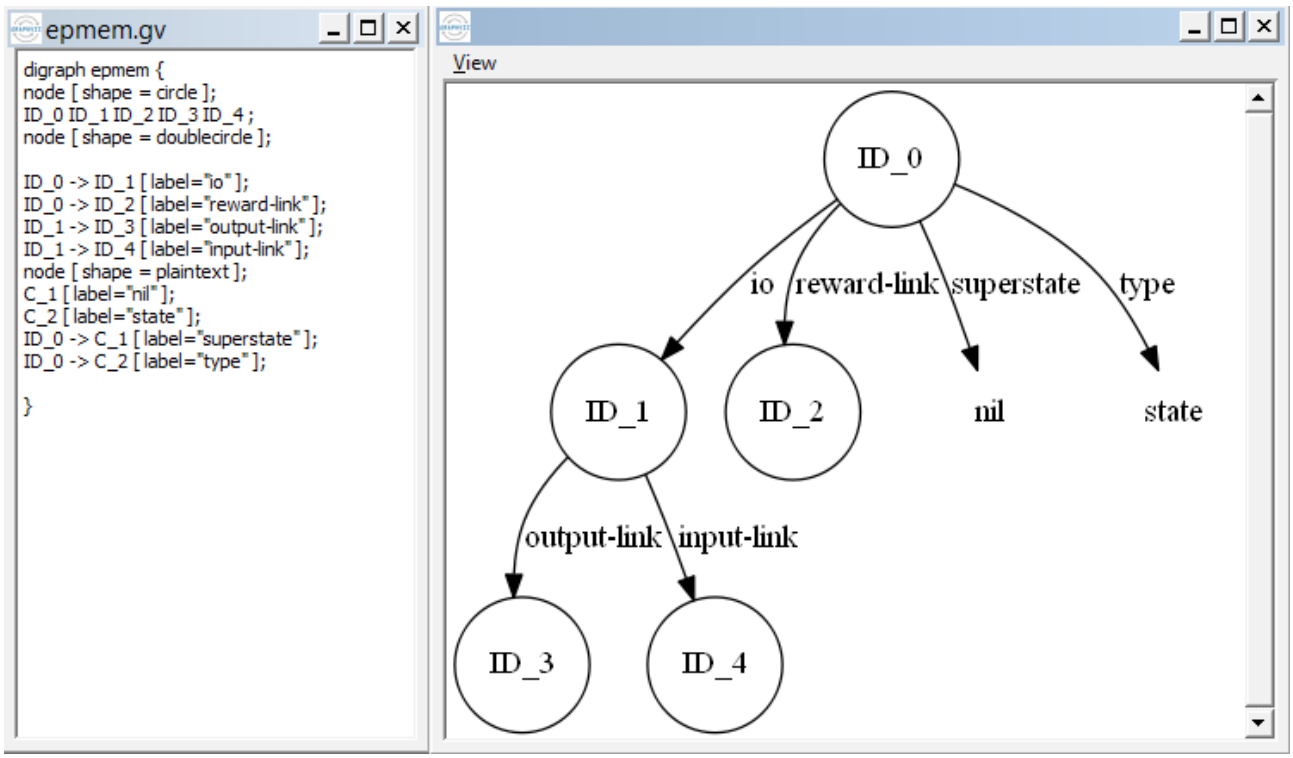

Figura 37 – Diagrama exemplo *epmem*.

Das duas maneiras é possível ver o conteúdo da memória episódica do agente em um determinado momento do tempo. Os próximos itens, abordam com mais detalhes como controlar o armazenamento automático de episódios e como os agentes podem recuperar o conhecimento episódico.

3.2 Armazenamento episódico

Sabe-se que a captura de episódios é automática e a captura armazena o estado superior da memória de trabalho do agente. Para desfrutar de tal mecanismo, basta habilitá-lo conforme o comando a seguir:

```
epmem --set learning on
```

Como já foi dito, por padrão os mecanismos de aprendizagem do SOAR são desabilitados. Existem alguns parâmetros importantes que controlam o armazenamento episódico, o primeiro é o gatilho (acontecimento) que desencadeia o armazenamento. Por *default* o SOAR a cada vez que adiciona um elemento na memória de trabalho, armazena um novo episódio, utilizando o *output-link* como identificador do episódio. Outra forma de armazenamento de episódio é através do mecanismo de ciclo de decisão, que pode ser habilitado pelo seguinte comando:

```
epmem --set trigger dc
```

O próximo parâmetro é fazem em que o episódio é armazenado. Por *default* o SOAR este processo ocorre no final da fase de produção, porém também é possível realizar tal tarefa no final da faze de decisão, ativando utilizando o seguinte comando:

```
epmem --set phase selection
```

Nem sempre os episódios armazenados automaticamente são desejados, por isso a memória episódica suporta a seleção de atributos para serem deletados. Caso a seleção automática de episódios encontre um elemento excluído não armazena esse elemento e nem subestruturas desse elemento. Para visualizar o conjunto de elementos excluídos basta utilizar o comando presente na

Figura 38.

```
epmem --get exclusions
epmem, smem
```

Figura 38 – Elementos excluídos.

