PICAT: Uma Linguagem de Programação Multiparadigma

Claudio Cesar de Sá

✉ ccs1664@gmail.comyahoo.com

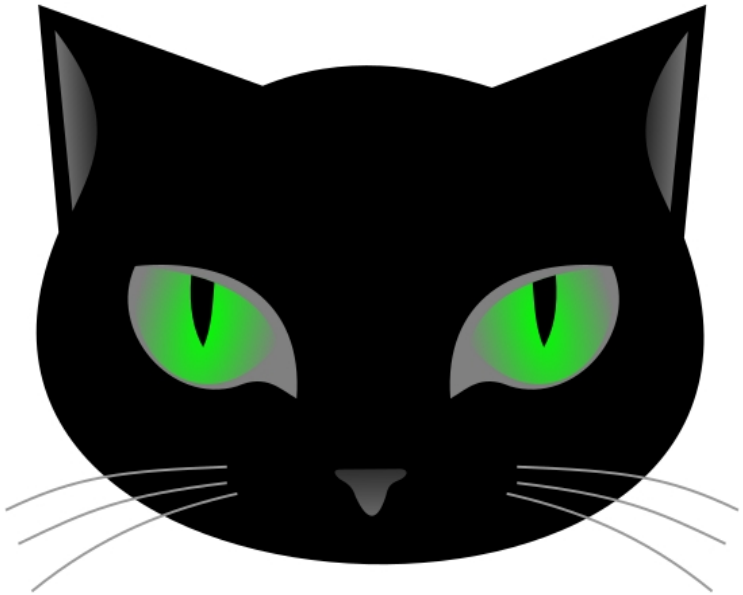
Pesquisador Independente

31 de maio de 2020

Sumário I

Introdução

- Histórico
- Contexto
- Exemplo: *Alo Mundo*
- Como usar
- Site e recursos



Histórico

- Criada em 2013 por Neng-Fa Zhou e Jonathan Fruhman
- Utiliza o B-Prolog como base de implementação, tendo a Lógica de Primeira-Ordem (LPO) como parte de seu mecanismo programação

Histórico

- Criada em 2013 por Neng-Fa Zhou e Jonathan Fruhman
- Utiliza o B-Prolog como base de implementação, tendo a Lógica de Primeira-Ordem (LPO) como parte de seu mecanismo programação
- Uma evolução ao Prolog após seus mais de 40 anos de sucesso!

Histórico

- Criada em 2013 por Neng-Fa Zhou e Jonathan Fruhman
- Utiliza o B-Prolog como base de implementação, tendo a Lógica de Primeira-Ordem (LPO) como parte de seu mecanismo programação
- Uma evolução ao Prolog após seus mais de 40 anos de sucesso!
- Sua atual versão é a 2.x (31 de maio de 2020).

Histórico

- Criada em 2013 por Neng-Fa Zhou e Jonathan Fruhman
- Utiliza o B-Prolog como base de implementação, tendo a Lógica de Primeira-Ordem (LPO) como parte de seu mecanismo programação
- Uma evolução ao Prolog após seus mais de 40 anos de sucesso!
- Sua atual versão é a 2.x (31 de maio de 2020).
- Código-aberto, segue as regras da FSF

Links do PICAT

- Site oficial: `http://picat-lang.org`
- **Videoaula 01: Introdução ao PICAT**, disponível no Youtube:
  `https://www.youtube.com/watch?v=0DmTyFFQPK8`

Links do PICAT

- Site oficial: `http://picat-lang.org`
- **Videoaula 01: Introdução ao PICAT**, disponível no Youtube:
  `https://www.youtube.com/watch?v=0DmTyFFQPK8`
- Editor on-line mantido pelo Alexandre:
  `http://retina.inf.ufsc.br/picat.html`

Links do PICAT

- Site oficial: `http://picat-lang.org`
- **Videoaula 01: Introdução ao PICAT**, disponível no Youtube:
  `https://www.youtube.com/watch?v=0DmTyFFQPK8`
- Editor on-line mantido pelo Alexandre:
  `http://retina.inf.ufsc.br/picat.html`
- Se não tiver *plugin* para Picat, escolha a sintaxe da linguagem *Erlang*.

Conhecendo PICAT

- Picat é uma linguagem de programação simples de usar, poderosa e multi-uso
- Alguma de suas características são associadas com linguagens lógicas, como Prolog, B-Prolog, Goedel, etc

Conhecendo PICAT

- Picat é uma linguagem de programação simples de usar, poderosa e multi-uso
- Alguma de suas características são associadas com linguagens lógicas, como Prolog, B-Prolog, Goedel, etc
- Picat é uma linguagem essencialmente multiparadigma, abrangendo partes de vários paradigmas de programação: declarativo (lógico e funcional) e imperativo

O que é ser Multiparadigma ?

- Paradigma: um conjunto de características baseado em alguma abordagem teórica

O que é ser Multiparadigma ?

- Paradigma: um conjunto de características baseado em alguma abordagem teórica
- Picat é uma linguagem multiparadigma pois abrange os seguintes paradigmas:
  - Lógico
  - Funcional
  - Procedural

O que é ser Multiparadigma ?

