PICAT: Uma Linguagem de Programação Multiparadigma

Claudio Cesar de Sá

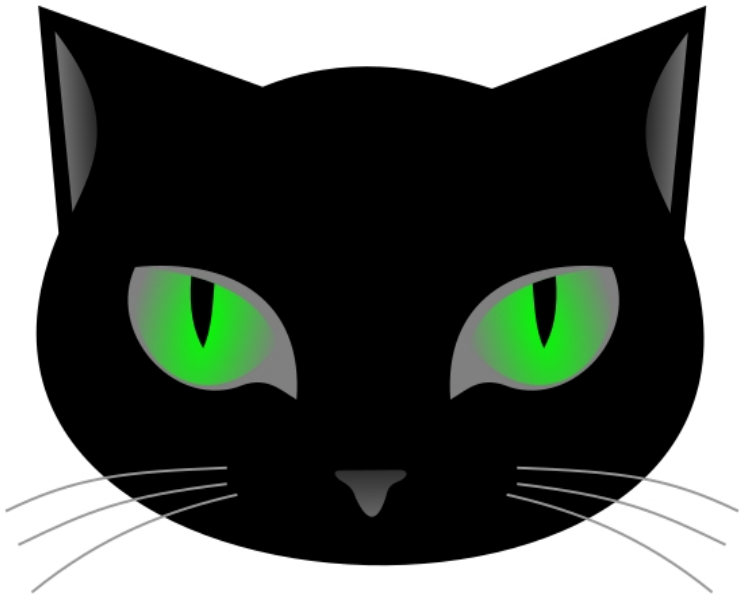
claudio.sa@udesc.br

Departamento de Ciência da Computação – DCC
Centro de Ciências e Tecnológias – CCT
Universidade do Estado de Santa Catarina – UDESC

9 de maio de 2019

Planejamento

- O que é Planejamento?
- Importância da área
- Muitas definições
- Exemplo



Planejamento

- Requisitos: recursividade, listas e PD

Planejamento

- Requisitos: recursividade, listas e PD
- Além destes: conceitos grafos, árvores de busca, nós, etc

Planejamento

- Requisitos: recursividade, listas e PD
- Além destes: conceitos grafos, árvores de busca, nós, etc
- *Planejamento* é um **termo amplo e em vários domínios**

Planejamento

- Requisitos: recursividade, listas e PD
- Além destes: conceitos grafos, árvores de busca, nós, etc
- *Planejamento* é um **termo amplo e em vários domínios**
- O que **não** é o nosso contexto de *planejamento*?
  Exemplo: planejamento estratégico das empresas, planejar como distribuir os dividendos da empresa, orçamento familiar, etc

Planejamento

- Requisitos: recursividade, listas e PD
- Além destes: conceitos grafos, árvores de busca, nós, etc
- *Planejamento* é um **termo amplo e em vários domínios**
- O que **não** é o nosso contexto de *planejamento*?
  Exemplo: planejamento estratégico das empresas, planejar como distribuir os dividendos da empresa, orçamento familiar, etc
- O que é o nosso contexto de *planejamento*?

Planejamento

- Requisitos: recursividade, listas e PD
- Além destes: conceitos grafos, árvores de busca, nós, etc
- *Planejamento* é um **termo amplo e em vários domínios**
- O que **não** é o nosso contexto de *planejamento*?
  Exemplo: planejamento estratégico das empresas, planejar como distribuir os dividendos da empresa, orçamento familiar, etc
- O que é o nosso contexto de *planejamento*? Questões que envolvam um ambiente, um agente (um programa, um robô, etc), sensores, e ações que modifiquem estados.
  Exemplo clássico: robótica em geral

- Problemas em geral necessitam de um **plano** para serem solucionados, assim, há uma visão que encontrar um plano para um problema $\Rightarrow$ ter uma solução!