Para alterar o conjunto de elementos excluídos basta executar o seguinte comando:

```
epmem --set exclusions <attribute>
```

Este comando segue a seguinte regra, se o atributo referenciado não existe no conjunto dos episódios excluídos, então o atributo é adicionado a exclusão. Mas se o atributo já pertence ao conjunto e o comando é chamado, o atributo deixa de fazer parte do conjunto. Por *default* o *epmem* e o *smem* fazem parte do conjunto dos episódios excluídos.
Para o rastreamento do da saída da memória episódica deve-se utilizar o comando:

```
watch --epmem
```

Este comando indica novos episódios gravados, podendo auxiliar nas depurações de consultas.

3.3 Interação com agente

Agentes interagem com a memória episódica através de estruturas especiais na memória de trabalho. O SOAR liga automaticamente os *links epmem* em cada estado, e cada *link epmem* tem uma subestrutura especializada. A Figura 39 possui o resultado da execução do comando "*print --depth 10 <s>*".

```
print --depth 10 <s>
(S5 ^attribute state ^choices none ^epmem E3 ^impasse no-change ^quiescence t
       ^reward-link R7 ^smem S6 ^superstate S3 ^type state)
  (E3 ^command C5 ^present-id 2 ^result R8)
  (S6 ^command C6 ^result R9)
  (S3 ^attribute state ^choices none ^epmem E2 ^impasse no-change
         ^quiescence t ^reward-link R4 ^smem S4 ^superstate S1 ^type state)
    (E2 ^command C3 ^present-id 2 ^result R5)
    (S4 ^command C4 ^result R6)
    (S1 ^epmem E1 ^io I1 ^reward-link R1 ^smem S2 ^superstate nil ^type state)
      (E1 ^command C1 ^present-id 2 ^result R2)
      (I1 ^input-link I2 ^output-link I3)
      (S2 ^command C2 ^result R3)
```

Figura 39 – Elementos excluídos.

Como visto, o agente preenche/limpa e mantém através de regras o *link* de resultado. Como os episódios são armazenados, as atualizações do aumento *present-id* indicam o atual *id* do episódio, cujo valor é um número inteiro.
Para que exista interação com o agente e a memória episódica, esse mecanismo deve ser ativado ("*epmem --set learning on*"). Sabe-se que todos os mecanismos de aprendizagem em SOAR, por *default* são desabilitados e por isso se deve ativar o mecanismo de memória episódica através do mesmo comando ("*epmem --set learning on*").

3.4 Recuperação *Cue-Based*

O principal método que um agente pode recuperar o conhecimento da memória episódica é chamado recuperação *cue-based:* o agente requisita da memória episódica um episódio que mais se aproxima dos elementos da memória de trabalho. Para realizar esta ação basta executar o comando: "*(<cmd> ^query <cue>)*" onde *<cue>* é a raiz, do diagrama da memória episódica.

Resumidamente, a memória episódica retorna o episódio mais recente com maior pontuação do conjunto de episódios.
A folha WME está satisfeita em relação a um episódio particular se existe um caminho ou uma sequência de WMEs a partir da raiz (considerando o episódio como raiz) para aquela folha. De forma mais clara, a folha está satisfeita, se existe um caminho no diagrama da memória de trabalho que parte do episódio e chega a ela mesma. Por padrão, a pontuação do episódio é simples: o número de folhas satisfeitas nos elementos da memória de trabalho.
Considerando os WMEs compondo a Figura 40, onde N1 é o valor de comando da consulta.

```
(N1 ^feature value
    ^id N2)
(N2 ^sub-feature value2
    ^sub-id N3)
```

Figura 40 – Exemplo WMEs.

A Figura 41, apresenta o grafo da estrutura presente na Figura 40.

