Aula 6
Atividade 2

Soar Tutorial 8 – *Episodic Memory*

1) Introdução

Em SOAR, o mecanismo de Memória Episódica (ME) é uma forma de armazenar o estado de um agente em um determinado momento de sua execução. A arquitetura faz este armazenamento automaticamente, bastando ativar esta capacidade com o comando

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

Para fazer melhor uso desta ferramenta, o usuário da arquitetura pode configurar seu agente de forma a determinar quando e que parte do estado armazenar em cada episódio.

2) Configurações

A princípio, este mecanismo captura **todas as propriedades** do estado superior do agente automaticamente a cada vez que um **elemento com o identificador *output-link* é adicionado à memória de trabalho** e **ao final da fase de *output*** do agente. Contudo, podemos modificar estas configurações através dos seguintes comandos:

`epmem --set trigger dc`	Passa a realizar o armazenamento dos episódios todo ciclo de decisão (dc)
`epmem --set phase selection`	Passa a realizar o armazenamento ao final da fase de decisão
`epmem --set exclusions <attribute>` [1]	Inclui a propriedade de nome <attribute> na lista de exclusão de propriedades armazenadas nos episódios ou a remove se já estiver inclusa
`watch --epmem`	Faz com que seja exibido um texto no output sempre que um novo episódio for armazenado

3) Interação do agente com a ME

Assim como o caso da Memória Semântica, para se interagir com a memória episódica utiliza-se augumentações especiais criadas em cada estado. A impressão do estado de um agente antes de qualquer execução nos dá:

```
(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)
```

[1] Para se visualizar as propriedades que constam no conjunto de exclusão utiliza-se o comando `epmem --get exclusions`

Os elementos destacados em negrito são a interface que utilizamos para interagir com o mecanismo de memória episódica. Assim como no caso da memória semântica, utilizamos o *command-link* (C1 acima) para sinalizar à arquitetura que um comando deve ser executado. Por sua vez, a arquitetura popula o *result-link* com os resultados dos comandos. A propriedade *present-id*[2]*,* um inteiro positivo, representa o id do episódio atual (ou próximo episódio a ser armazenado pela arquitetura) e é incrementado pela arquitetura a cada armazenamento.

Uma vez que a arquitetura armazena automaticamente os episódios de acordo com as configurações fornecidas pelo usuário, não existe necessidade de um comando para implicitamente realizar o armazenamento, como no caso do comando *store* no contexto da memória semântica. Estudaremos principalmente o uso mais importante desta ferramenta: o comando *query*.

4) O comando *query*

A função deste comando é similar à daquele de mesmo nome no contexto de memória semântica: dado um elemento da memória de trabalho (chamaremos esta estrutura de *cue)*, a arquitetura retorna um episódio que apresente similaridades com o mesmo. É importante notar que não é necessário que a estrutura de um episódio seja idêntica à estrutura da *cue* para ser um retorno válido. Para determinar o que retornar, a arquitetura compara a cue com todos os episódios da ME e atribui uma pontuação para cada um deles, retornando, através do *result-link*, aquele com maior pontuação.

Esta pontuação se baseia primariamente nos elementos folha da *cue*, onde uma folha é um elemento da memória de trabalho que possui um valor que é constante, um identificador de longo prazo, ou um identificador de curto prazo sem augumentações.

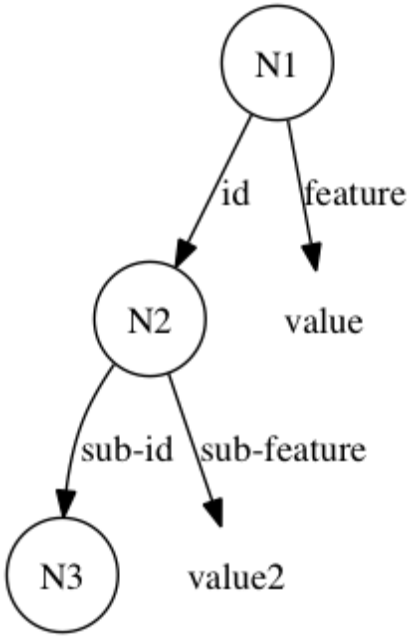
Uma folha é da cue é considerada compatível com uma folha do episódio (marcando assim, um ponto na pontuação do episódio) se existe um caminho da raiz do episódio até a folha episódio que seja equivalente a um caminho da raiz da *cue* até a folha da cue em questão. Um caminho do episódio é equivalente a um caminho da cue se cada um dos elementos intermediários no caminho da raiz do episódio à sua folha possuírem os mesmos valores do caminho na *cue.*

