Ações e Planejamento

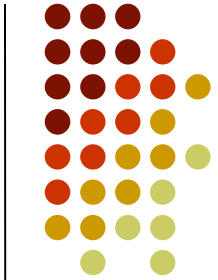
Profa. Josiane

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

Stuart Russel e Peter Norving - "Inteligência Artificial" - cap. 11

agosto/2008

Ações e Planejamento



- Agentes que raciocinam no tempo
  - O tempo passa enquanto os agentes agem e raciocinam
- Agentes que raciocinam sobre o tempo
  - O que o agente deve fazer depende dos seus objetivos, do que ele sabe sobre o mundo, e do que ele pretende fazer no futuro

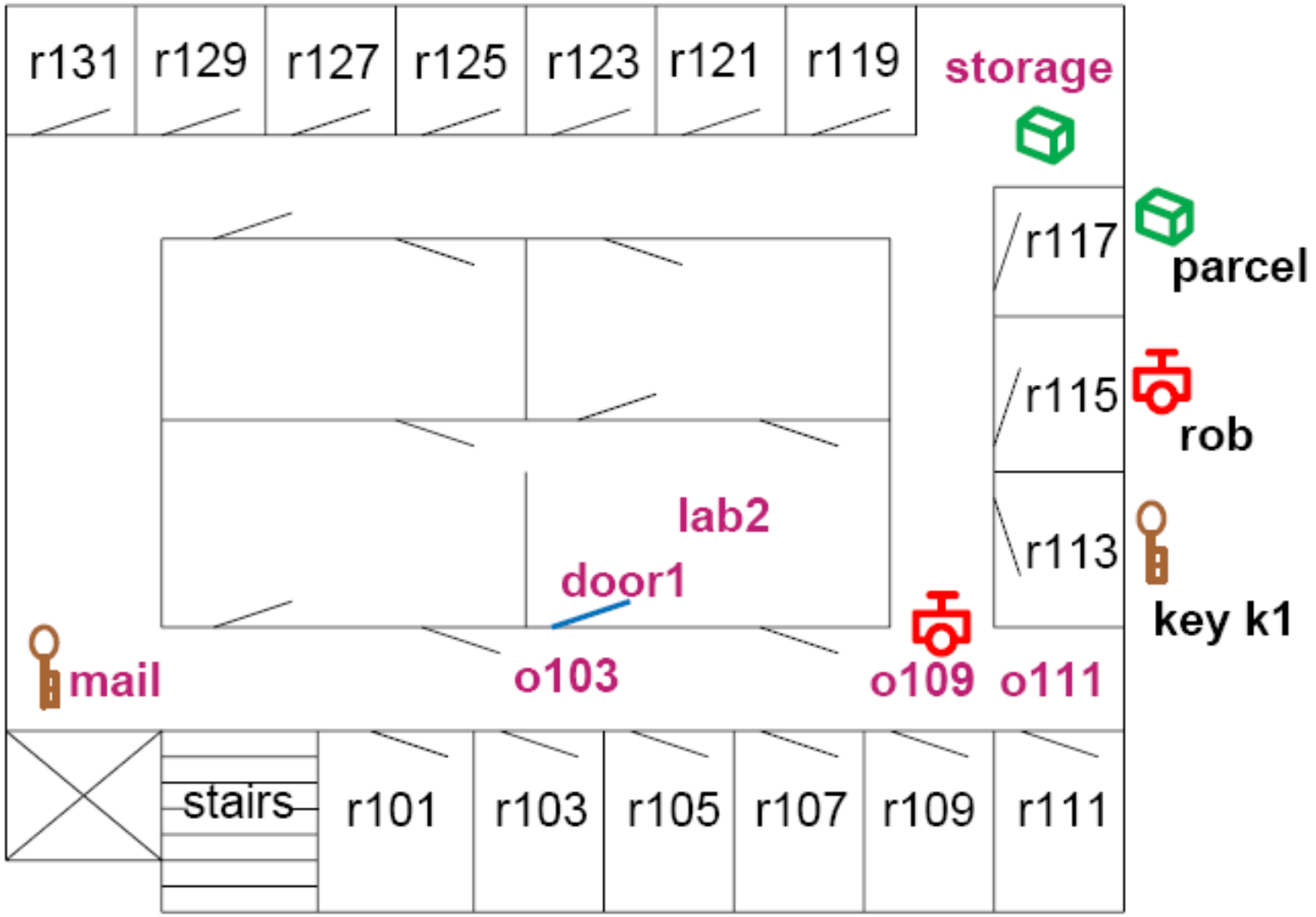
Representação de Tempo

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

Modelando as relações

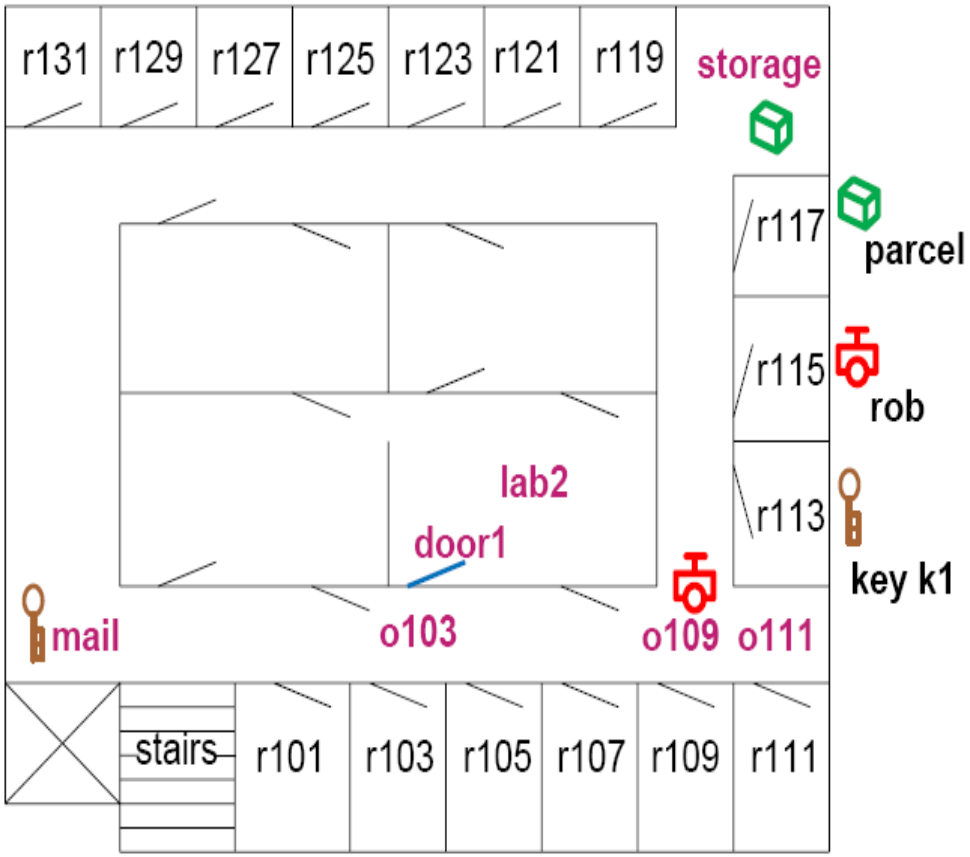
- ***open*(*key*, *Door*)** é verdade se a chave *key* abre a porta *Door*.
  - É modelada como uma relação estática
- ***adjacent*(Pos_1, Pos_2)** é verdade se a posição Pos_1 é adjacente à posição Pos_2 de forma que o robô pode se mover de Pos_1 para Pos_2 em um passo (derivada)