- Paradigma: um conjunto de características baseado em alguma abordagem teórica
- Picat é uma linguagem multiparadigma pois abrange os seguintes paradigmas:
  - Lógico
  - Funcional
  - Procedural
- Em resumo, *uma boa mistura* de: Haskell (Funcional) , Prolog (Lógica) e Python (Procedural e Funcional).

Paradigma Lógico

- Uma linguagem lógica é uma onde o programa é expresso como um conjunto de predicados lógicos, escritos por *fatos* e *regras*

Paradigma Lógico

- Uma linguagem lógica é uma onde o programa é expresso como um conjunto de predicados lógicos, escritos por *fatos* e *regras*
- Regras são escritas em formas de cláusulas, as quais são interpretadas como implicações lógicas.
  Dependem das premissas serem verdadeiras para esta ser verdadeira.

Paradigma Lógico

- Uma linguagem lógica é uma onde o programa é expresso como um conjunto de predicados lógicos, escritos por *fatos* e *regras*
- Regras são escritas em formas de cláusulas, as quais são interpretadas como implicações lógicas.
  Dependem das premissas serem verdadeiras para esta ser verdadeira.
- Fatos são cláusulas sem premissas, verdades absolutas.

Paradigma Lógico

- Uma linguagem lógica é uma onde o programa é expresso como um conjunto de predicados lógicos, escritos por *fatos* e *regras*
- Regras são escritas em formas de cláusulas, as quais são interpretadas como implicações lógicas.
  Dependem das premissas serem verdadeiras para esta ser verdadeira.
- Fatos são cláusulas sem premissas, verdades absolutas.
- Este paradigma é a **base** do Picat

Paradigma Funcional

- Uma linguagem funcional é uma onde os elementos do programa podem ser avaliados e tratados como funções matemáticas.

Paradigma Funcional

- Uma linguagem funcional é uma onde os elementos do programa podem ser avaliados e tratados como funções matemáticas.
- Um dos principais motivos em usar linguagens funcionais é a previsibilidade e facilidade no entendimento do estado atual do programa.

Paradigma Funcional

- Uma linguagem funcional é uma onde os elementos do programa podem ser avaliados e tratados como funções matemáticas.
- Um dos principais motivos em usar linguagens funcionais é a previsibilidade e facilidade no entendimento do estado atual do programa.
- Este fato de uma sintaxe simples, torna o Picat intuitivo e legível na funcionalidade de seus códigos.

Paradigma Procedural

- Uma linguagem procedural é uma que pode ser subdividida em *procedimentos*, também chamados de rotinas, subrotinas ou funções

Paradigma Procedural

- Uma linguagem procedural é uma que pode ser subdividida em *procedimentos*, também chamados de rotinas, subrotinas ou funções
- Em linguagens procedurais há um procedimento principal (em geral é chamado de *Main*) que controla o uso e a chamada de outros procedimentos. Em Picat há tal hierarquia.

Paradigma Procedural

- Uma linguagem procedural é uma que pode ser subdividida em *procedimentos*, também chamados de rotinas, subrotinas ou funções
- Em linguagens procedurais há um procedimento principal (em geral é chamado de *Main*) que controla o uso e a chamada de outros procedimentos. Em Picat há tal hierarquia.
- Em Picat, cada premissa é tratada como um procedimento, que é resolvido por meio de métodos de inferência lógica.

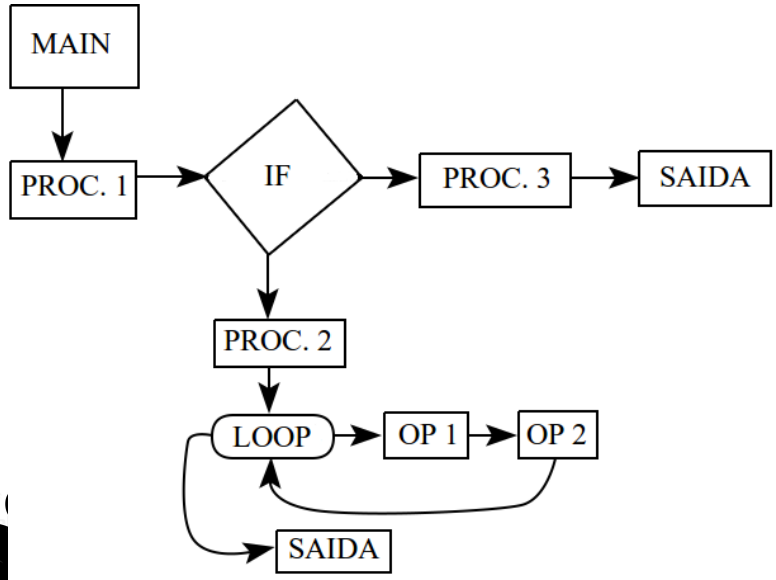


Figura 1: Fluxograma representando a estrutura de um programa Procedural

Algumas Características:

Algumas Características:

- Sintaxe elegante e simples, facilitando a leitura e entendimento do código

Algumas Características:

- Sintaxe elegante e simples, facilitando a leitura e entendimento do código
- Velocidade de execução em um ambiente *interpretado* (há uma *máquina virtual* como Python, Java e alguns Prologs)

Algumas Características:

- Sintaxe elegante e simples, facilitando a leitura e entendimento do código
- Velocidade de execução em um ambiente *interpretado* (há uma *máquina virtual* como Python, Java e alguns Prologs)
- Disponibilidade em vários sistemas operacionais e arquiteturas

Algumas Características:

- Sintaxe elegante e simples, facilitando a leitura e entendimento do código
- Velocidade de execução em um ambiente *interpretado* (há uma *máquina virtual* como Python, Java e alguns Prologs)
- Disponibilidade em vários sistemas operacionais e arquiteturas
- Análogo a Python, podem ser feitas *queries* ou *consultas* ao terminal de Picat.