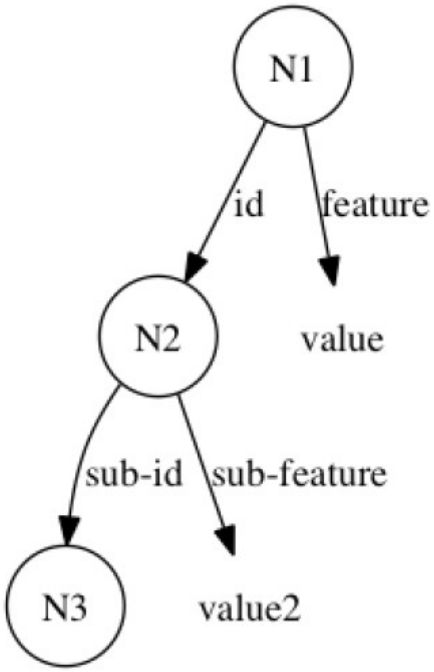

Figura 41 – Representação gráfica.

É fácil notar que a estrutura apresentada possui apenas 3 folhas, pois: [N1 ^*feature value*], [N2 ^*sub-feature value2*] e [N2 ^*sub-id* N3].
Agora, considerando a estrutura presente na Figura 42, como episódio. A primeira folha [N1 ^*feature value*] não é satisfeita pelo episódio, pois [E1 ^*feature value3*] e [E1 ^feature2 value] não coincide co a folha. Para a segunda folha [N2 ^*sub-feature value2*] é satisfeita pelo episódio, pois [E1 ^*id* E2] e [N1 ^*id* N2], e [E2 ^*sub-feature value2*] e [N2 ^*sub-feature value2*], possui caminho. E a última folha também é satisfeita pelo episódio, pois [E1 ^*id* E3] e [N1 ^*id* N2], e [E3 ^*sub-id* E5] e [N2 ^*sub-id* N3], também possui caminho. Então este episódio possui 2 pontos de critério.

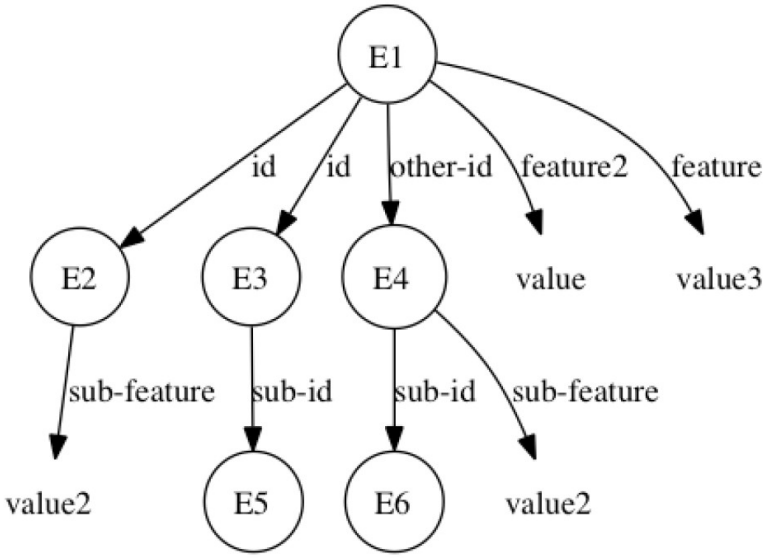

Figura 42 – Exemplo II.

No entanto, note que não é possível unificar o *cue* com o episódio, pois não existe um único operador que quando optado por N2 satisfaça [N2 ^*sub-feature value2*] e [N2 ^*sub-id* N3]. Se um episódio receber uma perfeita pontuação cada folha do WMEs será satisfeita, tentando verificar isomorfismo entre a sinalização e o episódio.

A Figura 43, apresenta o código presente no diretório "*epmem_tutorial9*", o mesmo presente no diretório agentes.

```
sp {propose*init
   (state <s> ^superstate nil
              -^name)
-->
   (<s> ^operator <op> + =)
   (<op> ^name init)}

sp {apply*init
   (state <s> ^operator <op>)
   (<op> ^name init)
-->
   (<s> ^name epmem
        ^feature2 value
        ^feature value3
        ^id <e2>
        ^id <e3>
        ^other-id <e4>)
   (<e2> ^sub-feature value2)
   (<e3> ^sub-id <e5>)
   (<e4> ^sub-id <e6>
         ^sub-feature value2)}

sp {epmem*propose*cbr
   (state <s> ^name epmem
              -^epmem.command.<cmd>)
-->
   (<s> ^operator <op> + =)
   (<op> ^name cbr)}

sp {epmem*apply*cbr-clean
   (state <s> ^operator <op>
              ^feature2 <f2>
              ^feature <f>
              ^id <e2>
              ^id <e3>
              ^other-id <e4>)
   (<e2> ^sub-feature value2)
   (<e3> ^sub-id)
   (<op> ^name cbr)
-->
   (<s> ^feature2 <f2> -
        ^feature <f> -
        ^id <e2> -
        ^id <e3> -
        ^other-id <e4> -)}

sp {epmem*apply*cbr-query
   (state <s> ^operator <op>
              ^epmem.command <cmd>)
   (<op> ^name cbr)
-->
   (<cmd> ^query <n1>)
   (<n1> ^feature value
         ^id <n2>)
   (<n2> ^sub-feature value2
         ^sub-id <n3>)}
```