Para compreender melhor este conceito, o melhor a se fazer é utilizar uma visualização em um grafo. Consideremos a seguinte estrutura de uma *cue*:

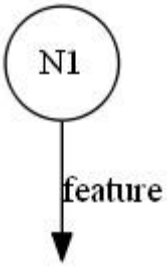
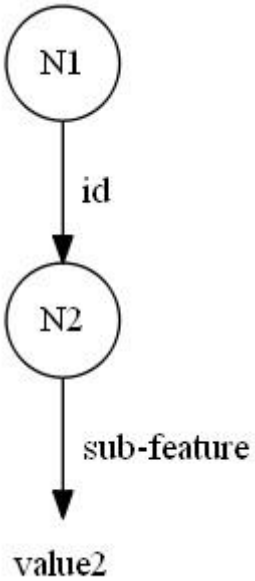
```
(N1 ^feature value
    ^id N2)
(N2 ^sub-feature value2
    ^sub-id N3
```

[2] No SOAR Java Debugger, utiliza-se o comando epmem --print <id> para se visualizar o que foi armazenado no episódio identificado por <id>

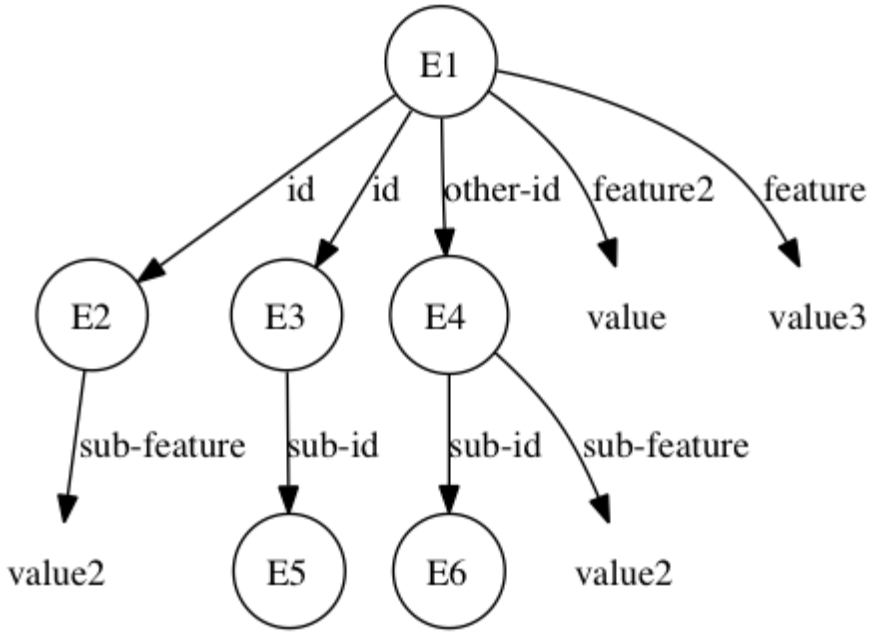
Podemos representa-la através do seguinte grafo:



As folhas desta *cue* e os caminhos da raiz até elas são:

N1 ^feature value	N2 ^sub-id N3	N2 ^sub-feature value2
		

São exatamente estes caminhos que a arquitetura irá buscar nos episódios armazenados na ME, onde os identificadores N1, N2 e N3 são de curta duração e são considerados como variáveis no caminho. Consideremos agora o seguinte episódio hipotético:



Como N1 é a raiz da *cue*, o valor que ele tomará é E1. A partir daí, os caminhos do episódio são explorados de forma a encontrar os caminhos da *cue* até suas folhas.

A primeira folha, `N1 ^feature value`, não é compatível, pois a única aresta de nome 'feature' que parte de E1 possui valor *value3.*

A segunda folha, `N2 ^sub-id N3`, é compatível quando N2 toma o valor E3 e N3 o valor E5, marcando então um ponto para este episódio.

A terceira folha, `N2 ^sub-feature value2`, também é compatível quando N2 toma o valor E2, marcando o segundo ponto para a avaliação deste episódio.

