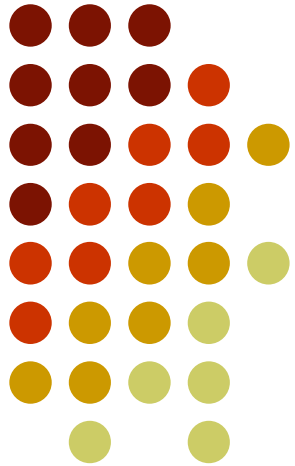
Ações e Planejamento

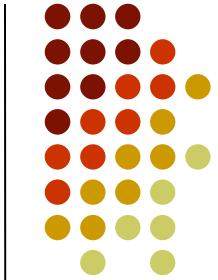
Profa. Josiane

David Poole, Alan Mackworth e Randy Goebel -
"*Computational Intelligence – A logical approach*" - cap. 8

setembro/2007

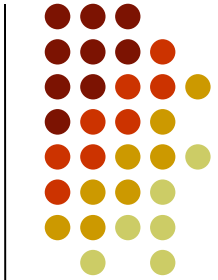


Ações e Planejamento



- Agentes que raciocinam no tempo
  - O tempo passa enquanto os agentes agem e raciocinam
- Agentes que raciocinam sobre o tempo
  - Dado um objetivo, é útil para o agente pensar sobre o que ele fará no futuro para determinar o que ele fará agora