- Problemas em geral necessitam de um **plano** para serem solucionados, assim, há uma visão que encontrar um plano para um problema $\Rightarrow$ ter uma solução!
- Em resumo, a área de planejamento é bem complexa, antiga na área da IA e robótica (1970 – STRIPS), efervescente, e de muito interesse na indústria.

- Problemas em geral necessitam de um **plano** para serem solucionados, assim, há uma visão que encontrar um plano para um problema $\Rightarrow$ ter uma solução!
- Em resumo, a área de planejamento é bem complexa, antiga na área da IA e robótica (1970 – STRIPS), efervescente, e de muito interesse na indústria.
- Várias abordagens sobre a visão clássica da IA. Mas temos evoluções significativas ...

- Problemas em geral necessitam de um **plano** para serem solucionados, assim, há uma visão que encontrar um plano para um problema $\Rightarrow$ ter uma solução!
- Em resumo, a área de planejamento é bem complexa, antiga na área da IA e robótica (1970 – STRIPS), efervescente, e de muito interesse na indústria.
- Várias abordagens sobre a visão clássica da IA. Mas temos evoluções significativas ...
- PDDL (*Planning Domain Definition Language*): unanimidade (ou próxima a esta) entre os pesquisadores de planejamento, como linguagem descritora de problemas de planejamento.

Planejamento

- Problemas em geral necessitam de um **plano** para serem solucionados, assim, há uma visão que encontrar um plano para um problema $\Rightarrow$ ter uma solução!
- Em resumo, a área de planejamento é bem complexa, antiga na área da IA e robótica (1970 – STRIPS), efervescente, e de muito interesse na indústria.
- Várias abordagens sobre a visão clássica da IA. Mas temos evoluções significativas ...
- PDDL (*Planning Domain Definition Language*): unanimidade (ou próxima a esta) entre os pesquisadores de planejamento, como linguagem descritora de problemas de planejamento.
- Vários problemas ainda sem solução, pois a complexidade é exponencial

Definições

- Plano: seqüência ordenada de ações

Definições

- Plano: seqüência ordenada de ações
  - problema: escalar o Everest, comprar um abacate, leite e uma furadeira (nesta ordem)

- Plano: seqüência ordenada de ações
  - problema: escalar o Everest, comprar um abacate, leite e uma furadeira (nesta ordem)
  - plano: ir ao supermercado, ir à seção de frutas, pegar as bananas, ir à seção de leite, pegar uma caixa de leite, ir ao caixa, pagar tudo, ir a uma loja de ferramentas, ..., voltar para casa.

Definições

- Plano: seqüência ordenada de ações
  - problema: escalar o Everest, comprar um abacate, leite e uma furadeira (nesta ordem)
  - plano: ir ao supermercado, ir à seção de frutas, pegar as bananas, ir à seção de leite, pegar uma caixa de leite, ir ao caixa, pagar tudo, ir a uma loja de ferramentas, ..., voltar para casa.
- Um Planejador: Combina conhecimento de um ambiente, um agente e suas ações possíveis, entradas (luz, cor, cheiro, sensor, etc), um estado corrente e/ou inicial, e com isto resolve de problemas planejar sequência de ações, que mudam de estados a cada ação, até atingir um estado final.

Exemplos do que é planejamento ...