Figura 43 – Código recuperação *cue-based*.

Agora, será apresentado a execução deste código no *SoarJavaDebugger*. A Figura 44 contém os parâmetros necessários para a inicialização.

```
source {D:\mateus\Dropbox\Mestrado\IA006\aula6\epmem_tutorial9\epmem-tutorial.soar}
********
Total: 8 productions sourced.
epmem --set trigger dc
epmem --set learning on
watch --epmem
```

Figura 44 – Inicialização recuperação *cue-based*.

A Figura 45 apresenta a execução dos respectivos comandos: "s*tep*", "*run 1 -p*" e "*print --depth 10 s1*".

```
source {D:\mateus\Dropbox\Mestrado\IA006\aula6\epmem_tutorial9\epmem-tutorial.soar}
********
Total: 8 productions sourced.
epmem --set trigger dc
epmem --set learning on
watch --epmem
step
     1: O: O1 (init)
run 1 --phase
print --depth 10 s1
(S1 ^epmem E1 ^feature value3 ^feature2 value ^id E2 ^id E3 ^io I1 ^name epmem
       ^operator O2 + ^other-id E4 ^reward-link R1 ^smem S2 ^superstate nil
       ^type state)
  (E1 ^command C1 ^present-id 1 ^result R2)
  (E2 ^sub-feature value2)
  (E3 ^sub-id E5)
  (I1 ^input-link I2 ^output-link I3)
  (O2 ^name cbr)
  (E4 ^sub-feature value2 ^sub-id E6)
  (S2 ^command C2 ^result R3)
```

Figura 45 – Execução parcial - I.

A Figura 46 apresenta a execução dos respectivos comandos: "s*tep*", "*run 1 -p*" e "*print --depth 10 s1*". Mas agora, note que foi identificado o primeiro episódio.

```
step
NEW EPISODE: 1
     2: O: O2 (cbr)
run 1 --phase
print --depth 10 s1
(S1 ^epmem E1 ^io I1 ^name epmem ^operator O3 + ^reward-link R1 ^smem S2
       ^superstate nil ^type state)
  (E1 ^command C1 ^present-id 2 ^result R2)
    (C1 ^query N1)
      (N1 ^feature value ^id N2)
        (N2 ^sub-feature value2 ^sub-id N3)
  (I1 ^input-link I2 ^output-link I3)
  (O3 ^name next)
  (S2 ^command C2 ^result R3)
```

Figura 46 – Execução parcial - II.

A Figura 47 apresenta a execução do comando: "*run 1 -p*". Note que o segundo episódio é criado e depois foi processada a consulta *cue-based.*

```
run 1 --phase
NEW EPISODE: 2
CONSIDERING EPISODE (time, cardinality, score): (1, 2, 2.000000)
NEW KING (perfect, graph-match): (false, false)
```

Figura 47 – Execução parcial - III.

A terceira linha da Figura 47, apresenta que a memória episódica em comparação a sugestão de episódio 1, constatou cardinalidade 2 e score 2. Como esse foi o primeiro episódio considerado, o

mesmo foi eleito como “rei”. Porém o episódio eleito não teve uma boa pontuação, sendo eliminado para a memória episódica não considerá-lo como uma otimização.
A Figura 48, apresenta o *print* de todo conteúdo, até do *link* de memória episódica.

```
print --depth 10 e1
(E1 ^command C1 ^present-id 3 ^result R2)
  (C1 ^query N1)
    (N1 ^feature value ^id N2)
      (N2 ^sub-feature value2 ^sub-id N3)
  (R2 ^cue-size 3 ^graph-match 0 ^match-cardinality 2 ^match-score 2.
        ^memory-id 1 ^normalized-match-score 0.6666666666666666 ^present-id 3
        ^retrieved R4 ^success N1)
    (R4 ^feature value3 ^feature2 value ^id I6 ^id I5 ^io I4 ^name epmem
          ^operator* O5 ^other-id O4 ^reward-link R5 ^superstate nil
          ^type state)
      (I6 ^sub-id S3)
      (I5 ^sub-feature value2)
      (I4 ^input-link I7 ^output-link O6)
      (O5 ^name cbr)
      (O4 ^sub-feature value2 ^sub-id S4)
```

Figura 48 – *Print* estrutura execução.