Representação de Tempo

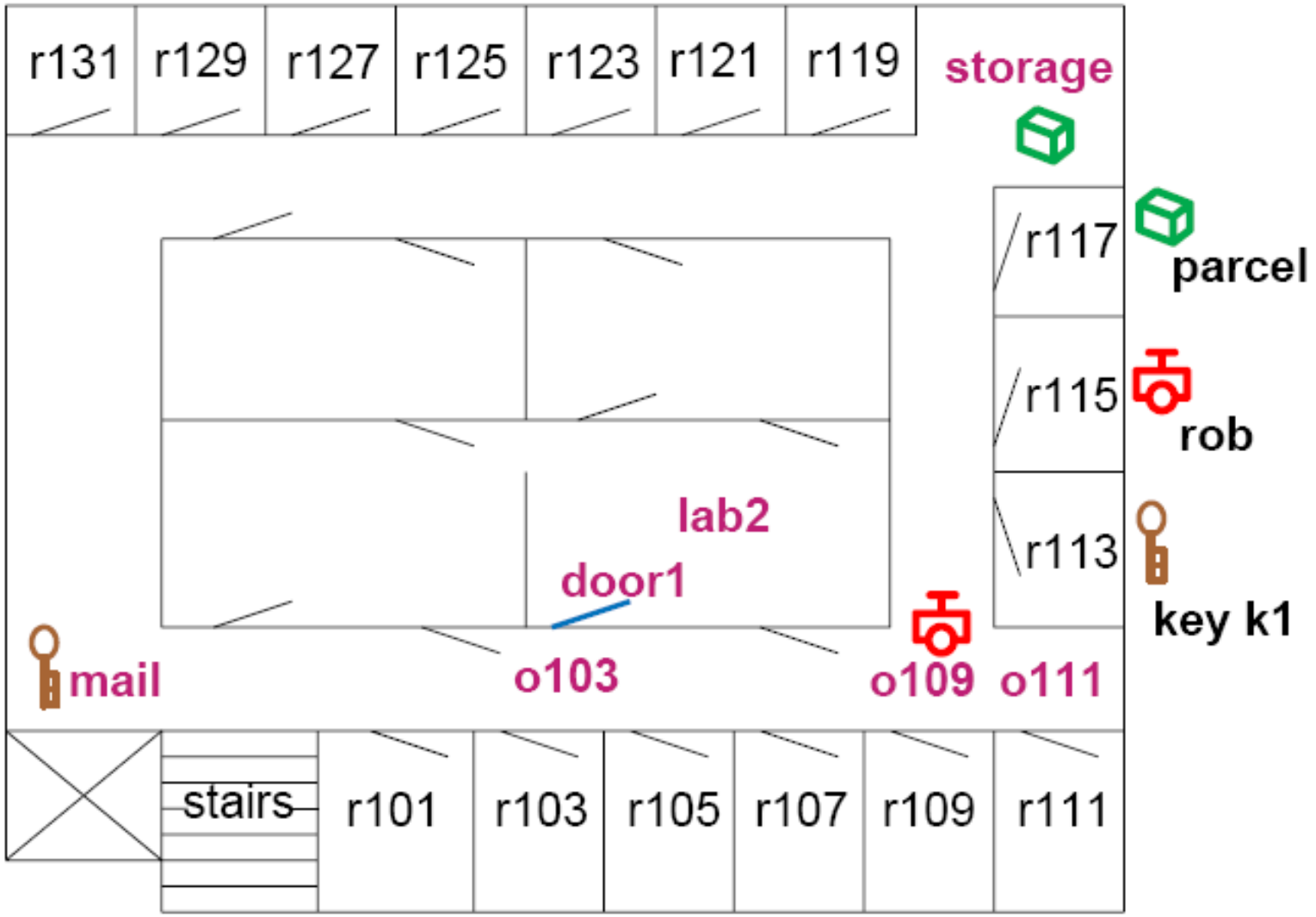


- Tempo pode ser modelado de várias formas:
  - Tempo **discreto**: Modelado como se tempo saltasse de um ponto para outro (uma ação por milissegundo ou dia, por exemplo)
  - Tempo **contínuo**: Modelado sem saltos
  - Tempo **baseado em evento**: Os passos no tempo não tem que ser uniforme; podemos considerar os passos como o tempo entre eventos interessantes (antes e depois de uma ação, por exemplo)
  - **Espaço de estados**: Ao invés de considerar tempo explicitamente, podemos considerar ações como mapeamento de um estado para outro
- Podemos modelar tempo em termos de pontos ou intervalos

Tempo e relações

- Dois tipos básicos de relações:
  - Relações **Estáticas**: aquelas para as quais os valores não dependem do tempo
  - Relações **Dinâmicas**: aquelas para as quais os valores verdade dependem do tempo. Podem ser:
    - Relações **Derivadas**: aquelas cuja definição pode ser derivada de outra relação para cada tempo
    - Relações **Primitivas**: aquelas cujo os valores verdade podem ser determinados considerando o tempo anterior
    - Relações derivadas ou primitivas são uma suposição de modelagem, e não uma propriedade do domínio

O mundo do robô de entrega



Modelando o mundo do robô de entrega

- Indivíduos: salas, portas, chaves, pacotes, e o robô.
- Ações:
  - Mover de uma sala para outra
  - Pegar e soltas chaves e pacotes
  - Destrancar portas (com as chaves corretas)
- Relações
  - A posição do robô
  - A posição dos pacotes e chaves e das portas trancadas
  - O que o robô está segurando

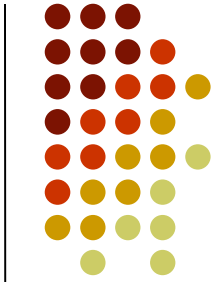
Modelando as relações

- ***at*(*Obj*, *Loc*)** é verdade em uma situação se o objeto *Obj* está na localização *Loc* na situação
- ***carrying*(*Ag*, *Obj*)** é verdade em uma situação se o objeto *Ag* está carregando *Obj* naquela situação
- ***sitting_at*(*Obj*, *Loc*)** é verdade em uma situação se o objeto *Obj* está parado (não sendo carregado) na localização *Loc* naquela situação
- ***unlocked*(*Door*)** é verdade em uma situação se a porta *Door* está destrancada na situação
- ***autonomous*(*Ag*)** é verdade se o agente *Ag* pode sem mover autonomamente.
  - É uma relação estática