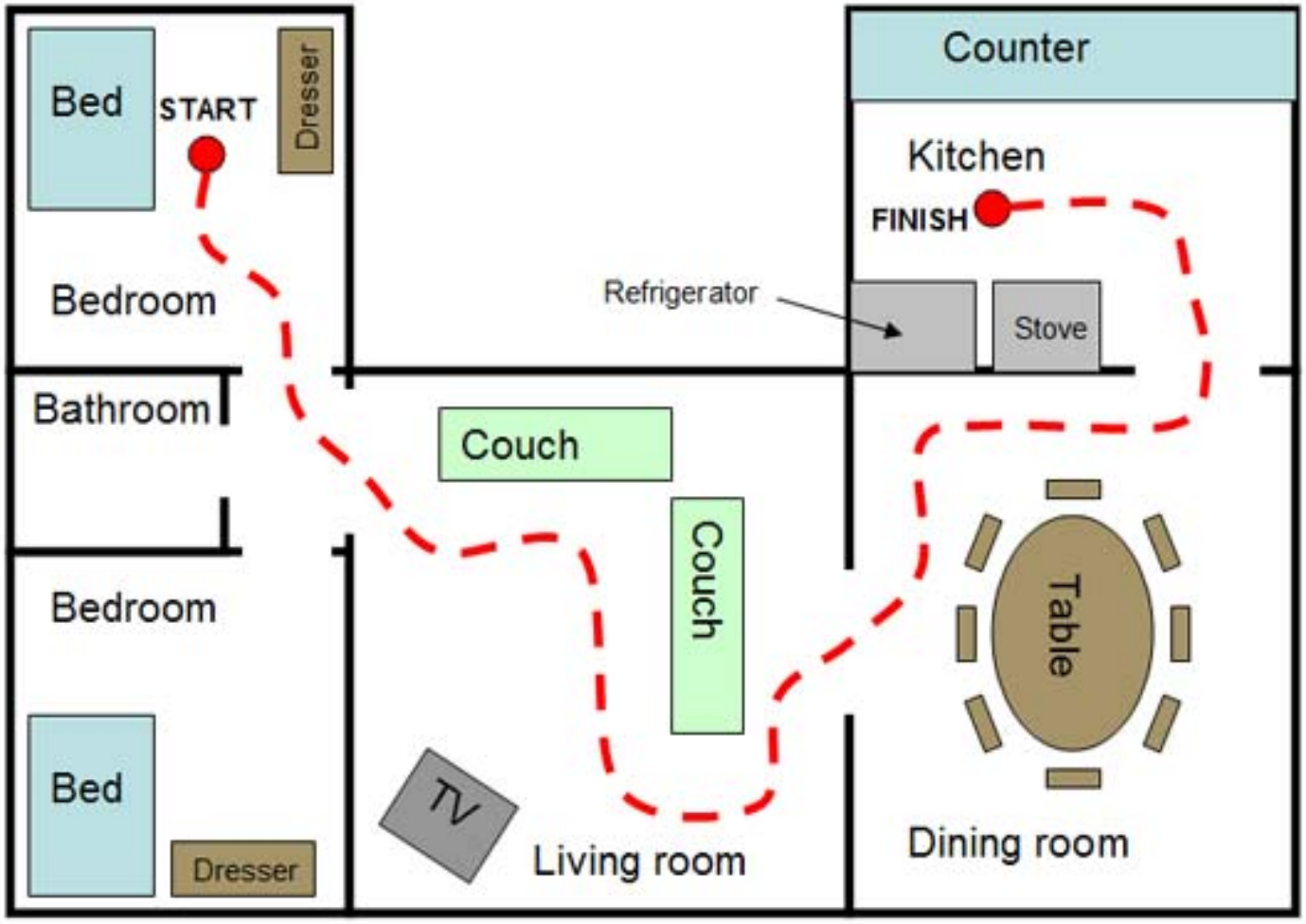


Figura 1: *A fome no meio da noite!*

Exemplos do que é planejamento ...