Se a primeira folha também fosse compatível teríamos uma pontuação perfeita. Contudo, é fácil verificar pelas imagens que, apesar disso, o grafo da *cue* não é um subgrafo do grafo que representa o episódio. Se o grafo que representa um episódio da ME for um supergrafo do grafo que representa a *cue* ele terá prioridade sobre episódios com pontuação perfeita que não o são. O critério de desempate final é o id do episódio, que representa sua ordem de criação. Entre dois episódios de pontuação perfeita, por exemplo, será preferido aquele mais recente, portanto de id maior.

5) Busca temporal

Além da busca baseada em *cues*, é possível recuperar os episódios imediatamente anterior e imediatamente posterior ao último episódio recuperado. A sintaxe para estes comandos são:

```
(<cmd> ^previous <id>)
(<cmd> ^next <id>)
```

Onde <id> são identificadores quaisquer. Como sempre, os resultados serão retornados pela arquitetura através do *result-link* da ME.

6) Exemplo

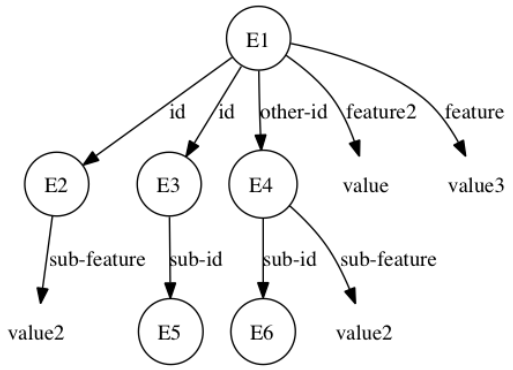
O tutorial fornece uma aplicação simples que serve de exemplo para a aplicação desta ferramenta. A seguir, o código da aplicação e comentários relativos.

`epmem --set trigger dc`
`epmem --set learning on`
`watch --epmem`
	Ativa o mecanismo de memória episódica e configura para que um episódio seja armazenado a cada ciclo de decisão

`sp {propose*init`
`(state <s> ^superstate nil`
`-^name)`
`-->`
`(<s> ^operator <op> + =)`
`(<op> ^name init)`
`}`
	Proposição do operador para inicializar o agente

`sp {apply*init`
`(state <s> ^operator <op>)`
`(<op> ^name init)`
	Aplicação do operador de inicialização

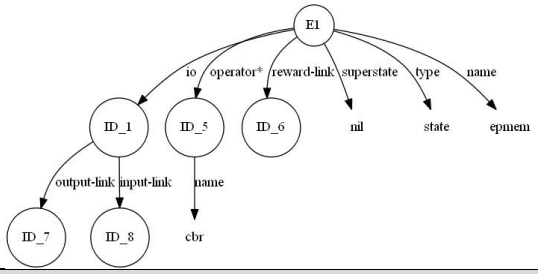
`-->`	

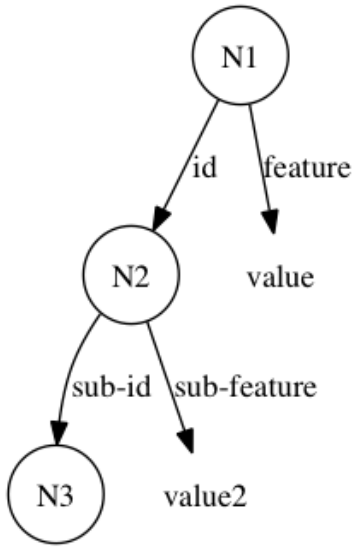
`(<s> ^name epmem`
`^feature2 value`
`^feature value3`
`^id <e2>`
`^id <e3>`
`^other-id <e4>)`
`(<e2> ^sub-feature value2)`
`(<e3> ^sub-id <e5>)`
`(<e4> ^sub-id <e6>`
`^sub-feature value2)`
	Muda o nome do estado para 'epmem' e cria uma estrutura inicial com a seguinte representação gráfica:
	 

`}`	

Ao final da fase de *output* deste ciclo um novo episódio foi criado. Executando o comando epmem --print 1 obtemos:

```
<id0> ^feature value3 ^feature2 value ^id <id2> <id3> ^io <id1> ^name epmem ^operator*
<id5> ^other-id <id4> ^reward-link <id6> ^superstate nil ^type state)
(<id1> ^input-link <id8> ^output-link <id7>)
(<id2> ^sub-feature value2)
(<id3> ^sub-id <id9>)
(<id4> ^sub-feature value2 ^sub-id <id10>)
(<id5> ^name cbr)
```