Modelando as relações

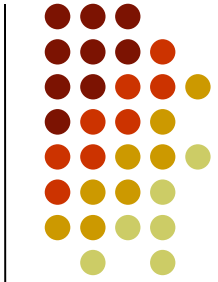
- ***open*(*key*, *Door*)** é verdade se a chave *key* abre a porta *Door*.
  - É uma relações estática
- ***adjacent*(Pos_1, Pos_2)** é verdade se a posição Pos_1 é adjacente à posição Pos_2 de forma que o robô pode se mover de Pos_1 para Pos_2 em um passo
- ***between*(*Door*, Pos_1, Pos_2)** é verdade se *Door* está entre as posições Pos_1 e Pos_2
  - Se a porta está aberta as duas posições são adjacentes
  - É uma relação estática

Modelando as ações



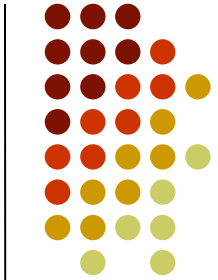
- ***move*(*Ag*, *From*, *To*)** é a ação do agente *Ag* se mover da localização *From* para a localização *To*
  - O agente pode fazer esta ação se ele estiver na localização *From* e a localização *To* é uma localização adjacente a *From*
- ***pickup*(*Ag*, *Obj*)** é a ação do agente *Ag* pegar o objeto *Obj*
  - O agente pode fazer esta ação se ele estiver na mesma localização que o objeto *Obj*
- ***putdown*(*Ag*, *Obj*)** é a ação do agente *Ag* soltar o objeto *Obj*
  - O agente pode fazer esta ação somente se ele estiver segurando o objeto *Obj*
- ***unlock*(*Ag*, *Door*)** é a ação do agente *Ag* a porta *Door*
  - O agente pode fazer esta ação somente se ele estiver do lado de fora da porta e estiver carregando a chave para aquela porta

Descrição do mundo inicial



- **Situação inicial:**
  - *sitting_at*(*rob*, *o109*).
  - *sitting_at*(*parcel*, *storage*).
  - *sitting_at*(*k1*, *mail*).
- **Fatos estáticos:**
  - *between*($door_1$, *o103*, lab_2).
  - *opens*(k_1, $door_1$).
  - *autonomous*(*rob*).

Relações derivadas



- *adjacent*(o*109*, o*103*).
- *adjacent*(o*103*, o*109*).
- *adjacent*(o*109*, *storage*).
- *adjacent*(*storage*, o*109*).
- *adjacent*(o*109*, o*111*).
- *adjacent*(o*111*, o*109*).
- *adjacent*(o*103*, *mail*).
- *adjacent*(*mail*, o*103*).
- *adjacent*(lab_2, o*109*).
- *adjacent*(P_1, P_2) ← *between*(*Door*, P_1, P_2) ^ *unlocked*(*Door*).
- *at*(*Obj*, *Pos*) ← *sitting_at*(*Obj*, *Pos*).
- *at(Obj, Pos)* ← *carying(Ag, Obj)* ^ *at(Ag, Pos).*

Representação das ações e mudanças

- Visões do tempo baseadas em estado. Mapeiam um estado para outro:
  - STRIPS (*STanford Research Institute Problem Solver*)
  - Cálculo situacional
- Visão que considera tempo explicitamente:
  - Cálculo de eventos

Representação em STRIPS

- Ações externas à lógica
- Dado um estado e uma ação, a representação em STRIPS é usada para determinar:
  - Se a ação pode ser feita no estado
  - O que é verdade no estado resultante
- Predicados são **primitivos** ou **derivados**
- A representação em STRIPS é usada para determinar os valores verdade dos predicados primitivos baseados no estados anterior e na ação
- Baseada na idéia que a maioria dos predicados não são afetados por uma simples ação
- **Suposição da STRIPS**: relações primitivas não mencionadas na descrição da ação não são mudadas

Representação de uma ação em STRIPS

- A representação de uma ação consiste de:
  - **Precondições**: uma lista dos átomos que necessitam ser verdade no estado para que a ação ocorra
  - **Lista a apagar**: uma lista daquelas relações primitivas que não serão mais verdadeiras no estado após a ação
  - **Lista a adicionar**: uma lista daquelas relações primitivas que se tornam verdadeiras no estado pela ação