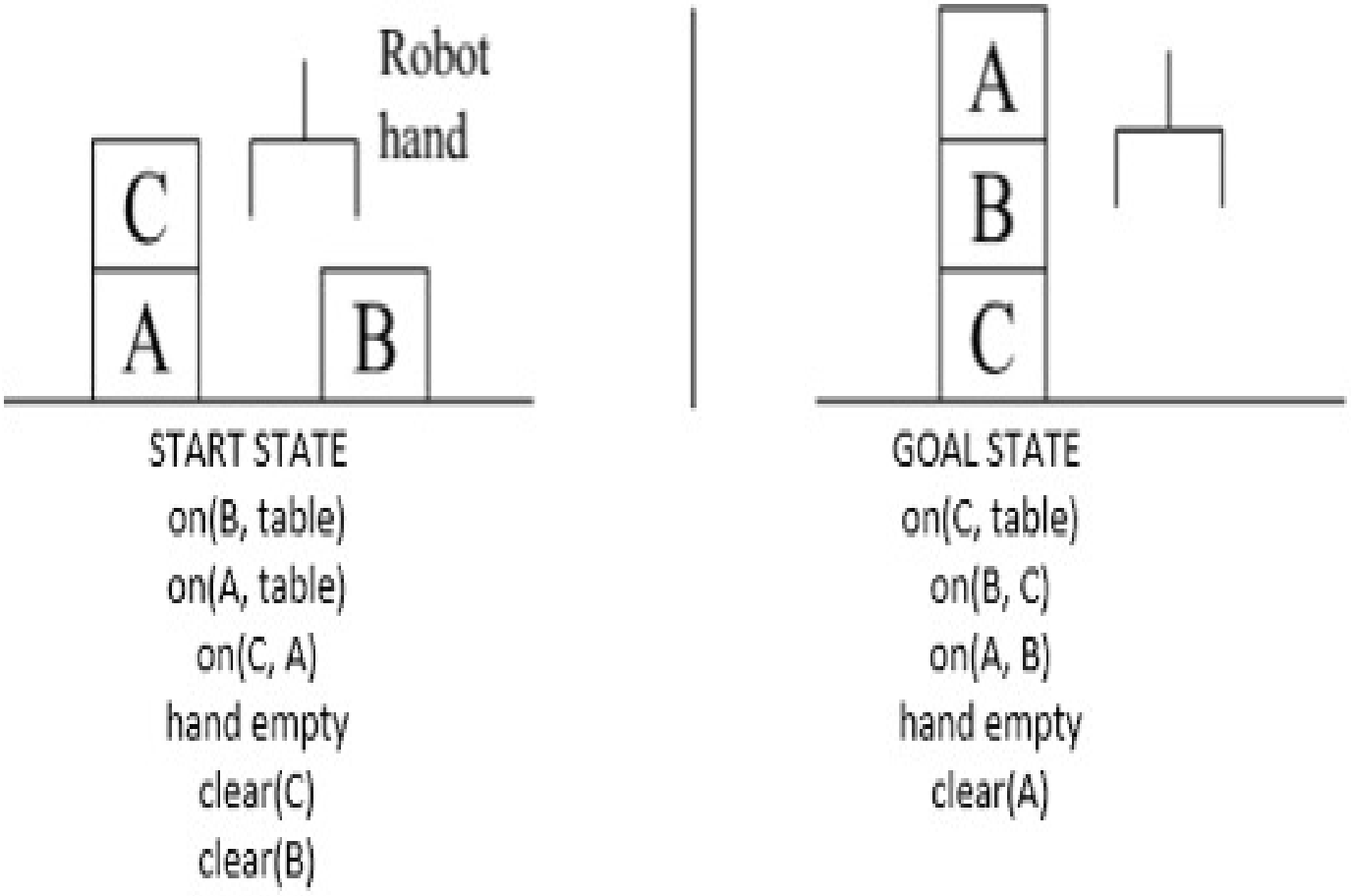


Figura 2: O *mundo dos blocos*

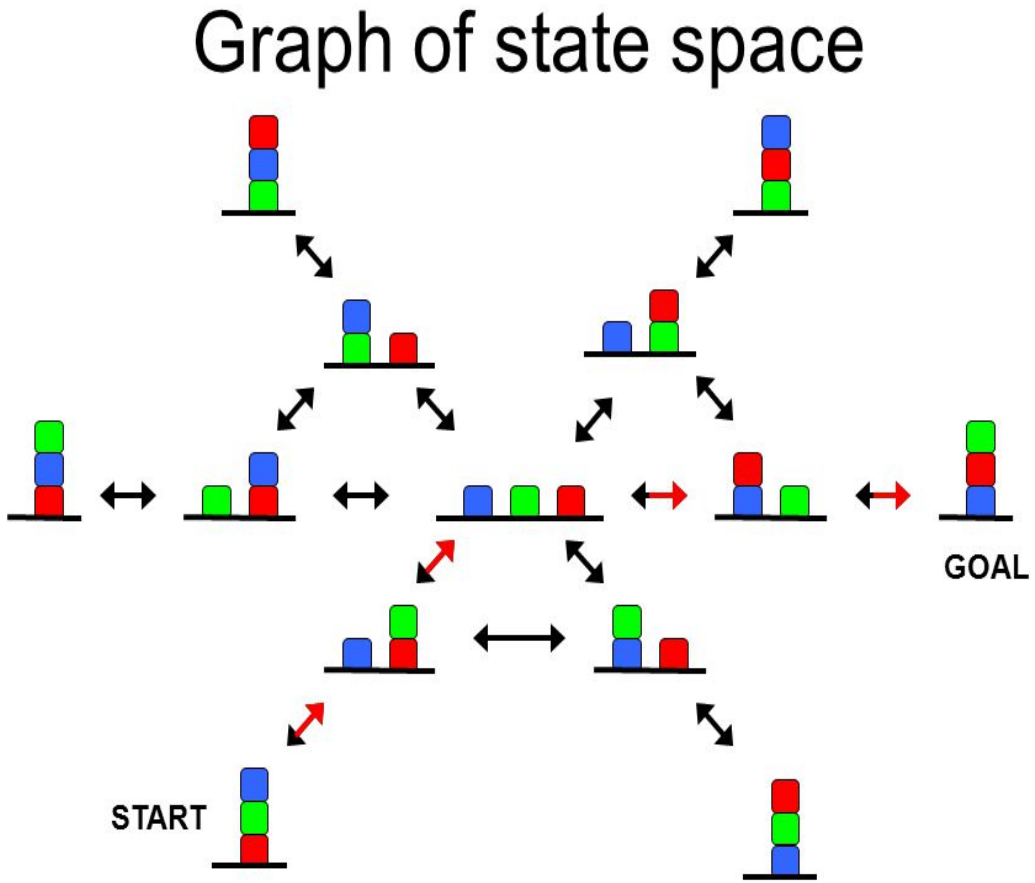


Figura 3: O espaço de estados do *mundo dos blocos* × ações

Elementos de um Planejador – Vocabulário I

- Plano: uma sequência ordenada de ações, criada incrementalmente a partir do estado inicial
  Ex. posições das peças de um jogo

$$S_1 < S_2 < ... < S_n$$

- Ambiente: onde um programa–agente vai receber entradas em um determinado estado e atuar com uma ação apropriada
- Estados: descrição completa de possíveis estados atingíveis
  Problema: quanto aos estados não-previstos, inacessíveis?
- Estado inicial: um estado particular onde nosso programa–agente inicia a sua busca
- Objetivos: estados desejados que o programa–agente precisa alcançar, isto é, um dos *estados finais* desejados

Elementos de um Planejador – Vocabulário II

- Percepções: cheiro, brisa, luz, choque, som, posições ou coordenadas, vizinhanças, etc
- Ações: provocam modificações entre os estados corrente e sucessor
  Exemplos: avançar para próxima célula, girar 90 graus à direita ou à esquerda pegar um objeto, atirar na direção do alvo, etc
- Operadores: vocabulário ou repertório de atuações atômicas do que o agente pode fazer.
  Exemplos: $pegar(X)$, $mover\_de(X, Y)$, $levantar(X)$, $livre(X)$, etc
- Uma eventual confusão: **uma ação é um conjunto de um ou mais operadores**, e ainda, **a ação é condicional**. A ação só é disparada se as condições de pré-requisitos forem satisfeitas.