A partir do resultado apresentado na Figura 48, é possível observar que a recuperação foi bem-sucedida, tem *link* para o conteúdo episódio completo (sendo R4 a raiz). Note que a WME com um operador de atributo (R4 ^*operador* R5) em um episódio recuperado, representa uma preferência aceitável WME em um episódio.
Existem modificadores opcionais para consultas *cue-based,* incluindo a capacidade para proibir episódios específicos de serem recuperados e indicando as características que não são desejáveis no episódio recuperado.

3.5 Progressão temporal

Outra forma do agente ter acesso aos episódios é recuperando o episódio que veio temporalmente antes ou depois do último episódio, que foi recuperado. Os comandos necessários são: *(<cmd> ^previous <id>)* e *(<cmd> ^next <id>),* sendo *<id>* é qualquer identificador.
A Figura 49 apresenta duas regras, relacionadas a recuperação temporal, que estão no código presente no diretório “*epmem_tutorial9*”.

```
sp {epmem*propose*next
   (state <s> ^name epmem
              ^epmem.command.query)
-->
   (<s> ^operator <op> + =)
   (<op> ^name next)}

sp {epmem*apply*next
   (state <s> ^operator <op>
              ^epmem.command <cmd>)
   (<op> ^name next)
   (<cmd> ^query <q>)
-->
   (<cmd> ^query <q> -
          ^next <next>)}
```

Figura 49– Regras progressão temporal.

As regras presentes na Figura 49, viabilizam a recuperação dos episódios de forma temporal. Note que o comando *query* foi substituído pelo comando *next.* Continuando a execução do item anterior a Figura 50, apresenta o resultado da execução dos seguintes comandos: "*step*", "*run 1 -p*" e "*print --depth 10 e1*".

```
step
     3: O: O3 (next)
run 1 --phase
print --depth 10 e1
(E1 ^command C1 ^present-id 3 ^result R2)
  (C1 ^next N4)
  (R2 ^cue-size 3 ^graph-match 0 ^match-cardinality 2 ^match-score 2.
         ^memory-id 1 ^normalized-match-score 0.6666666666666666 ^present-id 3
         ^retrieved R4 ^success N1)
    (R4 ^feature value3 ^feature2 value ^id I6 ^id I5 ^io I4 ^name epmem
           ^operator* O5 ^other-id O4 ^reward-link R5 ^superstate nil
           ^type state)
      (I6 ^sub-id S3)
      (I5 ^sub-feature value2)
      (I4 ^input-link I7 ^output-link O6)
      (O5 ^name cbr)
      (O4 ^sub-feature value2 ^sub-id S4)
    (N1 ^feature value ^id N2)
      (N2 ^sub-feature value2 ^sub-id N3)
```

Figura 50– Recuperação por progressão temporal – *I*.

Executando novamente os comandos "*run 1 -p*" e "*print --depth 10 e1*" se obtém o resultado presente na Figura 51.

```
run 1 --phase
NEW EPISODE: 3
print --depth 10 e1
(E1 ^command C1 ^present-id 4 ^result R2)
  (C1 ^next N4)
  (R2 ^memory-id 2 ^present-id 4 ^retrieved R6 ^success N4)
    (R6 ^io I8 ^name epmem ^operator* O7 ^reward-link R7 ^superstate nil
           ^type state)
      (I8 ^input-link I9 ^output-link O8)
      (O7 ^name next)
```

Figura 51– Recuperação por progressão temporal – *II*.

A estrutura resultado foi limpa, velhos resultados foram removidos e a função *command* teve sucesso na execução e um terceiro episódio foi recuperado.

4 Conclusão

Com a execução destas atividades presentes nesse relatório, já é possível ter noções básicas de manipulação de aprendizagem por reforço, armazenamento semântico e armazenamento episódico. Concluindo, para projetos estas ferramentas são de grande valia para atacar problemas específicos, sendo indispensáveis em alguns casos.