Algumas Características:

- Sintaxe elegante e simples, facilitando a leitura e entendimento do código
- Velocidade de execução em um ambiente *interpretado* (há uma *máquina virtual* como Python, Java e alguns Prologs)
- Disponibilidade em vários sistemas operacionais e arquiteturas
- Análogo a Python, podem ser feitas *queries* ou *consultas* ao terminal de Picat.
- Há várias bibliotecas da própria linguagem, e diversas ferramentas externas permitindo o incremento do poder do Picat.

Acrônimo de **P.I.C.A.T.**

- **P**: *Pattern-matching*: Utiliza o conceito de *casamento de padrões* entre objetos, bem como os conceitos da *unificação* da LPO
- **I**: *Intuitive*: Oferece estruturas de decisão, atribuição e laços de repetição, etc. Análogo há outras linguagens de programação mais populares
- **C**: *Constraints*: Suporta a programação por restrições (PR) para problemas combinatórios
- **A**: *Actors*: Suporte as chamadas a eventos, chamada via *atores*
- **T**: *Tabling*: Implementa a técnica de *memoization* com soluções imediatas para problemas de Programação Dinâmica (PD).

Usando Picat

- Os seus arquivos fontes utilizam a extensão **.pi**. Exemplo: `programa.pi`
- Há dois modos principais de utilização do Picat:
  - Modo interativo, onde seu código é digitado e compilado diretamente na linha de comando;
  - *Modo console* onde o console só é utilizado para compilar seus programas.

Usando Picat

- Os seus arquivos fontes utilizam a extensão **.pi**. Exemplo: `programa.pi`
- Há dois modos principais de utilização do Picat:
  - Modo interativo, onde seu código é digitado e compilado diretamente na linha de comando;
  - *Modo console* onde o console só é utilizado para compilar seus programas.
- Códigos executáveis 100% **stand-alone**: ainda não!
- Neste quesito, estamos em igualdade com Java, Prolog e Python

Exemplo – *Alô Mundo!*

Acompanhar as explicações do código de:
https://github.com/claudiosa/CCS/blob/master/picat/alo_mundo.pi

```
main => msg_01  ,
        msg_02 .

msg_01 =>  printf("  ALO MUNDO!!! ").
msg_02 =>  printf("\n  FIM \n").
```

Execução na Console Linux ou Windows

```
$ picat alo_mundo.pi
  ALO MUNDO!!!
  FIM
$
```

Execução na Console Linux ou Windows

```
$ picat alo_mundo.pi
  ALO MUNDO!!!
  FIM
$
```

Análogo ao desenvolvimento com Python!

Execução no Ambiente do Interpretador

```
$ picat
Picat 2.0, (C) picat-lang.org, 2013-2016.
Type 'help' for help.
Picat> cl(álo_mundo.pi').
Compiling:: alo_mundo.pi
alo_mundo.pi compiled in 0 milliseconds
loading...

yes

Picat> main
  ALO MUNDO!!!
  FIM

yes

Picat> msg_02

  FIM

yes
```

Ambiente do Interpretador – Uso do `getline`

- Inicialmente, aqui o código foi carregado com o comando '`cl`' (digite `help` na console), o qual **compila** o seu código e **carrega** em um código intermediário pronto para ser executado e testado

Ambiente do Interpretador – Uso do `getline`

- Inicialmente, aqui o código foi carregado com o comando '`cl`' (digite `help` na console), o qual **compila** o seu código e **carrega** em um código intermediário pronto para ser executado e testado
- Neste ambiente interpretado há comandos básicos de teclado (mouse não funciona aqui) do programa `getline` do Linux. Os mais importantes são:

Ambiente do Interpretador – Uso do `getline`

- Inicialmente, aqui o código foi carregado com o comando '`cl`' (digite `help` na console), o qual **compila** o seu código e **carrega** em um código intermediário pronto para ser executado e testado
- Neste ambiente interpretado há comandos básicos de teclado (mouse não funciona aqui) do programa `getline` do Linux. Os mais importantes são:
  - **Crtl-a**: move o cursor para o início da linha
  - **Crtl-e**: move o cursor para o final da linha (*end*)
  - **Crtl-f**: move o cursor de uma posição a frente (forward)
  - **Crtl-b**: move o cursor de uma posição para trás (backward)
  - **Crtl-d**: exclui o carácter sob o cursor (a 2a. vez – sai do ambiente)
  - **Crtl-u**: exclui a linha inteira
  - As flechas ... repetem os últimos comandos

Ambiente do Interpretador – Uso

- Use um editor externo de sua preferência. Por exemplo: `geany` com plugin do Picat