Representação de uma ação em STRIPS – Exemplos

- Exemplo 1: A ação pegar *pickup*(*Ag*, *Obj*) pode ser definida por:
  - **Precondições**: [*automomous*(*Ag*), *Ag* ≠ *Obj*, *at*(*Ag*, *Pos*), *sitting_at*(*Obj*, *Pos*)]
  - **Lista a apagar**: [*sitting_at*(*Obj*, *Pos*)]
  - **Lista a adicionar**: [*carrying*(*Ag*, *Obj*)]
- Exemplo 2: A ação mover *move*(Pos_1, Pos_2, *Ag*) pode ser definida por:
  - **Precondições**: [*automomous*(*Ag*), *adjacent*(Pos_1, Pos_2, *S*), *at*(*Ag*, *Pos*), *sitting_at*(*Ag*, Pos_1)]
  - **Lista a apagar**: [*sitting_at*(*Ag*, Pos_1)]
  - **Lista a adicionar**: [*sitting_at*(*Ag*, Pos_2)]

Representação gráfica de uma ação em STRIPS – Exemplos

- Exemplo 1: A ação pegar *pickup*(*Ag*, *Obj*) pode ser definida por:
  - **Precondições**: [*automomous*(*Ag*), *Ag* ≠ *Obj*, *at*(*Ag*, *Pos*), *sitting_at*(*Obj*, *Pos*)]
  - **Lista a apagar**: [*sitting_at*(*Obj*, *Pos*)]
  - **Lista a adicionar**: [*carrying*(*Ag*, *Obj*)]

automomous(*Ag*) ^ *Ag* ≠ *Obj* ^ *at*(*Ag*, *Pos*), *sitting_at*(*Obj*, *Pos*)

pickup(*Ag*, *Obj*)

~*sitting_at*(*Obj*, *Pos*) ^ *carrying*(*Ag*, *Obj*)

Representação gráfica de uma ação em STRIPS – Exemplos

- Exemplo 2: A ação mover *move*(Pos_1, Pos_2, *Ag*) pode ser definida por:
  - **Precondições**: [*automomous*(*Ag*), *adjacent*(Pos_1, Pos_2, *S*), *at*(*Ag*, *Pos*), *sitting_at*(*Ag*, Pos_1)]
  - **Lista a apagar**: [*sitting_at*(*Ag*, Pos_1)]
  - **Lista a adicionar**: [*sitting_at*(*Ag*, Pos_2)]

automomous(*Ag*) ^ *adjacent*(Pos_1, Pos_2, *S*) ^ *at*(*Ag*, *Pos*) ^ *sitting_at*(*Ag*, Pos_1)

move(Pos_1, Pos_2, *Ag*)

~*sitting_at*(*Ag*, Pos_1) ^ *sitting_at*(*Ag*, Pos_2)

Exemplos de Transições (aplicação da ações)

- Estado Inicial:

$$\left\{ \begin{array}{l} sitting\_at(rob, o109). \\ sitting\_at(parcel, storage). \\ sitting\_at(k1, mail). \end{array} \right\}$$

- Após a aplicação da ação ***move*(*rob*, *o109*, *storage*)**:

$$\left\{ \begin{array}{l} sitting\_at(rob, storage). \\ sitting\_at(parcel, storage). \\ sitting\_at(k1, mail). \end{array} \right\}$$

Exemplos de Transições
(aplicação da ações)

$$\left\{\begin{array}{l} sitting\_at(rob, storage). \\ sitting\_at(parcel, storage). \\ sitting\_at(k1, mail). \end{array}\right\}$$

- Após a aplicação da ação ***pickup*(*rob*, *parcel*)**:

$$\left\{\begin{array}{l} sitting\_at(rob, storage). \\ carrying(rob, parcel). \\ sitting\_at(k1, mail). \end{array}\right\}$$

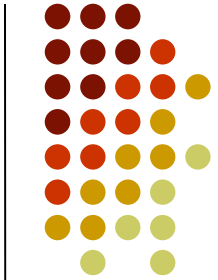
Cálculo Situacional

- Representação baseada em estados onde os estados são denotados por termos
- Uma **situação** é um termo que denota um estado
- Existem duas formas de referências a estados:
  - ***Init*** denota o estado inicial
  - ***do*(*A*, *S*)** denota o estado resultante de fazer a ação *A* no estado *S*, se for possível executar *A* em *S*
- Uma situação também codifica como ter acesso ao estado que ela denota

