PICAT: Uma Linguagem de Programação Multiparadigma

Miguel Alfredo Nunes, Jeferson L. R. Souza, Claudio Cesar de Sá

miguel.nunes@edu.udesc.br

jeferson.souza@udesc.br

claudio.sa@udesc.br

Departamento de Ciência da Computação
Centro de Ciências e Tecnológias
Universidade do Estado de Santa Catarina

3 de abril de 2019

Contribuições

- Alexandre Gonçalves;
- João Herique Faes Battisti;
- Paulo Victor de Aguiar;
- Rogério Eduardo da Silva;
- Hakan Kjellerstrand `http://www.hakank.org/picat/`
- Neng-Fa Zhou `http://www.picat-lang.org/`
- Outros anônimos que auxiliaram na produção deste documento;

Apresentação ao Curso de PICAT – I

- O que é o PICAT?

Apresentação ao Curso de PICAT – I

- O que é o PICAT?
  - Uma linguagem de programação de propósitos gerais
  - Uma evolução do PROLOG (consagrada linguagem dos primórdios da IA)
  - Tem elementos das linguagens Python, Prolog e Haskell
- Uso e finalidades do PICAT:

Apresentação ao Curso de PICAT – I

- O que é o PICAT?
  - Uma linguagem de programação de propósitos gerais
  - Uma evolução do PROLOG (consagrada linguagem dos primórdios da IA)
  - Tem elementos das linguagens Python, Prolog e Haskell
- Uso e finalidades do PICAT:
  - Uso de programas gerais; de simples à complexos (uma reflexão)
  - Provê suporte há vários solvers na área de Pesquisa Operacional
  - Área: IA, programação por restrições, programação inteira, planejamento, combinatória, etc

Apresentação ao Curso de PICAT – II

- Este curso é dirigido a voce?

Apresentação ao Curso de PICAT – II

- Este curso é dirigido a voce?
- Requisitos:

Apresentação ao Curso de PICAT – II

- Este curso é dirigido a voce?
- Requisitos:
  - Conhecimento: noções de lógica matemática (proposional e primeira-ordem), matemática elementar, e alguma outra linguagem de programação
  -
  - Dedicação: depende de você

Apresentação ao Curso de PICAT – II

- Este curso é dirigido a voce?
- Requisitos:
  - Conhecimento: noções de lógica matemática (proposional e primeira-ordem), matemática elementar, e alguma outra linguagem de programação
  -
  - Dedicação: depende de você
- Motivação:

Apresentação ao Curso de PICAT – II

- Este curso é dirigido a voce?
- Requisitos:
  - Conhecimento: noções de lógica matemática (proposional e primeira-ordem), matemática elementar, e alguma outra linguagem de programação
  -
  - Dedicação: depende de você
- Motivação:
  - Dependendo de sua dedicação, ao final voce vai estar apto a resolver problemas computacionais de simples à difíceis
  - Difícil: muitas linhas de código e muito conhecimento de algoritmos seriam necessários
  - Com Picat, há sofisticados esquemas prontos para se construir programas.

Apresentação ao Curso de PICAT – III

- Requisitos computacionais:

Apresentação ao Curso de PICAT – III

- Requisitos computacionais: Um computador qualquer (arquitetura 16, 32 ou 64 bits), com Linux, Mac ou Windows, que tenha um compilador C instalado completo, preferencialmente.
- Comunidade e ações: `http://picat-lang.org`

Apresentação ao Curso de PICAT – III

- Requisitos computacionais: Um computador qualquer (arquitetura 16, 32 ou 64 bits), com Linux, Mac ou Windows, que tenha um compilador C instalado completo, preferencialmente.
- Comunidade e ações: `http://picat-lang.org`
- Códigos e este material, sempre atualizados em:

Apresentação ao Curso de PICAT – III

- Requisitos computacionais: Um computador qualquer (arquitetura 16, 32 ou 64 bits), com Linux, Mac ou Windows, que tenha um compilador C instalado completo, preferencialmente.
- Comunidade e ações: `http://picat-lang.org`
- Códigos e este material, sempre atualizados em:
  - Este PDF e seu texto original: `http://github.com/claudiosa/Slides_Picat`
  - Os códigos de programas: `http://github.com/claudiosa/CCS/picat`
- Tópicos que serão cobertos no curso:

Sumário I

Listas

- Requisito: conceitos de recursividade e functores dominados!
- Os conceitos são os próximos os das LPs convencionais
- Essencialmente vamos computar sob uma árvore binária (cada nó tem duas ramificações)
- Lembrando que uma estrutura binária de árvore tem uma equivalência com uma árvore n-ária (ver livro de Estrutura de Dados)
- Logo, listas são estruturas flexíveis e poderosas!