Ambiente do Interpretador – Uso

- Use um editor externo de sua preferência. Por exemplo: `geany` com plugin do Picat
- Mantenha duas janelas de terminais abertas
  - Uma para o ambiente interpretado
  - Outra para usá-lo diretamente: `$console$ picat seu_programa.pi`

Ambiente do Interpretador – Uso

- Use um editor externo de sua preferência. Por exemplo: `geany` com plugin do Picat
- Mantenha duas janelas de terminais abertas
  - Uma para o ambiente interpretado
  - Outra para usá-lo diretamente: `$console$ picat seu_programa.pi`
- Os dois modos são importantes de se trabalhar simultaneamente

Ambiente do Interpretador – Uso

- Use um editor externo de sua preferência. Por exemplo: `geany` com plugin do Picat
- Mantenha duas janelas de terminais abertas
  - Uma para o ambiente interpretado
  - Outra para usá-lo diretamente: `$console$ picat seu_programa.pi`
- Os dois modos são importantes de se trabalhar simultaneamente
- Em dúvidas, digite: `Picat> help .`

Reflexões

Reflexões

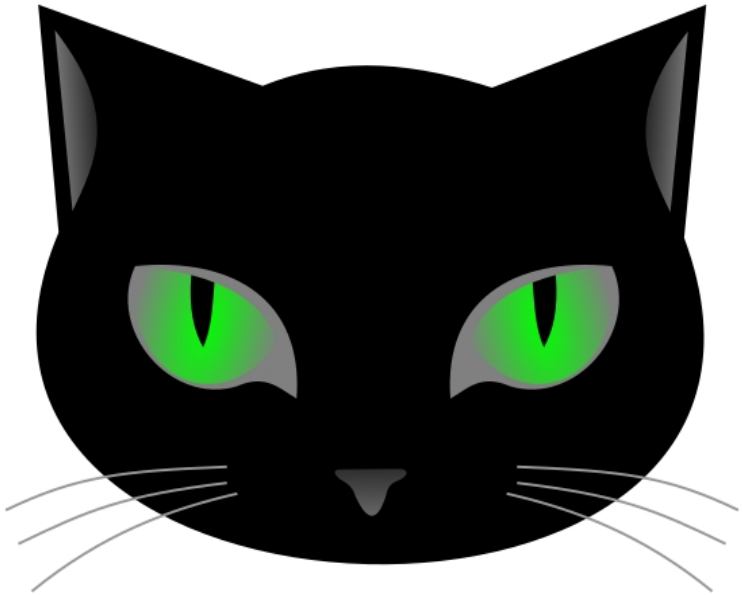
- Para próxima seção esteja com o Picat instalado em seu computador para um melhor aproveitamento.

Reflexões

- Para próxima seção esteja com o Picat instalado em seu computador para um melhor aproveitamento.
- Em códigos fontes: o símbolo '%' no início de linha, comenta a linha corrente

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

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!

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.

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 ...

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.

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)

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.

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.