Exemplo de Estados

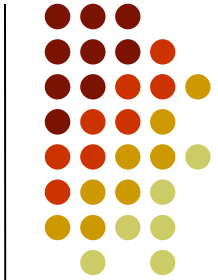
- *init*
- *do*(*move*(*rob*, o*109*, o*103*), *init*).
  do(A_1, S_1).
- *do*(*move*(*rob*, o*103*, *mail*), --> A_2
  do(*move*(*rob*, o*109*, o*103*), --> A_1
  init)). --> S_1 } S_2
- *do*(*pickup*(*rob*, *k1*), --> A_3
  do(*move*(*rob*, o*103*, *mail*), --> A_2
  do(*move*(*rob*, o*109*, o*103*), --> A_1
  init))). --> S_1 } S_2 } S_3

Usando o termo situação



- Adicionar um termo extra para cada predicado dinâmico indicando a situação
- Exemplo:
  - *at*(*rob*, *o109*, *init*) é verdadeiro se o robô *rob* estiver na posição *o109* na situação inicial
  - *at*(*rob*, o*109*, *do*(*move*(*rob*, *o109*, *o103*), *init*)) é verdadeiro se o robô *rob* estiver na *o103* na situação resultante de *rob* mover-se da da posição *o109* para a posição o103 da situação inicial
  - *at*(*k1*, *mail*, *do*(*move*(*rob*, *o109*, *o103*), *init*) é verdadeiro se *k1* estiver na posição *mail* na situação resultante de *rob* mover-se da da posição *o109* para a posição o103 da situação inicial

Axiomatização usando Cálculo Situacional



- Especificamos o que é verdade no estado inicial usando axiomas com ***init*** como o parâmetro da situação
- **Relações Primitivas** são definidas especificando quais instâncias são verdadeiras nas situações da forma *do*(*A*, *S*) em termos do que é válido na situação *S*
- **Relações Derivadas** são definidas usando cláusulas com uma variável livre no argumento da situação
  - Sua verdade em uma situação depende do que mais é verdade na situação
- **Relações Estáticas** são definidas sem referência à situação

Axiomatização usando Cálculo Situacional - Exemplo

- Situação inicial

$$sitting\_at(rob, o109, init).$$

$$sitting\_at(parcel, storage, init).$$

$$sitting\_at(k1, mail, init).$$

- Relações Derivadas

$$adjacent(P_1, P_2, S) \leftarrow$$

$$between(Door, P_1, P_2) \wedge$$

$$unlocked(Door, S).$$

$$adjacent(lab2, o109, S).$$

$$\ldots$$

Ações possíveis
(semelhante às precondições em STRIPS)

- ***poss*(*A*, *S*)** é verdade se a ação *A* é possível no estado *S*

$$poss(putdown(Ag, Obj), S) \leftarrow$$
$$carrying(Ag, Obj, S).$$

$$poss(move(Ag, Pos_1, Pos_2), S) \leftarrow$$
$$autonomous(Ag) \wedge$$
$$adjacent(Pos_1, Pos_2, S) \wedge$$
$$sitting\_at(Ag, Pos_1, S).$$

Axiomatização das relações primitivas

- **Exemplo**: Destrancar uma porta faz a porta ficar destrancada

$$unlocked(Door, do(unlock(Ag, Door), S)) \leftarrow$$
$$poss(unlock(Ag, Door), S).$$

- **Axioma de frame**: nenhuma ação tranca uma porta

$$unlocked(Door, do(A, S)) \leftarrow$$
$$unlocked(Door, S) \wedge$$
$$poss(A, S).$$

- **Axiomas de frame** especificam o que continua sem mudanças durante uma ação

Axiomatização das relações primitivas

- **Exemplo**: Pegar um objeto causa o efeito de ele ser carregado

$$carrying(Ag, Obj, do(pickup(Ag, Obj), S)) \leftarrow$$
$$poss(pickup(Ag, Obj), S).$$