Ilustrando uma Lista em Formato Binário

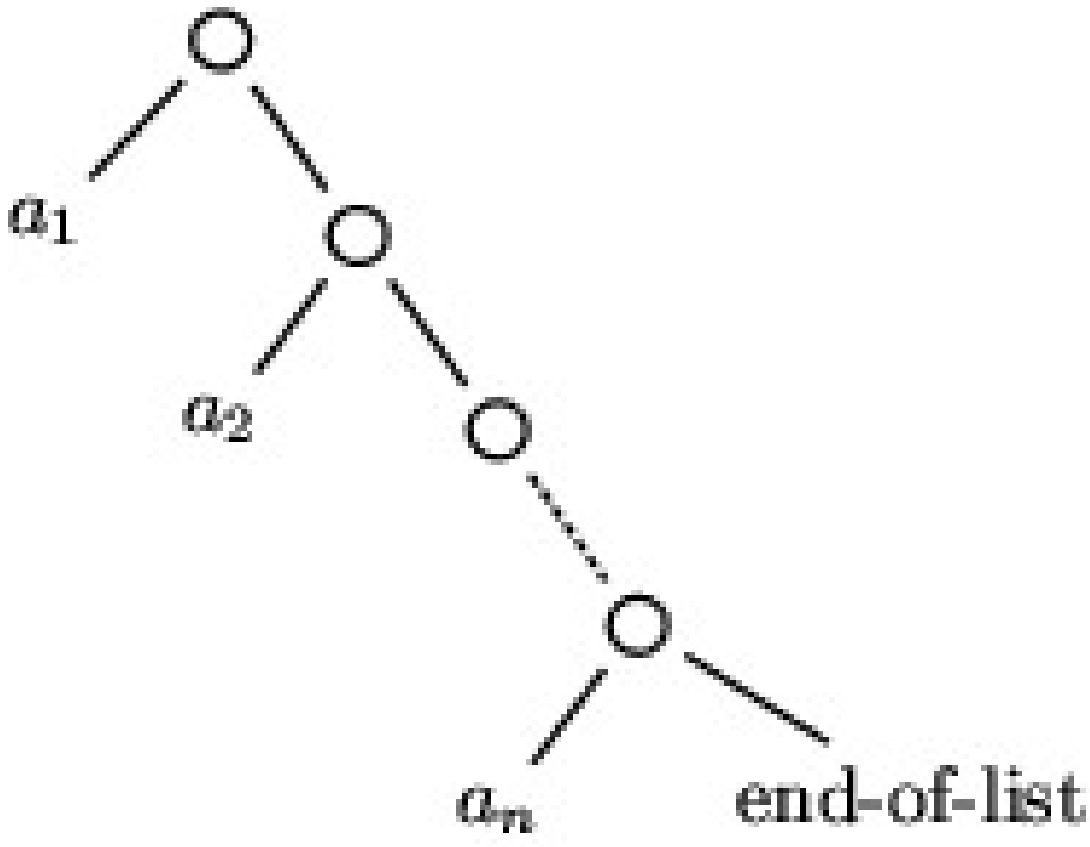

Figura 1: Uma estrutura Lista – Homogênea

Ilustrando a Listas

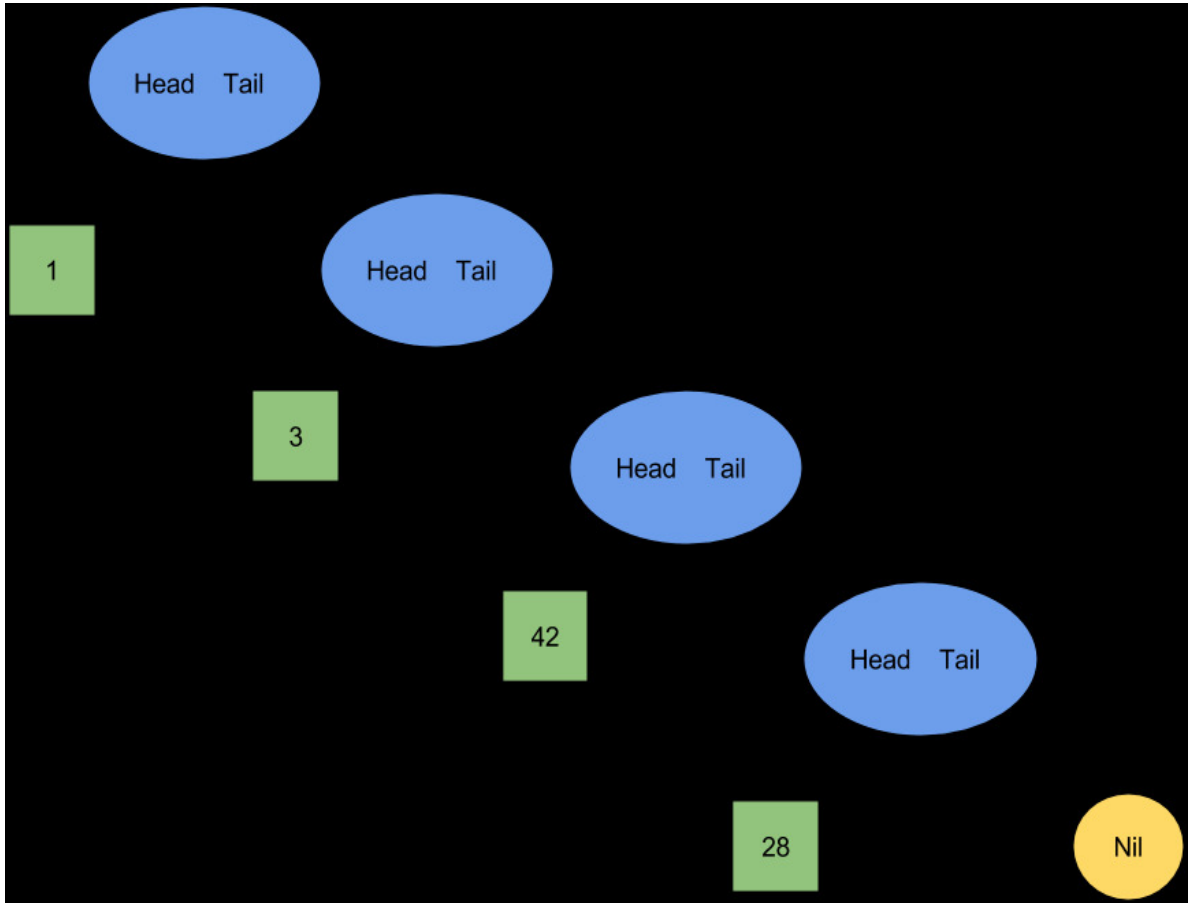

Figura 2: Listas são inerentemente **recursivas**!

Exemplificando as Listas

lista:	[a,b,c,d]
cabeça:	a
cauda:	[b,c,d]
lista:	[[a,b],c,[d,e,f],g]
cabeça:	[a,b]
cauda:	[c,[d,e,f],g]
lista:	[[A11,A12],[A21,A22]]
cabeça:	[A11,A12]
cauda:	[[A21,A22]]

Sintaxe das Listas I

Definições iniciais (e recursivas)

- Uma lista é uma sequência de objetos;
- Uma lista é uma estrutura de dados que representa uma coleção de objetos homogêneos;
- Uma lista apresenta uma hierarquia natural, internamente, em *cabeça* de lista e sub-lista, até o fim da lista.

Sintaxe das Listas II

Notação:

- O símbolo "[" é usado para descrever o início de uma lista, e "]" para o final da mesma;
- Exemplo: seja a lista [a, b, c, d], logo um predicado cujo argumento seja algumas letras, tem-se uma lista do tipo:
  - letras([a, b, c, d])
  - Onde 'a' é o *cabeça* (primeiro elemento) da lista
  - e [b, c, d]é uma *sub-lista* que é uma lista!
- Os elementos de uma lista são lidos da esquerda para direita;
- A "*sub-lista*" [b, c, d]é conhecida como *resto* ou "*cauda*" da lista;
- Esta sub-lista é uma lista, e toda definição segue-se recursivamente.

Sintaxe das Listas III

Operador "|":

- "*Como vamos distinguir de onde se encontra a cabeça da cauda da lista?*"
- Com as listas novos símbolos foram introduzidos, isto é, além dos delimitadores [...], há um novo operador que **separa** ou **define** quem é a elemento cabeça da lista e cauda.
- Este operador é conhecido como "*pipe*" (ou *barra vertical*), simbolizado por "|", que separa o lado esquerdo da direita da lista.
- Esta separação é necessário para se realizar os *casamentos de padrões* nas linguagens lógicas.