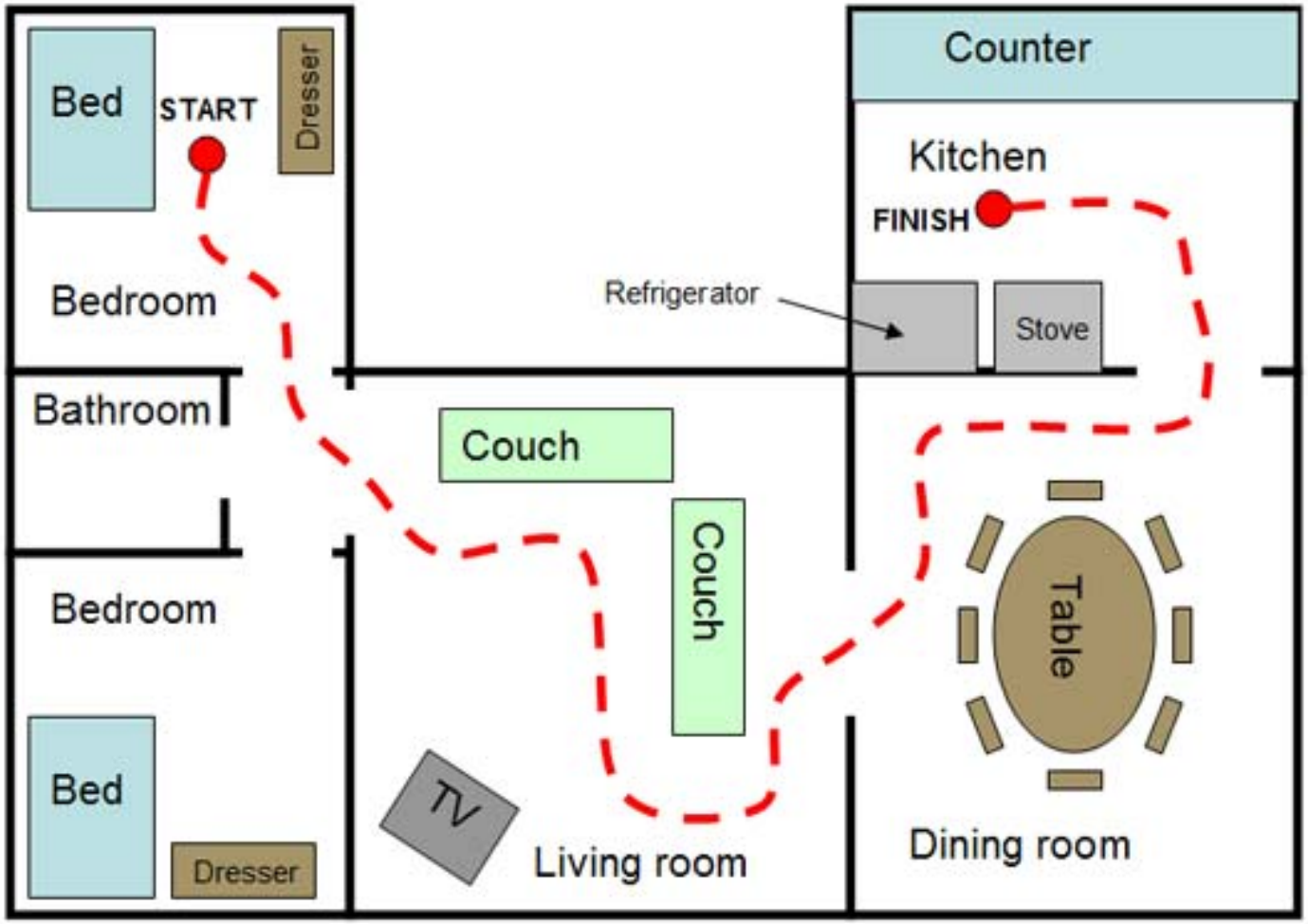


Figura 2: *A fome no meio da noite!*

-

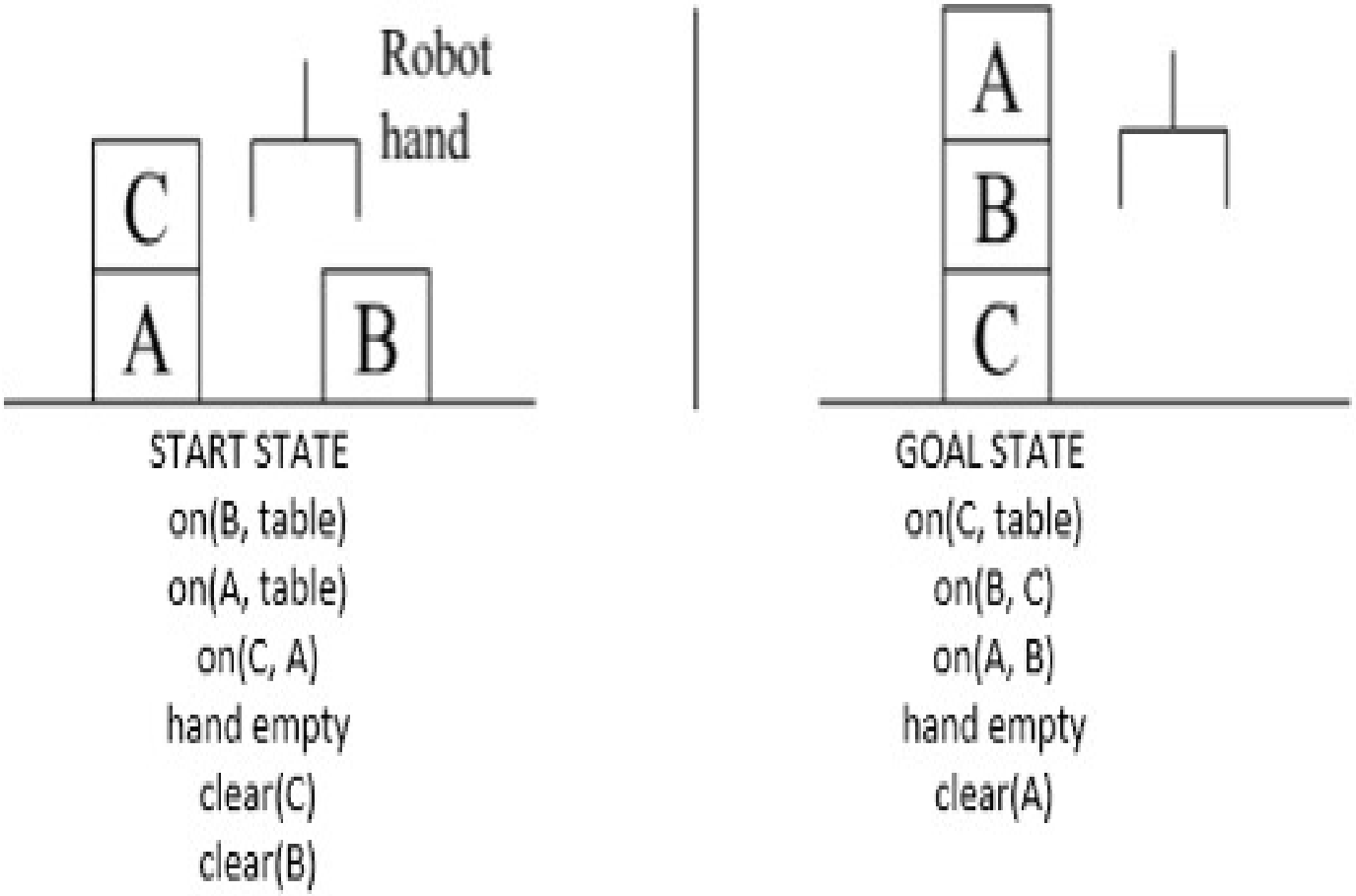


Figura 3: *O mundo dos blocos*

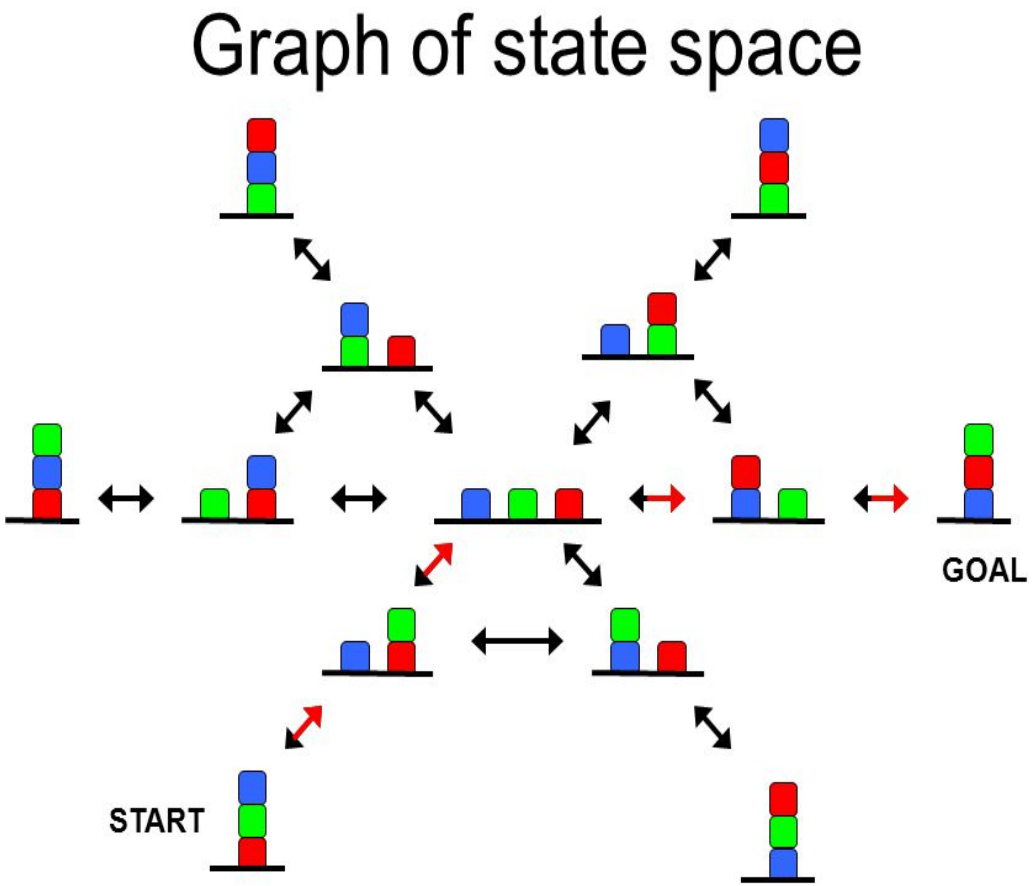


Figura 4: O espaço de estados do *mundo dos blocos* $\times$ ações

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

$$S_1 < S_2 < \ldots < S_n$$

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

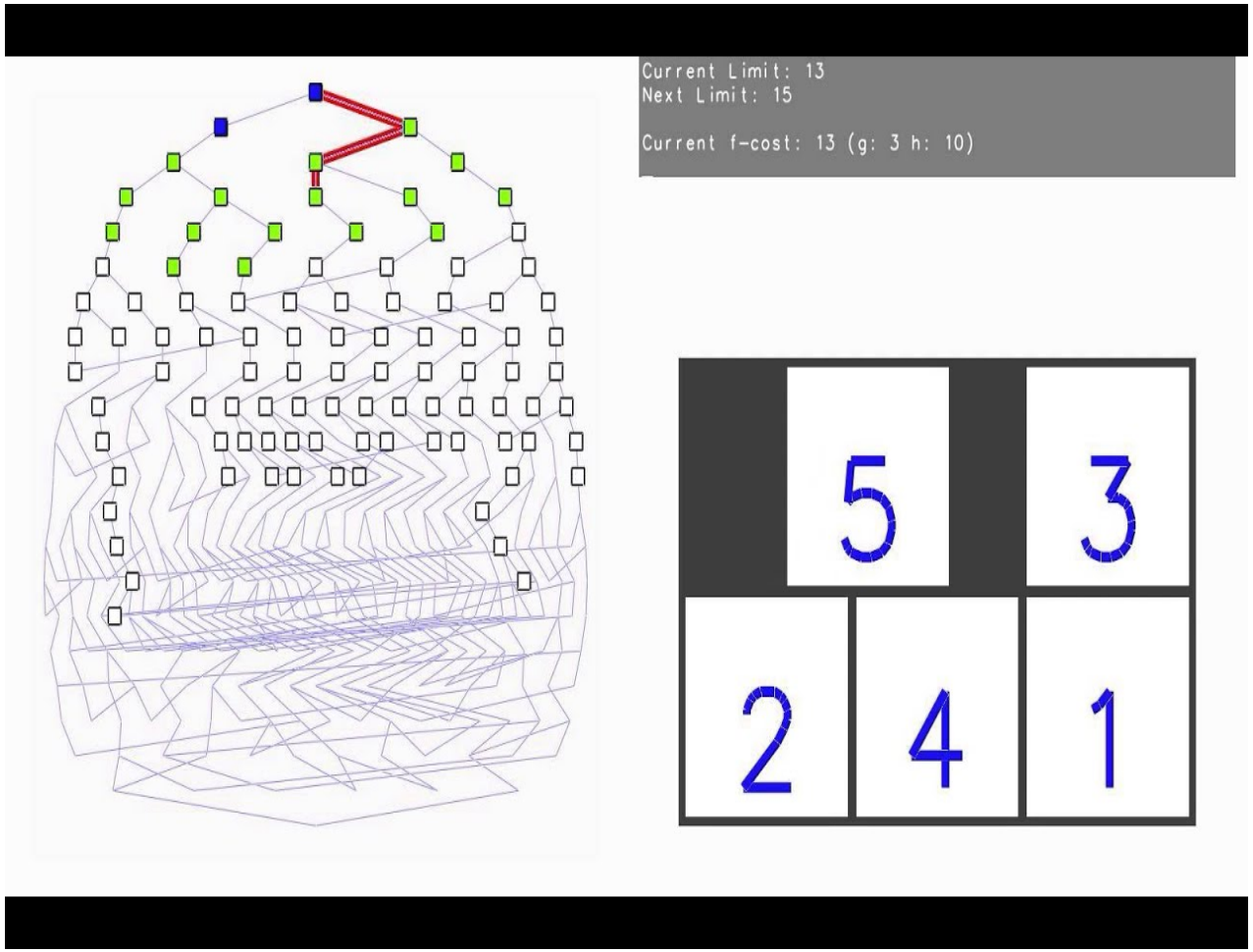


Figura 6: Sim, *simplificado* mas não muito!

```
/*
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)`

Reflexões

- 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

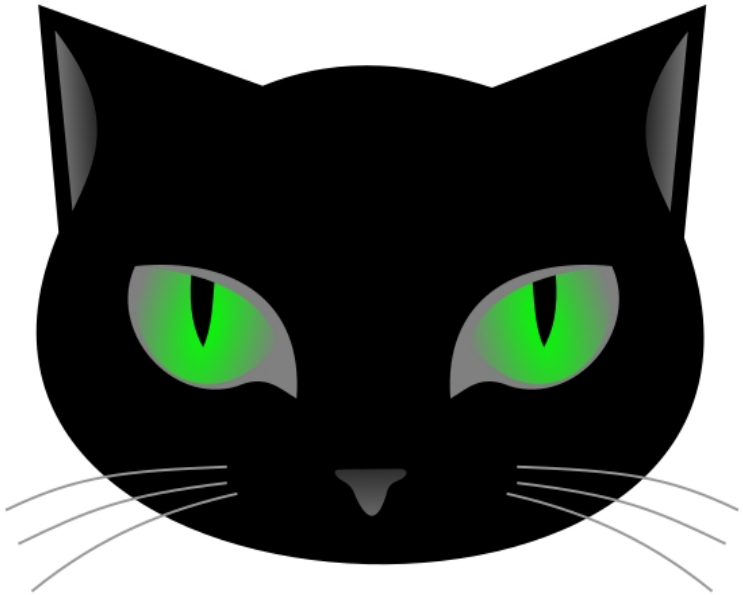
- Planejamento resolve uma classe ampla de problemas Havendo necessidade de **descobrir sequências ações** $\Leftrightarrow$ Planejamento
- Em geral, estes problemas são importantes na indústria
- Os modelos escritos em PDDL (*Planning Domain Definition Language*) facilmente portáveis para Picat
- Sob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

Conclusões

- O que foi visto
- O que tem a ser feito
- Oportunidades



Conclusões

- Picat é jovem $\approx$ 2013

Conclusões

- Picat é jovem $\approx$ 2013
- Uma evolução ao Prolog após seus mais de 40 anos de existência e sucesso!

Conclusões

- Picat é jovem $\approx$ 2013
- Uma evolução ao Prolog após seus mais de 40 anos de existência e sucesso!
- Sua sintaxe é moderna e intuitiva

Conclusões

- Picat é jovem $\approx$ 2013
- Uma evolução ao Prolog após seus mais de 40 anos de existência e sucesso!
- Sua sintaxe é moderna e intuitiva