- **Axioma de frame**: o objeto continua sendo carregado se ele estava sendo carregado antes, ao menos que a ação for soltar

$$carrying(Ag, Obj, do(A, S)) \leftarrow$$
$$carrying(Ag, Obj, S) \wedge$$
$$poss(A, S) \wedge$$
$$A \neq putdown(Ag, Obj).$$

Axioma de frame mais geral

- A única ação que desfaz *sitting_at* para o objeto *Obj* é quando *Obj* se move para algum lugar ou alguém pega *Obj*

$$sitting\_at(Obj, Pos, do(A, S)) \leftarrow$$
$$poss(A, S) \wedge$$
$$sitting\_at(Obj, Pos, S) \wedge$$
$$\forall Pos_1 \ A \neq move(Obj, Pos, Pos_1) \wedge$$
$$\forall Ag \ A \neq pickup(Ag, Obj).$$

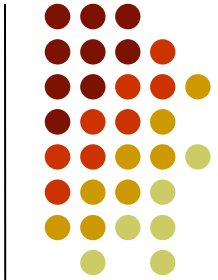
- A última linha é equivalente a

$$\sim \exists Ag \, A = pickup(Ag, Obj)$$

STRIPS e Cálculo Situacional

- Qualquer coisa que pode ser declarada em STRIPS pode ser declarada em cálculo situacional
- O cálculo situacional é mais poderoso
- Exemplo: supõe a ação *drop_everything*(*Ag*), onde o agente solta tudo o que estiver segurando
  - Não pode ser representada em STRIPS com a relação *sitting_at* pois a lista a apagar dependeria do que o agente está carregando
  - Ação *drop_everything*(*Ag*) não pode ser descrita desta forma:
    - **Precondições**: [*automomous*(*Ag*), *sitting_at*(*Ag*, *Pos*), *carrying*(*Ag*, Obj_1), *carrying*(*Ag*, Obj_2),..., *carrying*(*Ag*, Obj_n)]
    - **Lista a apagar**: [*carrying*(*Ag*, Obj_1), *carrying*(*Ag*, Obj_2),..., *carrying*(*Ag*, Obj_n)]
    - **Lista a adicionar**: [*sitting_at*(Obj_1, *Pos*), *sitting_at*(Obj_2, *Pos*),..., *sitting_at*(Obj_n, *Pos*)]

STRIPS e Cálculo Situacional



- O cálculo situacional é mais poderoso
  - *sitting_at*(*Obj*, *Pos*, *do*((*drop_everything*(*Ag*), *S*)) ←
    poss(*drop_everything*(*Ag*), *S*) ^
    at(*Ag*, *Pos*, *S*) ^
    carrying(*Ag*, *Obj*, *S*).

  - Devemos adicionar a exceção do axioma de frame para *carrying*(*Ag*, *Obj, S*)
  - *carrying*(*Ag*, *Obj*, *do*(*A*, *S*)) ←
    poss(*A*, *S*) ^
    carrying(*Ag*, *Obj*, *S*) ^
    A ≠ *drop_everything*(*Ag*) ^
    A ≠ *putdown*(*Ag*, *Obj*).

Planejamento

- Dado:
  - Uma descrição inicial do mundo
  - Uma descrição das ações disponíveis
  - Um objetivo
- Um **plano** é uma sequência de ações que irá alcançar o objetivo
- Um **planejador** (ou algoritmo de planejamento) é um resolvedor de problemas que pode produzir planos com essas entradas

Exemplo de Plano

- Vamos supor que queremos alcançar Rob segurando a chave *k1* e estando em *o103*, podemos perguntar pela query:
  - ?*carrying*(*rob*, *k1*) ^ *at*(*rob*, *o103*, *S*).
- Tem como resposta o plano:

$$S = do(move(rob, mail, o103),$$
$$do(pickup(rob, k1),$$
$$do(move(rob, o103, mail),$$
$$do(move(rob, o109, o103), init)))).$$