Elementos de um Planejador – Vocabulário III

- Heurística: alguma função que indica o progresso sobre os estados não visitados e sua convergência para uma finalização do plano

Figura 4: Um quebra-cabeça (2×3 ou 3×2) *simplificado* do conhecido 3×3

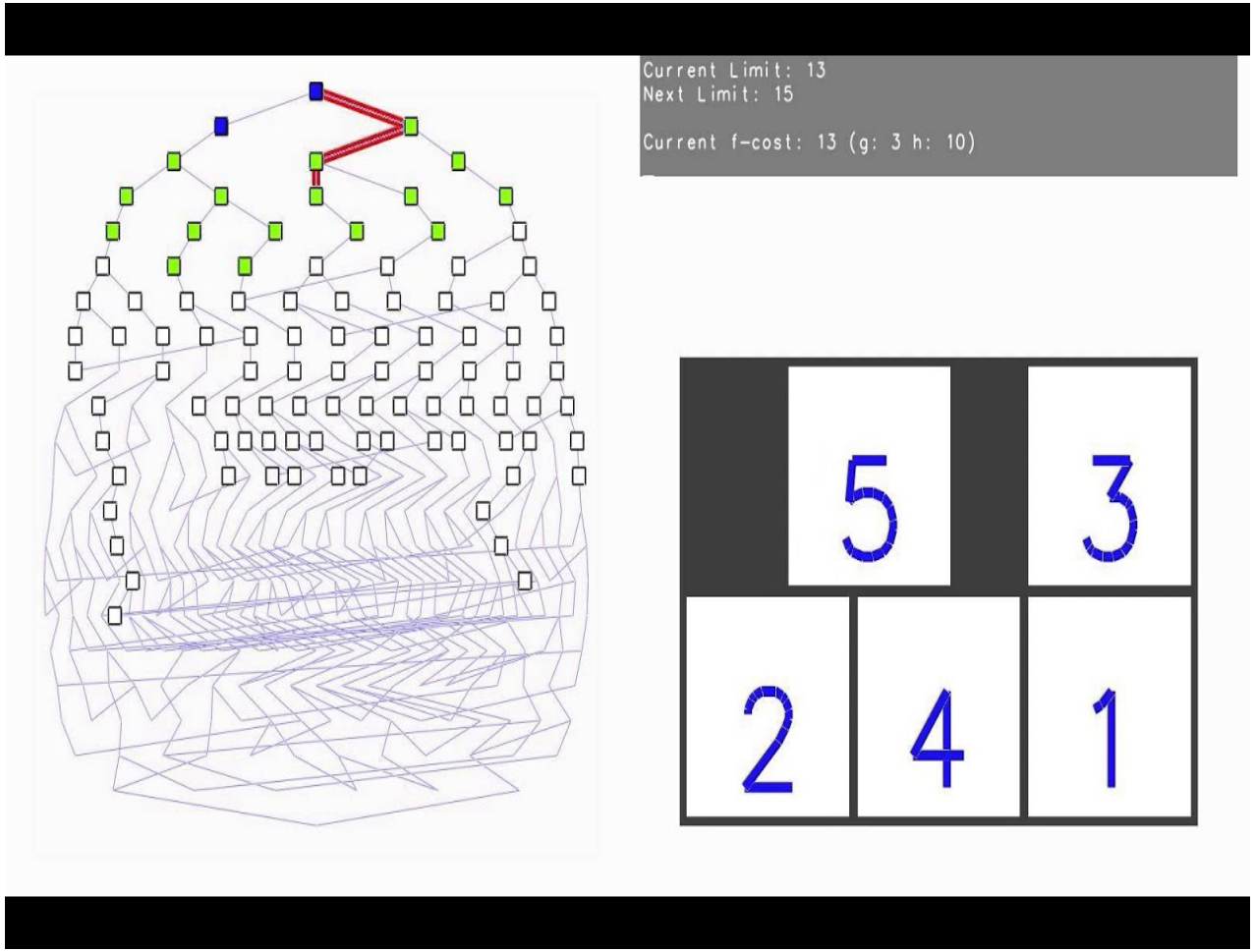


Figura 5: Sim, *simplificado* mas não muito!