- Código aberto, multi-plataforma e repleta de possibilidades (sistemas embarcados, combinatória, SO, escalonamentos, planejamento etc) $\Rightarrow$ NP's completos em geral

Conclusões

- Picat é jovem $\approx$ 2013
- Uma evolução ao Prolog após seus mais de 40 anos de existência e sucesso!
- Sua sintaxe é moderna e intuitiva
- Código aberto, multi-plataforma e repleta de possibilidades (sistemas embarcados, combinatória, SO, escalonamentos, planejamento etc) $\Rightarrow$ NP's completos em geral
- Muitas bibliotecas específicas prontas: CP (PR), SAT, Planner, etc

Conclusões

- Picat é jovem $\approx$ 2013
- Uma evolução ao Prolog após seus mais de 40 anos de existência e sucesso!
- Sua sintaxe é moderna e intuitiva
- Código aberto, multi-plataforma e repleta de possibilidades (sistemas embarcados, combinatória, SO, escalonamentos, planejamento etc) $\Rightarrow$ NP's completos em geral
- Muitas bibliotecas específicas prontas: CP (PR), SAT, Planner, etc
- A sintaxe de PR exige um pouco mais do programador

Conclusões

- Picat é jovem $\approx$ 2013
- Uma evolução ao Prolog após seus mais de 40 anos de existência e sucesso!
- Sua sintaxe é moderna e intuitiva
- Código aberto, multi-plataforma e repleta de possibilidades (sistemas embarcados, combinatória, SO, escalonamentos, planejamento etc) $\Rightarrow$ NP's completos em geral
- Muitas bibliotecas específicas prontas: CP (PR), SAT, Planner, etc
- A sintaxe de PR exige um pouco mais do programador
- Dúvidas: o guia do usuário, livro do Hakan e o Fórum de discussão do Picat

Ficou Faltando:

- Uso do `debug` e `trace` (cansativo – uma oportunidade) $\Rightarrow$ **Pendência resolvida!**

Ficou Faltando:

- Uso do `debug` e `trace` (cansativo – uma oportunidade) $\Rightarrow$ **Pendência resolvida!**