Planejamento para frente

- Busca no grafo do espaço de estados, onde os nós representam os estados e os arcos representam as ações
- Busca do estado inicial para o estado que satisfaz o objetivo
- Uma estratégia de busca completa (A* ou profundidade iterativa), garante encontrar a solução
- Fator de ramificação é o número de ações possíveis para qualquer estado (o que pode ser muito grande)

Planejador STRIPS

- Considera apenas ações relevantes
- Dividir e conquistar: para criar um plano que alcance uma conjunção de objetivos
  - crie um plano que alcance um objetivo, e então crie um plano para alcançar o resto dos objetivos
- Para alcançar uma lista de objetivos:
  - Escolha um deles para alcançar
  - Se ele já não tiver sido alcançado
    - Escolha uma ação que faz o objetivo verdadeiro
    - Alcance as precondições da ação
    - Inclua a ação no plano
  - Alcance o resto dos objetivos

Planejador STRIPS

- *achieve_all*(*Gs*, *W1*, *W2*) é verdadeiro se *W2* é o mundo resultante após alcançar cada objetivo da lista *Gs* de objetivos do mundo *W1*

$$achieve\_all([\,], W_0, W_0).$$

$$achieve\_all(Goals, W_0, W_2) \leftarrow$$

$$remove(G, Goals, Rem\_Gs) \wedge$$

$$achieve(G, W_0, W_1) \wedge$$

$$achieve\_all(Rem\_Gs, W_1, W_2).$$

Planejador STRIPS

- *achieve*(*G*, *W0*, *W1*) é verdadeiro se *W1* é o mundo resultante depois de alcançar o objetivo *G* do mundo *W0*

- *Objetivos válidos*

- *Relações derivadas*

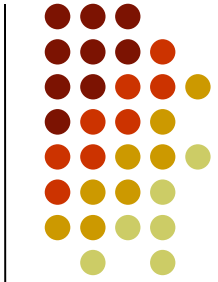
- *Relações primitivas*

$$achieve(G, W, W) \leftarrow$$
$$holds(G, W).$$
$$achieve(G, W_0, W_1) \leftarrow$$
$$clause(G, B) \wedge$$
$$achieve\_all(B, W_0, W_1).$$
$$achieve(G, W_0, do(Action, W_1)) \leftarrow$$
$$achieves(Action, G) \wedge$$
$$preconditions(Action, Pre) \wedge$$
$$achieve\_all(Pre, W_0, W_1).$$

Desfazendo os objetivos alcançados

- Exemplo: considere tentar alcançar
  - [*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *lab2*)] OK
- Exemplo: considere tentar alcançar
  - [*sitting_at*(*rob*, *lab2*), *carrying*(*rob*, *parcel*)] desfaz
- O planejador STRIPS, como apresentado, é incorreto
- Alcançar um subobjetivo pode desfazer subobjetivos já alcançados

Corrigindo planejador STRIPS



- Duas idéias para tornar STRIPS correto:
  - Proteger os subobjetivos de forma que, uma vez alcançado, até que ele seja necessário, ele não pode ser desfeito
    - Proteger subobjetivos torna STRIPS incompleto
    - Exemplo: supõe que o robô possa carregar somente um item de cada vez
    - Considere o objetivo:
      - [*sitting_at*(*rob*, *lab2*), *carrying*(*rob*, *parcel*)]
      - Não importa a ordem o primeiro objetivo sempre será desfeito
    - Não podemos considerar os subobjetivos isolados!
  - Realcançar os subobjetivos que foram desfeitos
    - Realcançar subobjetivos encontra planos maiores que o necessário

Regressão

- Idéia: não resolver um objetivo sozinho, mas guardar todos os subobjetivos que devem ser alcançados
- Dado um conjunto de objetivos:
  - Se eles todos são válidos no estado inicial, retorne o plano vazio
  - Caso contrário, escolha uma ação A que alcance um dos subobjetivos
    - Esta será a última ação do plano
  - Determine o que deve estar imediatamente antes de A de forma que todos os subobjetivos serão verdadeiros imediatamente depois
  - Recursivamente resolva os novos objetivos