Podemos ver que o elemento <id0> é o estado superior, com todos os elementos adicionados durante a aplicação de init, além dos elementos criados automaticamente pera arquitetura (*input-link* <id1>, operador <id5> *reward-link* <id6>, superestado, etc)	
sp {epmem*propose*cbr (state <s> ^name epmem -^epmem.command. <cmd>) --> (<s> ^operator <op> + =) (<op> ^name cbr) }	Propõe um operador cbr que realizará uma busca baseada em *cue* na ME.
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) -->	Aplicação 'clean' do operador cbr que, por design, será ativada no ciclo seguinte ao que foi aplicado o operador *init*
(<s> ^feature2 <f2> - ^feature <f> - ^id <e2> - ^id <e3> - ^other-id <e4> -) }	Remove todas as augumentações do estado <s> feitas pelo operador *init*. O estado assume a seguinte representação gráfica 
sp {epmem*apply*cbr-query (state <s> ^operator <op> ^epmem.command <cmd>) (<op> ^name cbr)	Aplicação 'cbr-query' do operador cbr que, por design, será ativada no ciclo seguinte ao que foi aplicado o operador *init*
--> (<cmd> ^query <n1>) (<n1> ^feature value ^id <n2>) (<n2> ^sub-feature value2 ^sub-id <n3>) }	Insere no *command-link* a busca pela cue representada pela seguinte estrutura:



Ao final da fase de output deste ciclo, um novo episódio foi armazenado na ME e o comando *query* foi executado. O output no console ao fim do armazenamento e busca é:

NEW EPISODE: 2
CONSIDERING EPISODE (time, cardinality, score): (1, 2, 2.000000)

Significando que um novo episódio de id 2 foi criado e que durante a busca foi considerado o episódio de id 1 (time, 1), que obteve um total de 2 folhas compatíveis (cardinality, 2) e um score final de 2 (score, 2).
O episódio 2 não chegou a ser considerado para uma busca pois nenhuma folha era compatível.

Ao imprimirmos o result-link da ME, obtemos o seguinte output:

```
(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)
```

Os valores em azul representam o que foi resgatado da ME como resultado da query. Os valores em vermelho são a os valores presentes na *cue* fornecida para query. Como esperado, o episódio 1, armazenado imediatamente após o ciclo onde ocorreu a inicialização, foi retornado pela busca.

`sp {epmem*propose*next`
`(state <s> ^name epmem`
`^epmem.command.query)`
`-->`
`(<s> ^operator <op> + =)`
	Propõe o operador 'next' que irá buscar na ME o episódio imediatamente posterior ao recuperado no ciclo anterior.

(<op> ^name next)	
}	
sp {epmem*apply*next	Aplicação do operador `next`
(state <s> ^operator <op>	
^epmem.command <cmd>)	
(<op> ^name next)	
(<cmd> ^query <q>)	
-->	
(<cmd> ^query <q> -	Remove o comando *query* do *command-link* adicionado no ciclo anterior pelo operador cbr e adiciona o comando next com um identificador <next> qualquer (conforme pede a sintaxe do comando)
^next <next>)	
}	

Ao final da fase de output, a arquitetura irá realizar a busca pelo episódio seguinte, no caso o episódio 2 armazenado após a limpeza do estado através do operador cbr na aplicação 'clean'. A impressão do *result-link* da ME neste momento é:

```
(R2 ^memory-id 2 ^present-id 4 ^retrieved R6 ^success N4)
    (R6 ^io I8 ^name epmem ^operator* O7 ^reward-link R
        ^superstate nil ^type state)
        (I8 ^input-link I9 ^output-link O8)
        (O7 ^name next)
```

O identificador R6 recuperado é justamente o estado que obtivemos após a limpeza.

sp {epmem*done	Operador utilizado para pausar a execução do agente
(state <s> ^name epmem	
^epmem <epmem>)	
(<epmem> ^command.next <next>	
^result.success <next>)	
-->	
(halt)	

7) Conclusão

A execução deste tutorial resulta na compreensão da utilização básica do mecanismo de memória episódica da arquitetura SOAR. A arquitetura mascara toda a complicação existente por trás do armazenamento e busca de episódios que um usuário teria que enfrentar para implementar a funcionalidade, além de permitir certo nível de customização do comportamento dos mesmos.