Mundo do Wumpus

Mundo de Wumpus - agentes

- O agente do Mundo Wumpus deve ter capacidade de raciocínio lógico
- Três tipos de agentes lógicos:
  - **agente reativo** (classificação de percepção e ação)
  - **agentes baseados em modelo** (possuem representação interna do mundo)
  - **agentes baseados em goal** (especificam goals e tentam satisfazê-los)

Interface

- Ações como funções*:*
  - *Vira(Direita), Vira(Esquerda), ParaFrente, Pega, Atira, Solta, Escala*
- Sentenças deverão incluir percepção e o tempo que ocorreram. O agente deve guardar todas as suas percepções do presente e passado. Ex.:

  Percebe([*Cheiro, Brisa, Brilho, Nada, Nada*],5)

Raciocínio do agente

Tell(*KB*,*Percebe*([*Cheiro, Brisa, Brilho, Nada, Nada*],5)

- Função `Make-Action-Query`:

  ASK(*KB*, $\exists$ *a* Action(*a*, 5))

Devolvendo uma lista de possíveis instâncias de variáveis, por ex.: {*a/Atire, a/ParaFrente*}

O agente registra na KB (*Tell*) qual ação foi tomada (*o agente não percebe a sua própria ação):*

Tell(*KB*,*Ação*(*Pega,5*)

Agente reativo

- Possui sentenças que associam diretamente percepções às ações. Exemplo:

$\forall$ ***s, b, u, c, t Percebe*([*s, b, Brilho, u, c*], t) => *Ação(Pega,t)***

- Em dois passos de inferência:

$\forall$ ***b, u, g, c, t Percebe*([*Cheiro, b, g, u, c*], t) => *Cheira(t)***
$\forall$ ***s, u, g, c, t Percebe*([*s, Brisa, g, u, c*], t) => *Brisa (t)***
$\forall$ ***s, b, u, c, t Percebe*([*s, b, Brilho, u, c*], t) => *Em-ouro(t)***

$\forall$***t Em-ouro(t)* => *Ação(Pega,t)***

Agente reativo: problemas

- Um agente *reativo puro* :
  - não sabe se ele ja pegou o ouro
  - não sabe se ele está no quadrado de saída
  - não sabe quando ele deve escalar para a saída ou se o gol já foi satisfeito
  - não sabe se ele decidiu que uma posição é perigosa
  - não pode evitar loops infinitos (pode voltar para um quadrado e receber as mesmas percepções e fazer o mesmo que fez anteriormente)

KB

- A Base de Conhecimento possue
  - Histórico de todas as percepções
  - As ações que ele executou
- Faltam: regras que se referem a percepções atuais e passadas
- Problema: é difícil escrever sentenças que relacionem todas as percepções do passado com o presente

KB

- Outra possibilidade:
  - manter um modelo interno do mundo atualizado (aspectos relevantes). Ex.: chaves perdidas.
    - Inferências sobre um Video-tape
  - o agente adiciona na KB somente fatos que "importam"
  - analogia com planejamento: projeção temporal de um plano e manutenção do estado atual do mundo

KB

- Maneira de se lidar com as mudanças do mundo:
  - modificar a KB (retire a sentença “Agente em [1,1]” e insira a sentença “Agente em [1,2]”)
  - bom para responder perguntas sobre a situação atual mas a história do agente é perdida.
- Manter várias bases de conhecimento.
  - O agente pode explorar situações hipotéticas. Problema: o agente não pode raciocinar sobre mais do que uma situação simultaneamente.

Cálculo de Situações - caracterização formal

- Muitos trabalhos mas poucos completos [McCarthy & Hayes, 1969]
- CS pode ser visto como uma ontologia particular incluindo: *situações*, *fluentes* e *ações*. Além disso, CS inclue certos predicados e funções, como *Result* e *Holds*
- Uma formalização que usa essas funções e predicados envolve *axiomas de efeito* e *restrições de domínio*

Definição

Uma *linguagem de cálculo de situações* é uma linguagem de cálculo de predicados de primeira ordem multi-tipada (*many-sorted*) com igualdade que inclue:

- um tipo para *situação, com variáveis s, s1, s2, etc.,*
- um tipo para *fluentes*, com variáveis f, f1, f2, etc.,
- um tipo para *ações*, com variáveis a, a1, a2, etc.,
- uma função *Result* de ações e situações para situações,
- um predicado Holds cujos argumentos são um fluente e uma situação

Fluentes

- Função cujo domínio é o espaço de situações e cujo contra-domínio é {V, F}, ou seja, é um predicado cujo argumento é uma situação. Ex: **Chovendo(S0)**
- Reificação do fluente:

  Holds(Chovendo, S0)
- Chovendo era um predicado e foi convertido em um objeto concreto
- Através da reificação de fluentes é mais fácil escrever fórmulas genéricas sobre um conjunto de fluentes

Exemplo

"O cozinheiro chefe de um restaurante é Fred na situação S0 e Maria na situação S1"

- versão não-reificada:
  - CozinheiroChefe(S0, Fred)
  - CozinheiroChefe(S1, Maria)
- versão reificada (convenção adotada):
  - Holds(CozinheiroChefe(Fred), S0)
  - Holds(CozinheiroChefe(Maria), S1)

Fórmula de Cálculo de Situações

- Existem três classes de fórmulas comumente encontradas em CS na literatura:
  - axiomas de efeito
  - restrições do domínio
  - sentenças de observação
- Técnicas populares incluem os predicados: *Causes*, *Precond*, ou *Poss*.

Definição

Um *axioma de efeito* é uma fórmula da forma:

$\forall s\ [Holds(\beta, Result(\alpha,s)) \leftarrow \Pi]$ ou

$\forall s\ [Holds(\beta, Result(\alpha,s))]$ ou

$\forall s\ [\neg Holds(\beta, Result(\alpha,s)) \leftarrow \Pi]$ ou

$\forall s\ [\neg Holds(\beta, Result(\alpha,s))]$

onde Π é uma fórmula na qual toda a ocorrência de um termo do tipo situação está na fórmula da forma: $Holds(\beta_i,s)$ ou $\neg Holds(\beta_i,s)$

Definição

Uma *restrição do domínio* é uma fórmula da forma:

$\forall s\ \Pi$

onde Π é uma fórmula na qual toda a ocorrência de um termo do tipo situação está na fórmula da forma: $Holds(\beta_i,s)$ ou $\neg Holds(\beta_i,s)$ e que contém pelo menos uma ocorrência deste tipo.

Definição

- Chamamos de *Teoria do Domínio* ou *Descrição do Domínio* uma coleção de axiomas de efeitos e restrições do domínio

Ações ramificadas

- Um domínio contendo restrições do domínio, significa que ações podem ter ramificações que vão além de seus efeitos diretos
- Exemplo:

$$Holds(Stack(x,y,z),s) \leftarrow x \neq Table \wedge Holds(On(y,x),s) \wedge Holds(On(z,y),s)$$

Implied domain constraints

- É uma consequência de um conjunto de axiomas de efeito mas que não é um axioma. Por exemplo:

  Holds(F, Result(A,s)) $\leftarrow$ Holds(F1,s)

  $\neg$ Holds(F, Result(A,s)) $\leftarrow$ Holds(F2,s)

portanto, a seguinte restrição do domínio é uma consequência lógica dessas duas fórmulas

$\neg$ Holds(F1,s) $\wedge$ Holds(F2,s)

Sentença de observação

- Uma sentença de observação é uma fórmula na qual toda ocorrência de Holds é da forma: Holds(β, σ) ou $\neg$ Holds(β,σ), onde σ é um termo que não é variável. Exemplos:

  Holds(On(C, Table), S0)

"um dispositivo está em um único estado (x ou y) na situação S0"

$\exists$x (Holds(Status(x),S0)$\wedge\neg\exists$y [Holds(Status(y),S0)$\wedge$x$\neq$y]]

que deve ser usado com o seguinte axioma de efeito:

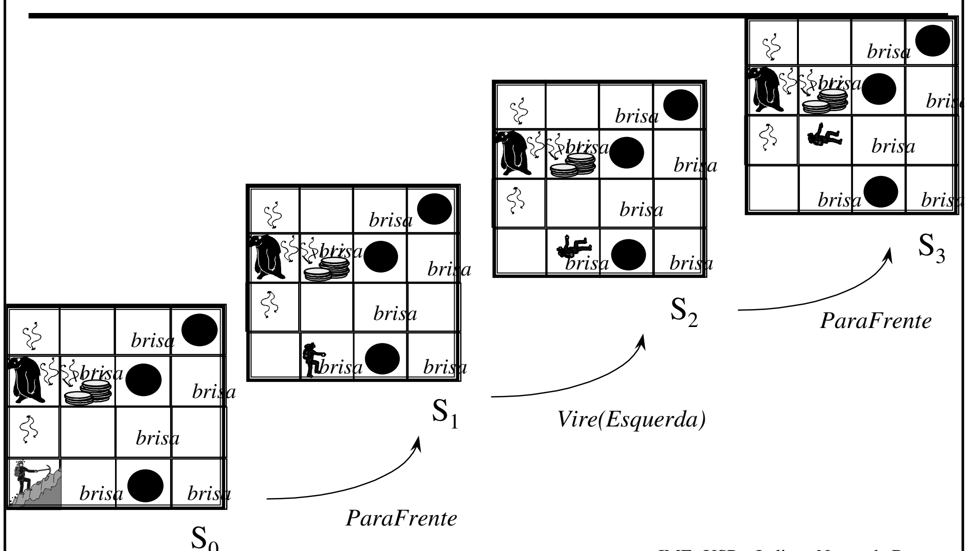
Holds(Status(x),Result(Toggle,s)) $\leftarrow$ Holds(Status(y),s) $\wedge$ Opposite(x,y)