Sintaxe das Listas IV

Exemplos de "*casamentos*":

```
[ a, b, c, d ] = X
[ X | b, c, d ]  =  [ a, b, c, d ]
[ a | b, c, d ]  =  [ a, b, c, d ]
[ a , b | c, d ]  =  [ a, b, c, d ]
[ a , b , c | d ]  =  [ a, b, c, d ]
[ a , b , c , d | [] ]  =  [ a, b, c, d ]
[] = X
[ [ a | b , c , d] ] = [ [ a , b , c , d ] ]
[  a | b , c , [ d ] ] = [  a , b , c , [ d ] ]
[  _ | b , c , [d ] ] = [  a , b , c , [ d ] ]
[  a | Y ] = [  a , b , c ,  d ]
[  a | _ ] = [  a , b , c ,  d ]
[  a , b | c , d ] = [  X , Y | Z ]
```

Sintaxe das Listas V

Contra-exemplos de "*casamentos*":

```
[ a , b | [c, d] ]  !=  [ a, b, c, d ]
[ [ a , b , c , d] ]  !=  [ a, b, c, d ]
[  a , b , [ c ] , d, e ]  !=  [ a, b, c, d, e ]
[ [ [ a ] | b , c , d] ] != [ [ a , b , c , d] ]
```

Sintaxe das Listas VI

- Estes casamentos de objetos de uma lista são também conhecidos por "*matching*"
- Devido ao fato de listas modelarem qualquer estrutura de dados, invariavelmente, seu uso é extensivo há problemas em geral (dos simples a complexos)

Exemplo: encontrar o comprimento de uma lista I

- O comprimento de uma lista é o comprimento de sua **sub-lista**, mais **um**
- O comprimento de uma lista vazia ([]) é zero.

Em Picat, este enunciado é escrito por:

```
REFAZER

#1 compto([ ], 0).
#2 compto([X | T], N) => compto(T, N1), N is N1+1.

? - compto([a, b, c, d], X).
X = 4
```

Exemplo: encontrar o comprimento de uma lista II

Um "*mapa de memória*" é dado por:

	Regra	X	T	N1	N = N+1
compto([a,b,c,d],N)	#2	a	[b,c,d]	3 →	3+1=4
compto([b,c,d],N)	#2	b	[c,d]	2 →	↖ 2+1
compto([c,d],N)	#2	c	[d]	1 →	↖ 1+1
compto([d],N)	#2	d	[]	0 →	↖ 0+1
compto([],N)	#1	–	–	–	↖ 0

Exemplo: verificar a pertinência de um objeto na lista I

Verifica se um dado objeto pertence há uma lista

```
REFAZER
member( H, [ H |  _ ] ).
member( H, [ _  | T ] ) :- member(H, T).
```

O interessante é observar a versatilidade deste predicado em várias situações:

```
?- member(3, [4,5,3]).

Yes
?- member(X, [4,5,3]).
X = 4 ;
X = 5 ;
X = 3 ;
```

Exemplo: verificar a pertinência de um objeto na lista II

```
No
?- member(3, X).
X = [3|_G231]
Yes
?- member(3, X).

X = [3|_G231] ;
X = [_G230, 3|_G234] ;
X = [_G230, _G233, 3|_G237]
.........................
```

Exemplo: adicionar um elemento em uma lista I

Um objeto é adicionado a lista sem repetição caso este já esteja contido na lista:

REFAZER

```
add_to_set(X, [ ], [X]).
add_to_set(X, Y, Y) :- member(X, Y).
add_to_set(X, Y, [X | Y]).
```

Concluindo Listas I

- Há muitos predicados e funções prontas sobre listas no módulo lists.pi (conferir)
- Se aprende sobre listas, fazendo muitos métodos
- A recursividade em sua modelagem, define a metodologia de se *programar em lógica*
- Exercitar-se
- Usar as listas em problemas complexos

Resumindo

- Picat é jovem (nascida em 2013);
- Uma evolução ao Prolog após seus mais de 40 anos de existência e sucesso!
- Sua sintaxe é moderna;
- Código aberto, multi-plataforma, e repleta de possibilidades;
- Uso para fins diversos;
- Muitas bibliotecas específicas prontas: CP, SAT, Planner, etc;
-
- .