- Explorar uso dos *solvers* de PO (fácil)

Ficou Faltando:

- Uso do `debug` e `trace` (cansativo – uma oportunidade) $\Rightarrow$ **Pendência resolvida!**
- Explorar uso dos *solvers* de PO (fácil)
- Explorar a criação e uso de módulos (mais fácil ainda)

Dicas de Programação:

- Use o interpretador e o compilador concomitamente. O interpretador sempre acusa *warnings* etc. O modo compilado na console não apresenta alguns *warnings*. Efeito *cascata* de erro ...

Dicas de Programação:

- Use o interpretador e o compilador concomitamente. O interpretador sempre acusa *warnings* etc. O modo compilado na console não apresenta alguns *warnings*. Efeito *cascata* de erro ...
- No modo interpretado, cada linha de código pode ser testada isoladamente, assim, o efeito global desta é restrita. Qualquer erro ou falha é rapidamente detectada.

Dicas de Programação:

- Use o interpretador e o compilador concomitamente. O interpretador sempre acusa *warnings* etc. O modo compilado na console não apresenta alguns *warnings*. Efeito *cascata* de erro ...
- No modo interpretado, cada linha de código pode ser testada isoladamente, assim, o efeito global desta é restrita. Qualquer erro ou falha é rapidamente detectada.
- Consulte o manual do usuário *on-line* em *html*, mantido pelo Alexandre Gonçalves – UFSC
  `http://retina.inf.ufsc.br/picat_guide`

Dicas de Programação:

- Use o interpretador e o compilador concomitamente. O interpretador sempre acusa *warnings* etc. O modo compilado na console não apresenta alguns *warnings*. Efeito *cascata* de erro ...