- ***between*(*Door*, 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 modelada como uma relação estática (não criamos portas)

Modelando as ações

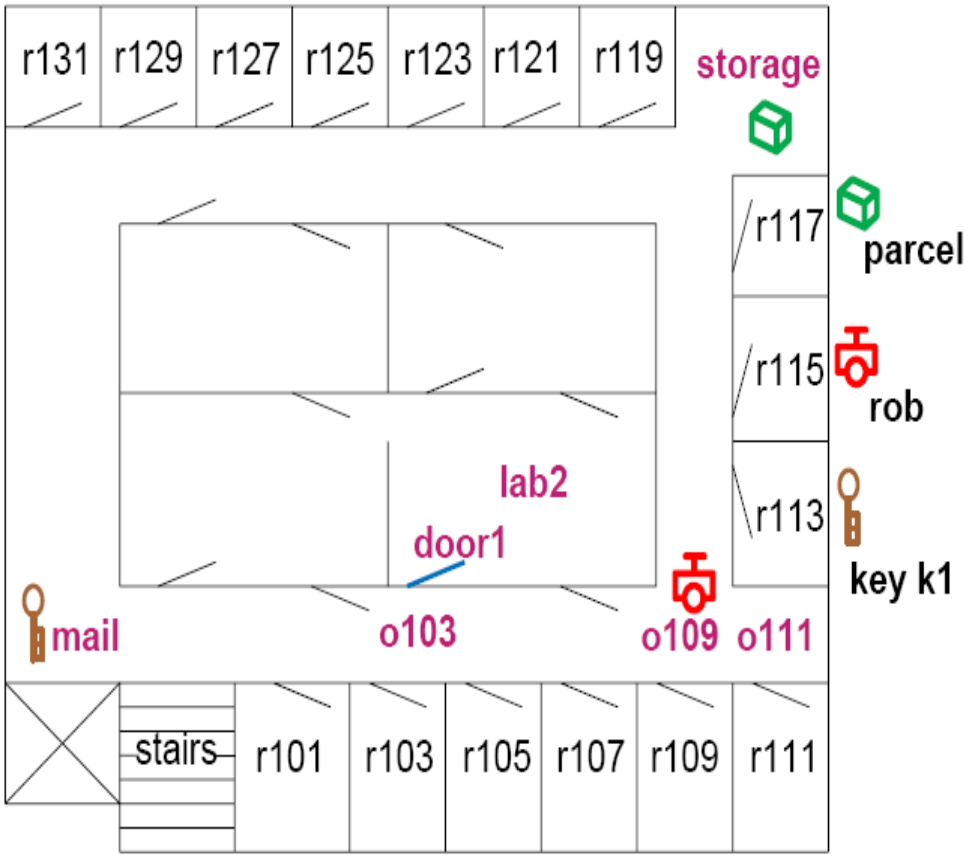
- ***move*(*Ag*, *From*, To)** é a ação do agente *Ag* se mover da localização *From* para a localização *To*
  - O agente pode fazer esta ação se ele estiver na localização *From* e a localização *To* é uma localização adjacente a *From*
- ***pickup*(*Ag*, *Obj*)** é a ação do agente *Ag* pegar o objeto *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* destrancar 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*$_2$).
  - *open*(*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*)
    - As ações e estados são externos à lógica
  - Cálculo situacional
    - Os estados são objetos da lógica (estados são reificados)
- Visão que considera tempo explicitamente:
  - Cálculo de eventos