Partes do código comentado I

```
/*
A  B  C
D  E  F
*/

%%%%%%%%
%   1 5 %
% 4 3 2 %
%%%%%%%%

import datetime.
import planner.
```

Atenção quanto a modelagem do problema, as 3 primeiras posicões da lista correspondem a linha superior (A .. C), e as 3 últimas, a linha inferior (D .. F).

Partes do código comentado II

```
index(-)
estado_inicial( [0,1,5,4,3,2]  ).
%%%%%%%%%%%%%==> A,B,C,D,E,F

%% funcao final do planner
final( [1,2,3,4,5,0] ) => true .
%%%%==> A,B,C,D,E,F
%% pode ter uma condicional de parada
```

Partes do código comentado III

```
% Up <-> Down
/* Descrevendo as possiveis acoes para o planner */
action([A,B,C, D,E,F], S1, Acao, Custo_Acao ) ?=>
    Custo_Acao = 1,
    ( A == 0 ),  %% conj. condicoes
    S1 = [D,B,C, 0,E,F],
    Acao = ($up(D),S1). %%a acao + estado modificado

action([A,B,C, D,E,F], S1, Acao, Custo_Acao ) ?=>
    Custo_Acao = 1,
    (A == 0 ),  %% conj. condicoes
    S1 = [0,B,C, A,E,F],
    Acao = ($dow(A),S1). %%a acao + estado modificado
.....................................................
```

Partes do código comentado IV

```
% Left <-> Right
action([A,B,C, D,E,F], S1, Acao, Custo_Acao ) ?=>
    Custo_Acao = 1,
    (A == 0),  %% conj. condicoes
    S1 = [B,0,C, D,E,F],
    Acao = ($left(B), S1). %%a acao + estado modificado

action([A,B,C, D,E,F], S1, Acao, Custo_Acao ) ?=>
    Custo_Acao = 1,
    (B == 0),  %% conj. condicoes
    S1 = [0,A,C, D,E,F],
    Acao = ($right(A), S1). %%a acao + estado modificado
.....................................................
```

Partes do código comentado V

```
main  ?=>
    estado_inicial( Q ),
    best_plan_unbounded( Q , Sol_Acoes),
    println(sol = Sol_Acoes),

    printf("\n Estado Inicial: "),
    w_Quadro( Q ),
    w_L_Estado( Sol_Acoes ),
    Total := length(Sol_Acoes) ,
    Num_Movts := (Total -1) ,
    printf("\n Inicial  (estado): %w ", Q),
    printf("\n Total de acoes: %d", Total),
    printf(" \n ========================================\n ")
    %%% fail ou false: descomente para multiplas solucoes
    .
 main => printf("\n Para uma solução .... !!!!" ) .
```

O código

- Acompanhar as explicações do código de: `https://github.com/claudiosa/CCS/blob/master/picat/puzzle_2x3_planner.pi`
- Confira a execução

Parte da Saída I

```
[ccs@gerzat picat]$ picat puzzle_2x3_planner.pi
sol = [(left(1),[1,0,5,4,3,2]),(left(5),[1,5,0,4,3,2]),
(up(2),[1,5,2,4,3,0]),(right(3),[1,5,2,4,0,3]),(dow(5),[1,0,2,4,5,3]),
(left(2),[1,2,0,4,5,3]),(up(3),[1,2,3,4,5,0])]

 Estado Inicial:
 0 1 5
 4 3 2

Acao: left(1)
 1 0 5
 4 3 2

Acao: left(5)
 1 5 0
 4 3 2
```