Regressão como procura de caminho

- Os nós são conjuntos de objetivos
- Arcos correspondem a ações
- Um nó com um conjunto de objetivos G tem um vizinho para cada ação A que alcança um dos objetivos de G
- O vizinho correspondente para a ação A é o nó com os objetivos G_A que deve ser verdadeiro imediatamente antes da ação A e desta forma todos os objetivos em G são verdadeiros imediatamente após A
  - G_A é a precondição mais fraca para a ação A e o conjunto de objetivos G
- A busca pode parar quando temos um nó onde todos os objetivos são verdadeiros no estado inicial

Precondições mais fracas

- wp(A, GL, WP) é verdadeiro se WP é a precondição mais fraca que deve ocorrer imediatamente antes da ação A, assim todo elemento da lista de objetivos GL é verdadeiro imediatamente após A

- Para a representação STRIPS (como todos os predicados primitivos):

  - wp(A, GL, WP) é falso se qualquer elemento de GL está na lista a apagar da ação A

  - Caso contrário WP é
    - preconds(A) U {G $\in$ GL: G $\notin$ lista_adicionar(A)}

Exemplo de precondições mais fracas

- A precondição mais fraca para
  - [*sitting_at*(*rob*, *lab2*), *carrying*(*rob*, *parcel*)]
- Para ser verdadeira após a ação move(rob, Pos, lab2) é que

$$[autonomous(rob),$$
$$adjacent(Pos, lab2),$$
$$sitting\_at(rob, Pos),$$
$$carrying(rob, parcel)]$$

- É verdadeiro imediatamente antes da ação

Um planejador por Regressão

- solve(G, W) é verdadeiro se todo elemento da lista de objetivos GL for verdadeiro no mundo W

$$
\begin{aligned}
&solve(GoalSet, init) \leftarrow \\
&\quad holdsall(GoalSet, init). \\
&solve(GoalSet, do(Action, W)) \leftarrow \\
&\quad consistent(GoalSet) \wedge \\
&\quad choose\_goal(Goal, GoalSet) \wedge \\
&\quad choose\_action(Action, Goal) \wedge \\
&\quad wp(Action, GoalSet, NewGoalSet) \wedge \\
&\quad solve(NewGoalSet, W).
\end{aligned}
$$

Exercício 1 – Calçar os sapatos

- Objetivo: calçado(sapato, direito) ^ calçado(sapato, esquerdo)
- Situação Inicial: sem(meia, esquerdo) ^ sem(meia, direito)
- Ações possíveis:
  - *calçar*(S*, Lado*)
    - **Precondições**: [*colocado*(*meia, Lado*)]
    - **Lista a apagar**: [ ]
    - **Lista a adicionar**: [*calçado*(S*, Lado*)]
  - *calocar*(*M*, *Lado*)
    - **Precondições**: [*sem*(*M*, *Lado*)]
    - **Lista a apagar**: [ ]
    - **Lista a adicionar**: [*calocado*(*M*, *Lado*)]

Exercício 2 – Trocar o pneu furado

- Objetivo: em(furado, portamalas) ^ em(estepe, eixo)
- Situação Inicial: em(furado, eixo) ^ em(estepe, portamalas)
- Ações possíveis:
  - *remover*(*P, portamalas*)
    - **Precondições**: [*em*(*P, portamalas*)]
    - **Lista a apagar**: [*em*(*P, portamalas*)]
    - **Lista a adicionar**: [*em*(*P, fora*)]
  - *desmontar*(*P, eixo*)
    - **Precondições**: [*em*(*P, eixo*)]
    - **Lista a apagar**: [*em*(*P, eixo*)]
    - **Lista a adicionar**: [vazio(eixo), em(P, fora)]
  - *montar*(*P, eixo*)
    - **Precondições**: [*em*(*P, fora*), vazio(eixo)]
    - **Lista a apagar**: [*em*(*P, fora*), vazio(eixo)]
    - **Lista a adicionar**: [*em*(*P, eixo*)]
  - *colocar*(*P, portamalas*)
    - **Precondições**: [*em*(*P, fora*)]
    - **Lista a apagar**: [*em*(*P, fora*)]
    - **Lista a adicionar**: [*em*(*P, portamalas*)]