Representação em STRIPS

- Ações e estados externos à lógica
- Dado um estado e uma ação, é usada para determinar:
  - Se a ação pode ser feita no estado
  - O que é verdade no estado resultante
- É usada para determinar os valores verdade dos predicados primitivos baseada no estado anterior e na ação
  - As relações derivadas podem ser inferidas dos valores verdade das relações primitivas
- Baseada na idéia que a maioria dos predicados não são afetados por uma simples ação
- **Suposição de STRIPS**: relações primitivas não mencionadas na descrição da ação continuam sem mudanças

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 aplicação da 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), *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*(*Ag*, Pos_1, Pos_2) pode ser definida por:
  - **Precondições**: [*autonomous*(*Ag*), *adjacent*(Pos_1, Pos_2), *sitting_at*(*Ag*, Pos_1)]
  - **Lista a apagar**: [*sitting_at*(*Ag*, Pos_1)]
  - **Lista a adicionar**: [*sitting_at*(*Ag*, Pos_2)]

autonomous(*Ag*) ^ *adjacent*(Pos_1, Pos_2) ^ *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)

- Estado Anterior:

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

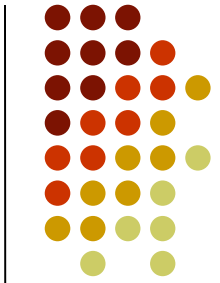
- Após a aplicação da ação ***pickup*(*rob*, *parcel*)**:

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

Cálculo Situacional

- Representação orientada a estados-ações na qual os estados e as ações são reificadas (são termos da lógica)
- 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*

Exemplo de Estados

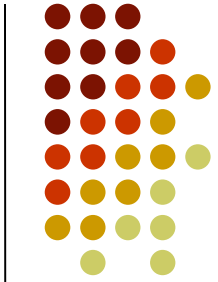


- *init*
- *do*(*move*(*rob*, o*109*, o*103*), *init*).
  do(A_1, S_1).
- *do*(*move*(*rob*, o*103*, *mail*), --> A_2
  do(*move*(*rob*, o*109*, o*103*), --> A_1
  init)). --> S_1 } S_2
- *do*(*pickup*(*rob*, *k1*), --> A_3
  do(*move*(*rob*, o*103*, *mail*), --> A_2
  do(*move*(*rob*, o*109*, o*103*), --> 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*(*Obj*, *Loc*, ***S***) *é verdade se o objeto Obj está na localização Loc na situação S*
  - *at*(*rob*, *o109*, ***init***) é verdadeiro se o robô *rob* estiver na posição *o109* na situação inicial
  - *at*(*rob*, o*103*, ***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 definida por especificar 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

- Temos que axiomatizar como o que é verdade em um estado depende do estado anterior e da ação ocorrida entre eles
- Para cada relação primitiva, especificamos:
  - Como ela se torna verdadeira e
  - Quando ela continua sendo verdadeira (axioma de frame)
- **Axiomas de frame** especificam o que continua sem mudanças durante uma ação

Axiomatização das relações primitivas
Exemplo

- **Exemplo**: Destrancar uma porta causa o efeito da porta ficar destrancada

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

- **Axioma de frame**: a porta continua destrancada porque nenhuma ação tranca uma porta

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

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*
- Axioma de frame para o relação *sitting_at*:

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

- A última linha é equivalente a

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

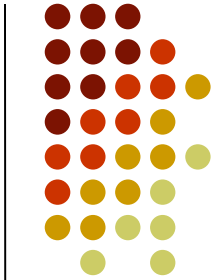
STRIPS e Cálculo Situacional

- 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*
  - Porque as instâncias de *sitting_at* a apagar dependeriam do que o agente está carregando
  - E não poderiam ser especificadas em um passo

STRIPS e Cálculo Situacional

- Exemplo: ação *drop_everything*(*Ag*)
  - **Precondições**: [*autonomous*(*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*)]
- Ação *drop_everything*(*Ag*) não pode ser descrita desta forma porque não temos conhecimento de todas as relações *carrying*(*Ag*, Obj_x)

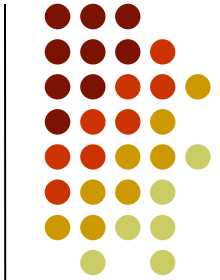
STRIPS e Cálculo Situacional



- 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

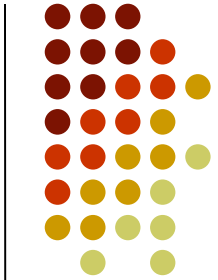
- Vamos supor que queremos alcançar Rob segurando a chave *k1* e estando em *o103*, podemos perguntar pela query:
  - ?*carrying*(*rob*, *k1*) ^ *at*(*rob*, *o103*, *S*).
- Tem como resposta o plano:

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

Planejamento como uma busca no espaço de estados

- Busca no grafo do espaço de estados, onde os nós representam os estados e os arcos representam as ações
  - Para frente: a partir do estado inicial (por progressão)
  - Para trás: a partir do objetivo (por regressão)
- Estado inicial da busca
  - Progressão: Estado inicial do problema de planejamento
  - Regressão: Estado objetivo do problema de planejamento
- Função sucessora
  - Progressão: Todas as ações aplicáveis a um estado (aquelas cujas precondições são satisfeitas)
  - Regressão: Todas as ações que alcançam alguma parte do objetivo (aquelas cujo(s) efeito(s) alcança(m) algum subobjetivo(s))

Planejamento como uma busca no espaço de estados



- Teste de objetivo
  - Progressão: Verifica se o estado satifaz o objetivo do problema de planejamento
  - Regressão: Verifica se o estado satisfaz o estado inicial do problema de planejamento
- Custo do passo
  - Geralmente constante
- 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
  - Progressão: o fator de ramificação é muito grande
  - Regressão: o fator de ramificação diminui porque apenas ações relevantes são testadas

Exemplo de um problema de busca
Transporte aéreo de cargas

Início: *em*(c*1*, *sfo*) ^ *em*(c*2*, *jfk*) ^ *em*(a*1*, *sfo*) ^ *em*(a*2*, jfk) ^ *carga*(c*1*) ^ *carga*(c*2*) ^ a*vião*(a*1*) ^ a*vião*(a*2*) ^ a*eroporto*(j*fk*) ^ a*eroporto*(*sfo*)

Objetivo: *em*(c*1*, *jfk*) ^ *em*(c*2*, *sfo*)

Ação *carregar(C, A, L)*

Precondição: *em(C, L) ^ em(A, L) ^ carga(C) ^ avião(A) ^ aeroporto(L)*

Lista a apagar: *em(C, L)*

Lista a Inserir: *dentro(C, A)*

Ação *descarregar(C, A, L)*

Precondição: *dentro(C, A) ^ em(A, L) ^ carga(C) ^ avião(A) ^ aeroporto(L)*

Lista a apagar: *dentro*(C, *A*)

Lista a Inserir: *em*(C, *L*)

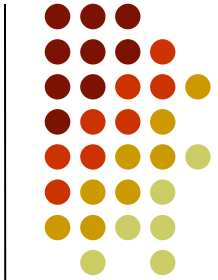
Ação *voar(A, De, Para)*

Precondição: *em(A, De) ^ avião(A) ^ aeroporto(De) ^ aeroporto(Para)*

Lista a apagar: e*m*(A, *De*)

Lista a Inserir: e*m*(A, *Para*)

Busca por Progressão



- Ineficiente
  - Não ataca o problema de ações irrelevantes
  - Gera um fator de ramificação muito grande
- Problema das cargas
  - 10 aeroportos, 5 aviões, 20 itens de carga
  - Mover toda a carga do aeroporto A para B
  - Em média 1000 ações possíveis
  - Média de 1000^{41} nós na árvore de busca
- Precisa de uma heurística muito precisa

Busca por Regressão

- Vantagem: Permite considerar apenas ações **relevantes**
  - Menor fator de ramificação
  - Exemplo do problema de cargas: 1000 ações para a frente e 20 ações para trás
- Uma ação é relevante para um objetivo conjuntivo se ela alcança um dos elementos da conjunção do objetivo
  - Exemplo do problema de cargas: 10 aeroportos e 20 itens em B
    - $em(c_1, sfo) \wedge em(c_2, sfo) \wedge em(c_3, sfo) \wedge \ldots \wedge em(c_{20}, sfo)$
    - Podemos buscar por ações que tem cada elemento da conjunção como efeito
    - Ex: d*escarregar*(c_1, A, sfo) tem como efeito e$m(c_1, sfo)$
- Quais são os estados a partir dos quais a aplicação de uma dada ação leva ao objetivo?