Parte da Saída II

```
.................................
Acao: left(2)
 1 2 0
 4 5 3

Acao: up(3)
 1 2 3
 4 5 0

 Inicial  (estado): [0,1,5,4,3,2]
 Total de acoes: 7
 ========================================
```

O módulo do *planner* do Picat

- O que efetivamente voce precisa saber

O módulo do *planner* do Picat

- O que efetivamente voce precisa saber
- Importar um módulo
  `import planner.`

O módulo do *planner* do Picat

- O que efetivamente voce precisa saber
- Importar um módulo
  `import planner.`
- O predicado: *final*

```
final(S,Plan,Cost) => Plan=[], Cost=0, final(S).
```

O módulo do *planner* do Picat

- O que efetivamente voce precisa saber
- Importar um módulo
  `import planner.`
- O predicado: *final*
  `final(S,Plan,Cost) => Plan=[], Cost=0, final(S).`
- O predicado *action*
  `action(S,NextS,Action,ActionCost)`

O módulo do *planner* do Picat

- A **eficiência** do planner do Picat se dá devido a sua combinação de técnicas: busca em profundidade (e variações) e Programação Dinâmica (PD) (uso do *tabling*)

O módulo do *planner* do Picat

- A **eficiência** do planner do Picat se dá devido a sua combinação de técnicas: busca em profundidade (e variações) e Programação Dinâmica (PD) (uso do *tabling*)
- O núcleo de busca dos planejadores disponíveis no Picat são de 2 tipos:
  1. Usam um busca em profundidade com limites (*Depth-Bounded Search*)
  2. Usam um busca em profundidade ilimitada de recursos (*Depth-Unbounded Search*)
- Contudo, estes 2 tipos apresentam muitas variações e opções:

O módulo do *planner* do Picat

- A **eficiência** do planner do Picat se dá devido a sua combinação de técnicas: busca em profundidade (e variações) e Programação Dinâmica (PD) (uso do *tabling*)
- O núcleo de busca dos planejadores disponíveis no Picat são de 2 tipos:
  1. Usam um busca em profundidade com limites (*Depth-Bounded Search*)
  2. Usam um busca em profundidade ilimitada de recursos (*Depth-Unbounded Search*)
- Contudo, estes 2 tipos apresentam muitas variações e opções: Sem escapatória $\Rightarrow$ consultar o manual do Picat (*User Guide to Picat*)
- No exemplo aqui apresentado:
  `best_plan_unbounded(S,Plan)`

- Planejamento resolve uma classe ampla de problemas
  Havendo necessidade de **descobrir sequências ações** $\Leftrightarrow$ Planejamento

Reflexões

- Planejamento resolve uma classe ampla de problemas
  Havendo necessidade de **descobrir sequências ações** $\Leftrightarrow$ Planejamento
- Em geral, estes problemas são importantes na indústria

Reflexões

- Planejamento resolve uma classe ampla de problemas
  Havendo necessidade de **descobrir sequências ações** $\Leftrightarrow$ Planejamento
- Em geral, estes problemas são importantes na indústria
- Os modelos escritos em PDDL (*Planning Domain Definition Language*) facilmente portáveis para Picat

Reflexões

- Planejamento resolve uma classe ampla de problemas
  Havendo necessidade de **descobrir sequências ações** $\Leftrightarrow$ Planejamento
- Em geral, estes problemas são importantes na indústria
- Os modelos escritos em PDDL (*Planning Domain Definition Language*) facilmente portáveis para Picat
- Sob um uso mais restrito, um modelo em PDDL é executado diretamente em Picat

Reflexões

- Planejamento resolve uma classe ampla de problemas
  Havendo necessidade de **descobrir sequências ações** $\Leftrightarrow$ Planejamento
- Em geral, estes problemas são importantes na indústria
- Os modelos escritos em PDDL (*Planning Domain Definition Language*) facilmente portáveis para Picat
- Sob um uso mais restrito, um modelo em PDDL é executado diretamente em Picat
- Na próxima seção uma outra técnica de resolver problemas: **PR**