Problemas de *predição* ou *projeção temporal*

Problemas em que uma coleção de axiomas de efeito e restrições de domínio é integrada (conjunção) a uma coleção de sentenças de observação sobre uma *situação inicial* (S0). As consequências lógicas desta conjunção (Σ) deve nos dizer quais fluentes valem depois de alguma sequência de ações. Por exemplo, poderiamos ter:

Σ |= Holds(Colour(A,Red),Result(Paint(A,Red), Result(Paint(A,Blue),S0)))

Cálculo de Situações



Cálculo de Situações

- Toda relação ou objeto que muda no mundo deve possuir um argumento extra:

Localizado-em(Agente, [1,1], S_0*) ^ Localizado-em(Agente, [1,1],* S_1*)*

que descreve a localização do agente em duas situações diferentes

- relações ou propriedades que não mudam com o tempo não precisam do argumento extra de situação.
  Ex.: *Tem-parede(0,1)*

Cálculo de Situações

- Como representar as mudanças de mundo ?
- através da função ***Resulta(ação, situação)*** que retorna a situação que resulta da execução da ação em uma dada situação:

Resultado(ParaFrente, S_0***) =*** S_1

Resultado(Vira(Esquerda), S_1***) =*** S_2

Resultado(ParaFrente, S_2***) =*** S_3

Situações

- Situações são geradas (ligadas) a partir de situações anteriores através da descrição dos efeitos de ações.
- Representando efeitos para seleção da ação que atinge gols (raciocínio sobre situações)

$$\forall\, s\ \textit{Em-ouro}(s) \Rightarrow \textit{Segura}(\textit{Ouro}, \underbrace{\textit{Resultado}(\textit{Pega},s)}_{s'})$$

$$\forall\, a,x,s\ \textit{Segura}(x,s) \wedge (a \neq \textit{Solta}) \Rightarrow \textit{Segura}(x, \textit{Resultado}(a,s))$$

Axiomas de efeito

- Ações são descritas através de seus efeitos, ou seja, propriedades das situações que resultam com a execução da ação
- Ex.: “uma vez pego o ouro ele está onde o agente está”

$$\forall\, s\ \textit{Em-ouro}(s) \Rightarrow \textit{Segura}(\textit{Ouro}, \underbrace{\textit{Resultado}(\textit{Pega},s)}_{s'})$$

Axiomas de Frame

O agente deve ser capaz de descrever (saber) como o mundo permanece o mesmo depois da execução das ações.

Exemplo:

"O agente não segura nada depois de uma ação solta":

$\forall x,s \; \neg \mathbf{\mathit{Segura(x, Resultado(Solta,s))}}$

"Se o agente segura qualquer coisa e não a solta, então ele a segura no próximo estado"

$\forall a,x,s \; \mathbf{\mathit{Segura(x,s)}} \wedge (\mathbf{\mathit{a}} \neq \mathbf{\mathit{Solta}}) \Rightarrow \mathbf{\mathit{Segura(x, Resultado(a,s))}}$

Axioma de sucessor-de-estado

- Maneira mais elegante de se representar os dois axiomas (cada axioma é sobre um predicado e não sobre uma ação):

O que é verdade no mundo $\Leftrightarrow$ [*uma ação tornou verdadeiro* $\vee$ *já era verdade e nenhuma ação tornou falso*] ou

P é verdade $\Leftrightarrow$ [*uma ação tornou P verdade* $\vee$ *P já era verdade e nenhuma ação tornou P falso*]

Exemplo:

$\forall a,x,s \; \mathbf{\mathit{Segura(x, Resultado(a,s))}} \Leftrightarrow (\mathbf{\mathit{a = Pega}} \wedge (\mathbf{\mathit{Presente(x,s)}})$
$\vee ((\mathbf{\mathit{Segura(x,s)}} \wedge (\mathbf{\mathit{a}} \neq \mathbf{\mathit{Solta}}))$

Frame Problem

- ***Representational frame problem*** (resolvido com o axioma de sucessores de estado: evita o crescente número de axiomas de frame)
- ***Qualification problem*:** descrições do mundo real requer uma profundidade ilimitada: e se o ouro for escorregadio ou colado no chão ou ...
- ***Ramification problem***: ações reais possuem muitas consequências secundárias (a partir de restrições do domínio)

Mundo de Wumpus

- Questão:
  - Como o agente pode deduzir onde estão os abismos, o Wumpus e ainda ter certeza que ele fez uma busca completa pelo ouro?