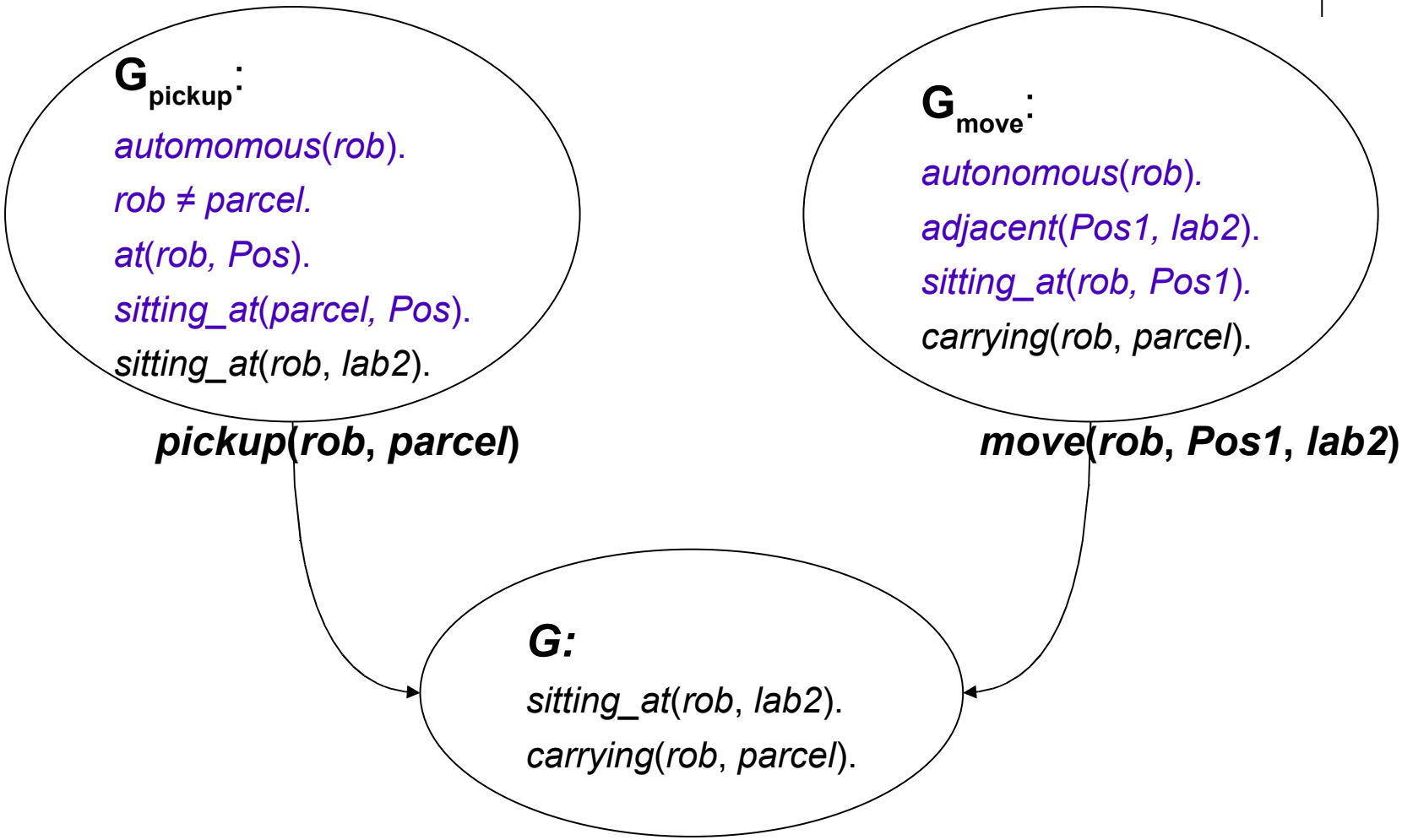
Planejamento por Regressão

- 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 ser verdade imediatamente antes da ação A de forma que **todos** os subobjetivos serão verdadeiros imediatamente depois de A (precondição mais fraca)
  - Recursivamente resolva os novos objetivos (= precondição mais fraca)
  - Obs: Subobjetivos não podem ser considerados separadamente
    - Alcançar um subobjetivo pode desfazer um subobjetivo já alcançado

Planejamento por Regressão

- 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

Planejamento por Regressão
Espaço de estados



Precondicão mais fraca

- O predicado *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 (com 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*

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

Exemplo de Planejamento por Regressão

- Problema de 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*)]
  - *colocar*(*M*, *Lado*)
    - **Precondições**: [*sem*(*M*, *Lado*)]
    - **Lista a apagar**: [ ]
    - **Lista a adicionar**: [*colocado*(*M*, *Lado*)]

Exemplo de Planejamento por Regressão

Situação inicial: [*sem*(*meia*, *esquerdo*) ^ *sem*(*meia*, *direito*)]

Lista de objetivos inicial G: [*calçado*(*sapato*, *direito*) ^ *calçado*(*sapato*, *esquerdo*)]

Subobjetivo selecionado: *calçado*(*sapato*, *esquerdo*)

Ação: ***calçar*(S, *Lado*)** => unificação {*S*/*sapato*, *Lado*/*esquerdo*}

Precondição mais fraca para **G** e a ação **calçar**:

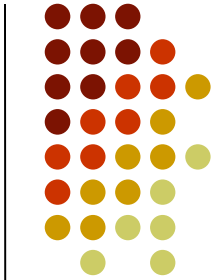
[*calçado*(*sapato*, *direito*) ^ *colocado*(*meia, esquerdo*)]

colocado(*meia*, *esquerdo*)

calçar(*sapato*,*esquerdo*)

calçado(*sapato*, *esquerdo*)

Exemplo de Planejamento por Regressão



Situação inicial: [*sem*(*meia*, *esquerdo*) ^ *sem*(*meia*, *direito*)]

Nova lista de objetivos inicial G1:

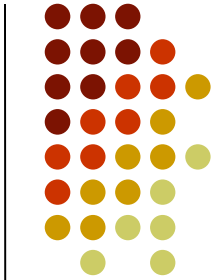
[*calçado*(*sapato*, *direito*) ^ *colocado*(*meia, esquerdo*)]

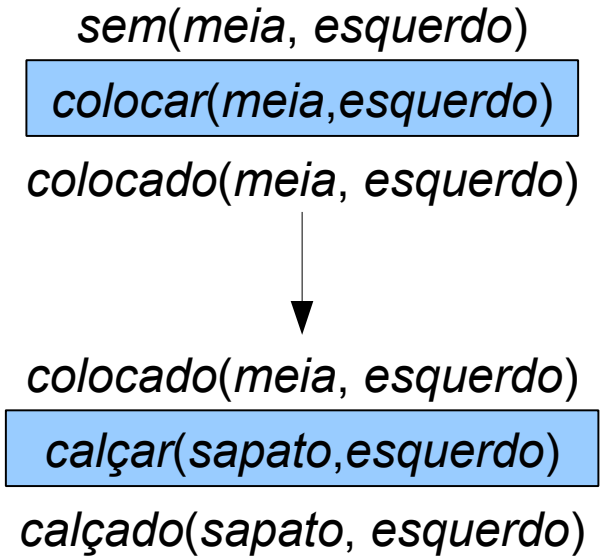
Subobjetivo selecionado: *colocado*(*meia, esquerdo*)