- No modo interpretado, cada linha de código pode ser testada isoladamente, assim, o efeito global desta é restrita. Qualquer erro ou falha é rapidamente detectada.
- Consulte o manual do usuário *on-line* em *html*, mantido pelo Alexandre Gonçalves – UFSC
  `http://retina.inf.ufsc.br/picat_guide`
- Consulte o site do Picat e dos grandes mestres Hakan, Neng-Fa, Roman Barták, Sergii Dymchenko, etc

Dicas de Programação:

- Use o interpretador e o compilador concomitamente. O interpretador sempre acusa *warnings* etc. O modo compilado na console não apresenta alguns *warnings*. Efeito *cascata* de erro ...
- No modo interpretado, cada linha de código pode ser testada isoladamente, assim, o efeito global desta é restrita. Qualquer erro ou falha é rapidamente detectada.
- Consulte o manual do usuário *on-line* em *html*, mantido pelo Alexandre Gonçalves – UFSC
  `http://retina.inf.ufsc.br/picat_guide`
- Consulte o site do Picat e dos grandes mestres Hakan, Neng-Fa, Roman Barták, Sergii Dymchenko, etc
- Inscreva-se no fórum e consulte o Guia do Usuário (tudo em inglês)

Agradecimentos

- Muito obrigado a voce!

Agradecimentos

- Muito obrigado a voce!
- Algumas pessoas que deram opiniões neste material e me incentivaram a terminá-lo

Agradecimentos

- Muito obrigado a voce!
- Algumas pessoas que deram opiniões neste material e me incentivaram a terminá-lo
- Em especial a minha esposa: Maria Luiza

Agradecimentos

- Muito obrigado a voce!
- Algumas pessoas que deram opiniões neste material e me incentivaram a terminá-lo
- Em especial a minha esposa: Maria Luiza
- Claudio Cesar de Sá

- Contacto:
  - ✉ `claudio.sa@udesc.br`
  - ✉ `claudio@colmeia.udesc.br`

... espero que tenham gostado e obrigado!