Mundo de Wumpus

- Localização é importante. Não pode ser percebida diretamente mas o agente precisa se lembrar:
  - onde ele esteve? o que ele viu?
- para:
  - poder deduzir onde estão os abismos, o Wumpus e ainda para ter certeza de ter feito uma busca completa pelo ouro.
- Localização inicial:

$$Localizado\text{-}em(\ Agente,\ [1,1], S_0\ )$$

Raciocinando no Mundo de Wumpus

- Localização da brisa ou cheiro: *Brisa(l)* e *Cheiro(l)*

  $\forall$ ***l,s Localizado-em(Agent,l,s)*** $\wedge$ ***Brisa(s)*** $\Rightarrow$ ***Brisa(l)***
- Dois tipos de regras que relacionam propriedades do mundo do mesmo estado:
  - Regras causais (propriedades do mundo causam a geração de percepções)

$\forall\, l_1, l_2, s\ \mathbf{Localizado\text{-}em(Abismo,}\ l_1, s) \wedge \mathbf{Adjacente}(l_1, l_2) \Rightarrow \mathbf{Brisa}(l_2)$

  - Regras de diagnóstico (a partir da percepção infiro as propriedades do mundo)

$\forall$ ***l,s Localizado-em(Agent,l,s)*** $\wedge$ ***Brisa(s)*** $\Rightarrow$ ***Brisa(l)***

$\forall\, l_1, s\ \mathbf{Cheiro}(l_1) \Rightarrow (\exists\, l_2\ \mathbf{Localizado\text{-}em(Wumpus,}\ l_2, s) \wedge (l_2 = l_1 \vee \mathbf{Adjacente}(l_1, l_2)\,)\,)$

Tipos de raciocínio

- Sistemas que raciocinam com regras causais são também chamados de **Sistemas Baseados em Modelo** (*Model Based Reasoning Systems*)
- Sistemas que raciocinam com regras de diagnóstico são também chamados de **Sistemas de Classificação Heurística**

Caminho de volta

- As sentenças que descrevem as ações são suficientes para garantir que o agente encontre o ouro de maneira segura
- Uma vez que o ouro foi encontrado, o agente deve retornar à saída o mais "rápido" possível

$\forall$ ***s Segura(Ouro,s)*** $\Rightarrow$ ***Localização-goal([1,1], s)***

O agente deve gerar uma sequência de ações para atingir este objetivo => o agente deve planejar o seu caminho de volta

Construindo planos

- Condição inicial em KB:

  Localizado-em(Agent,[1,1],S_0)

  Localizado-em(Ouro,[1,2],S_0)

  Pergunta: *ASK*(*KB*, ∃ *s* Segura(*Ouro*, s))

  "*Em que situação eu seguro o Ouro?*"

  Resposta:

  {s/*Resultado(Pega,Resultado(ParaFrente,S_0))*}

  "Vou para frente e pego o ouro"

Planejamento

- Representação de planos como uma sequência de ações [a_1, a_2, …,a_n] (aplicação sucessiva de funções)
- *ResultadoPlano(p,s)* é a situação resultante da execução de *p* em *s*
- Pergunta: *ASK*(*KB*,∃ *p* Segura(*Ouro, ResultadoPlano(p,S_0)*))
- Solução: {*p*/[*ParaFrente, Pega*]}
- Definição de *ResultadoPlano* em termos de *Resultado*:

$\forall\, s\ \textbf{ResultadoPlano}([],s) = s$

$\forall\, a,p,s\ \textbf{ResultadoPlano}([a|p],s) = \textbf{ResultadoPlano}(p,\textbf{Resultado}(a,s))$

Sistemas de Planejamento

- Sistemas de planejamento são processos de raciocínio de *propósito específico* (algoritmos) designados a fazer esse tipo de inferência de maneira *mais eficiente* do que os processos de raciocicínio de propósito geral como a LP ou LPO
- Questão: é possível construir um planejador baseado em lógica que seja tão *eficiente* quanto o algoritmo de planejamento POP?

Mais sobre o mundo de wumpus

O que o agente precisa saber:

- $Orientação(Agente,S_0) = Leste$
- Mapa de localizações: valores da função *LocationToward*
- Identificar o quadrado da frente
- Identificar o quadrado adjacente
- Localização das paredes
- Controle da orientação