Ação: ***colocar*(*M, Lado*)** => unificação {*M*/*meia*, *Lado*/*esquerdo*}

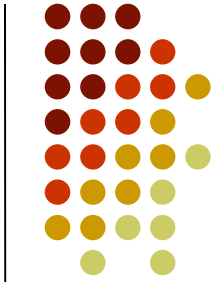
Precondição mais fraca para **G1** e a ação **colocar**:

[*calçado*(*sapato*, *direito*) ^ *sem*(*meia*, *esquerdo*)]




sem(meia, esquerdo)
colocar(meia,esquerdo)
colocado(meia, esquerdo)
colocado(meia, esquerdo)
calçar(sapato,esquerdo)
calçado(sapato, esquerdo)

Exemplo de Planejamento por Regressão



Situação inicial: [*sem*(*meia*, *esquerdo*) ^ *sem*(*meia*, *direito*)]

Nova lista de objetivos inicial G2:

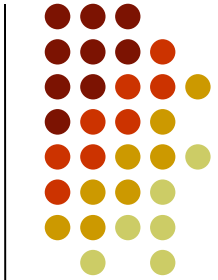
[*calçado*(*sapato*, *direito*) ^ *sem*(*meia*, *esquerdo*)]

Subobjetivo selecionado: *sem*(*meia*, *esquerdo*)

sem(*meia*, *esquerdo*) => verdadeiro no estado inicial

Nova lista de objetivos G3: [*calçado*(*sapato*, *direito*)]

Exemplo de Planejamento por Regressão



Situação inicial: [*sem*(*meia*, *esquerdo*) ^ *sem*(*meia*, *direito*)]

Nova lista de objetivos inicial G3:

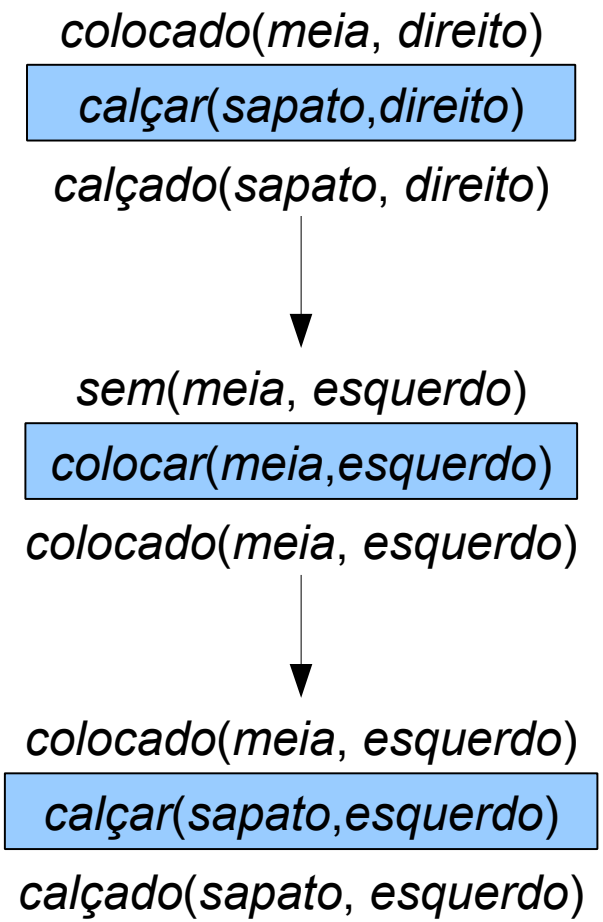
[*calçado*(*sapato*, *direito*)]

Subobjetivo selecionado: *calçado*(*sapato*, *direito*)

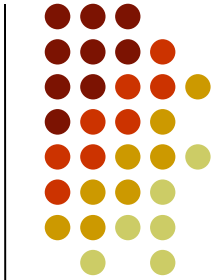
Ação: ***calçar*(*S, Lado*)** => unificação {*S*/*sapato*, *Lado*/*direito*}

Precondição mais fraca para **G3** e a ação **calçar**:

[*colocado*(*meia, direito*)]


colocado(meia, direito)
calçar(sapato,direito)
calçado(sapato, direito)
sem(meia, esquerdo)
colocar(meia,esquerdo)
colocado(meia, esquerdo)
colocado(meia, esquerdo)
calçar(sapato,esquerdo)
calçado(sapato, esquerdo)

Exemplo de Planejamento por Regressão



Situação inicial: [*sem*(*meia*, *esquerdo*) ^ *sem*(*meia*, *direito*)]

Nova lista de objetivos inicial G4:

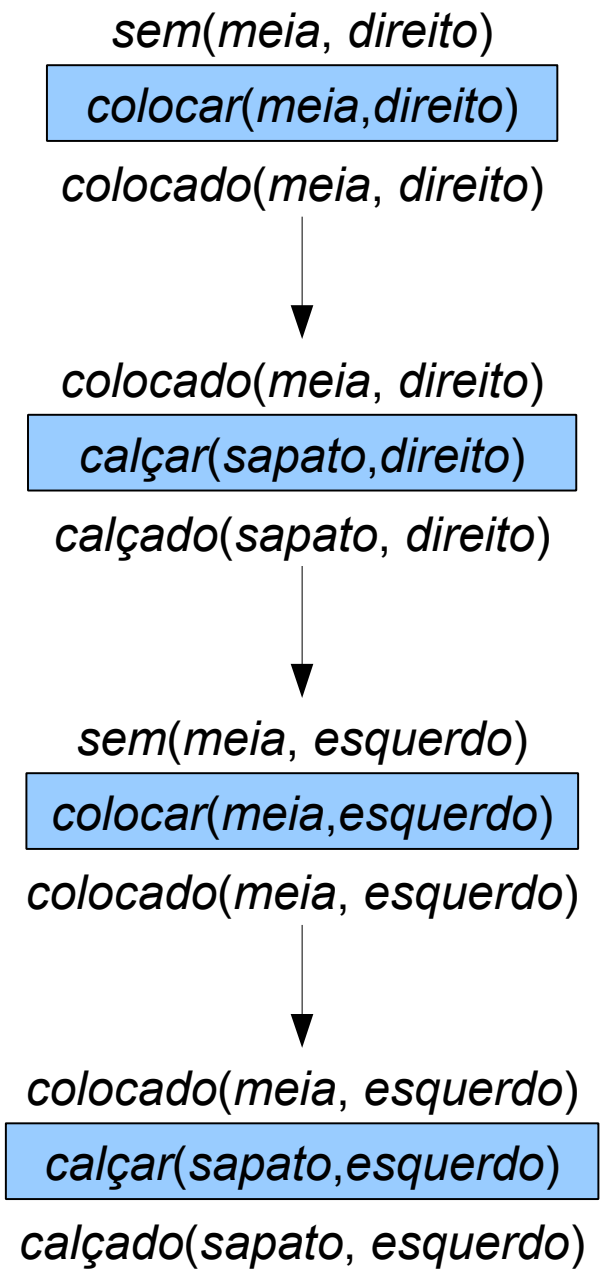
[*colocado*(*meia, direito*)]

Subobjetivo selecionado: [*colocado*(*meia, direito*)]

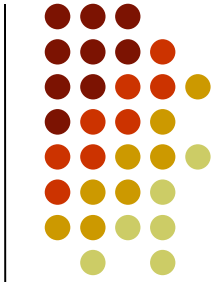
Ação: ***colocar*(*M, Lado*)** => unificação {*M*/meia, *Lado*/*direito*}

Precondição mais fraca para **G4** e a ação **colocar:**

[*sem*(*meia, direito*)]


sem(meia, direito)
colocar(meia,direito)
colocado(meia, direito)
colocado(meia, direito)
calçar(sapato,direito)
calçado(sapato, direito)
sem(meia, esquerdo)
colocar(meia,esquerdo)
colocado(meia, esquerdo)
colocado(meia, esquerdo)
calçar(sapato,esquerdo)
calçado(sapato, esquerdo)

Exemplo de Planejamento por Regressão



Situação inicial: [*sem*(*meia*, *esquerdo*) ^ *sem*(*meia*, *direito*)]

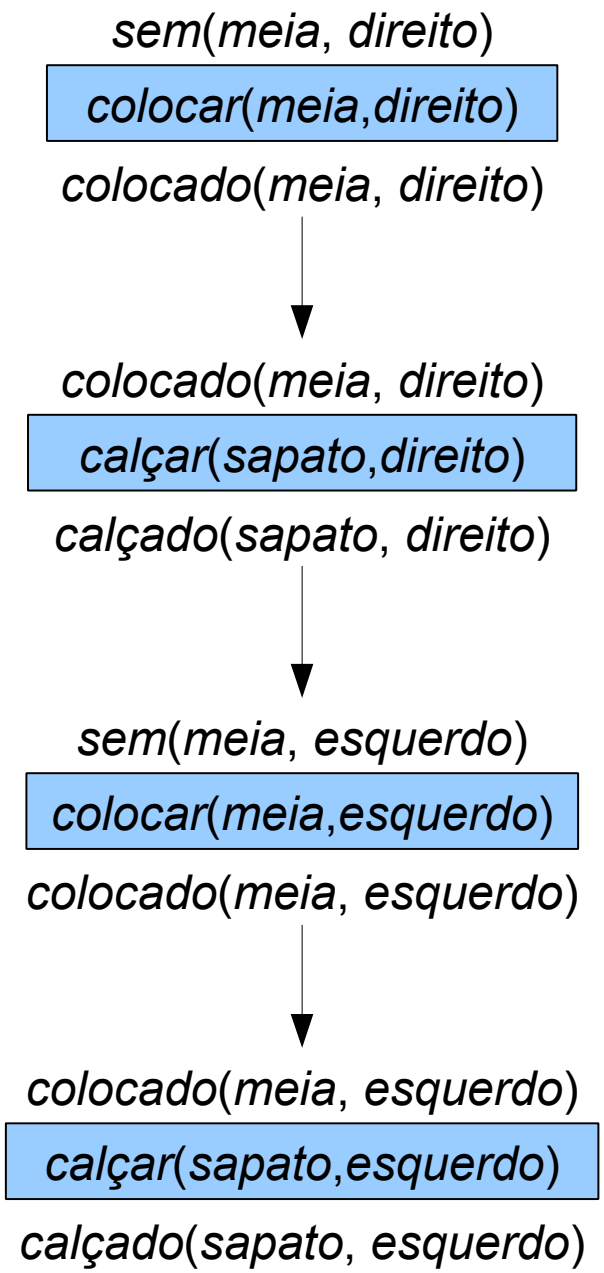
Nova lista de objetivos inicial G5:

[*sem*(*meia, direito*)]

Subobjetivo selecionado: [*sem*(*meia, direito*)]

sem(*meia, direito*) =. verdadeiro no estado inicial

Nova lista de objetivos G6: [ ]


sem(meia, direito)
colocar(meia,direito)
colocado(meia, direito)
colocado(meia, direito)
calçar(sapato,direito)
calçado(sapato, direito)
sem(meia, esquerdo)
colocar(meia,esquerdo)
colocado(meia, esquerdo)
colocado(meia, esquerdo)
calçar(sapato,esquerdo)
calçado(sapato, esquerdo)

Exemplo de Planejamento por Regressão com relações derivadas

Situação inicial: [*sitting_at*(*rob*, *o109*),*sitting_at*(*parcel*, *storage*), *sitting_at*(*k1*, *mail*)]

Lista de objetivos inicial G: [*carrying*(*rob*, *parcel*), ***sitting_at*(*rob*, *lab2)***]

Ação: ***move*(*rob, P*, *lab2*)**

Precondição mais fraca para **G** e a ação **mover**:

[*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *P*), *adjacent*(*P*, *lab2*), *automomous*(*rob*)]

Exemplo de Planejamento por Regressão com relações derivadas

Nova lista de objetivos G1:

[*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *P*), *adjacent*(*P*, *lab2*), *automomous*(*rob*)]

automomous(*rob*) => relação estática verdadeira

adjacent(*P*, *lab2*) => é uma relação derivada

Resolve com $Adjacent(P_1, P_2) \leftarrow between(Door, P_1, P_2) \wedge unlocked(Door)$

Unificação: $\{P_1/P, P_2/lab2\}$

$Adjacent(P, lab2) \leftarrow between(Door, P, lab2) \wedge unlocked(Door)$

Aplicando a relação derivada temos G1:

[*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *P*), *between*(*Door*, *P*, *lab2*), *unlocked*(*Door*)]

between(*Door*, *P*, *lab2*) resolve com *between*(*door1*, o*103*, *lab2*)

Unificação: {*Door/door1*, *P/o103*}

between(*door1*, o*103*, *lab2*) => relação estática verdadeira

Exemplo de Planejamento por Regressão com relações derivadas

Situação inicial: [*sitting_at*(*rob*, *o109*),*sitting_at*(*parcel*, *storage*), *sitting_at*(*k1*, *mail*)]

Lista de objetivos G1:

[*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *o103*), ***unlocked*(*door1*)**]

Ação: ***unlock*(*door1*)**

Precondição mais fraca para **G1** e a ação **unlock**:

[*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *o103*), *autonomous*(*rob*), *between*(*door1*, *o103*, *lab2*), *at*(*rob*, o*103*), *open*(*Key*, *door1*), *carrying*(*rob*, *Key*)]

Exemplo de Planejamento por Regressão com relações derivadas

Nova lista de objetivos G2:

[*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *o103*), *autonomous*(*rob*), *between*(*door1*, *o103*, *lab2*), *at*(*rob*, o*103*), *open*(*Key*, *door1*), *carrying*(*rob*, *Key*)]

autonomous(*rob*) => relação estática verdadeira

between(*door1*, o*103*, *lab2*) => relação estática verdadeira

at(*rob*, o*103*) => é uma relação derivada

Resolve com *at*(*Obj*, *Pos*) ← *sitting_at*(*Obj*, *Pos*) unificação {Obj/rob, Pos/o103}

at(*rob*, o*103*) ← *sitting_at*(*rob*, *o103*)

sitting_at(*rob*, *o103*) => verdadeiro em G2

open(*Key*, *door1*) resolve com *open*(*k1, door1*) => relação estática verdadeira

Aplicando as relações derivadas e retirando as relações estáticas verdadeiras temos G2: [*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *o103*), *carrying*(*rob*, *k1*)]

Exemplo de Planejamento por Regressão com relações derivadas

Situação inicial: [*sitting_at*(*rob*, *o109*),*sitting_at*(*parcel*, *storage*), *sitting_at*(*k1*, *mail*)]

Lista de objetivos G2:

[*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *o103*), ***carrying*(*rob*, *k1*)**]

Ação: ***pickup*(*rob*, *k1*)**

Precondição mais fraca para **G2** e a ação **pickup**:

[*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *o103*), *autonomous*(*rob*),rob ≠ *k1*, *at*(*rob*, *Pos*), *sitting_at*(*rob*, *Pos*)]

Exemplo de Planejamento por Regressão com relações derivadas

Nova lista de objetivos G3:

[*carrying*(*rob*, *parcel*), *sitting_at*(*rob*, *o103*), *autonomous*(*rob*), rob ≠ *k1*, *at*(*rob*, *Pos*), *sitting_at*(*rob*, *Pos*)]

autonomous(*rob*) => relação estática verdadeira

rob ≠ *k1* => verdadeira

... continua até que a lista de objetivos Gx seja vazia.

Exercício – 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*)]