

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 Tecnologias
Universidade do Estado de Santa Catarina

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;

Recursão

- ▶ A *recursão* é um importante conceito da matemática e presente em muitas linguagens de programação. Exemplo: LISP, Haskell, etc

Recursão

- ▶ A *recursão* é um importante conceito da matemática e presente em muitas linguagens de programação. Exemplo: LISP, Haskell, etc
- ▶ Permite expressar conceitos complexos em uma sintaxe abstrata, mas simples de ler.

Recursão

- ▶ A *recursão* é um importante conceito da matemática e presente em muitas linguagens de programação. Exemplo: LISP, Haskell, etc
- ▶ Permite expressar conceitos complexos em uma sintaxe abstrata, mas simples de ler.
- ▶ Uma regra é dita recursiva quando ela faz auto-referência.

Recursão

- ▶ A *recursão* é um importante conceito da matemática e presente em muitas linguagens de programação. Exemplo: LISP, Haskell, etc
- ▶ Permite expressar conceitos complexos em uma sintaxe abstrata, mas simples de ler.
- ▶ Uma regra é dita recursiva quando ela faz auto-referência.
- ▶ Em Picat, a recursão pode ser usada sob uma notação em *lógica* ou *funcional*

Recursão

- ▶ A *recursão* é um importante conceito da matemática e presente em muitas linguagens de programação. Exemplo: LISP, Haskell, etc
- ▶ Permite expressar conceitos complexos em uma sintaxe abstrata, mas simples de ler.
- ▶ Uma regra é dita recursiva quando ela faz auto-referência.
- ▶ Em Picat, a recursão pode ser usada sob uma notação em *lógica* ou *funcional*
- ▶ A funcional apresenta muita clareza ao código!

Conceitos de Recursividade via Exemplos – I

Somatório dos N naturais

O somatório dos n primeiros números naturais é recursivamente definido como a soma de todos $n-1$ números, mais o termo n . Ou seja:

$$S(n) = \begin{cases} 1 & \text{para } n = 1 \\ S(n-1) + n & \text{para } n \geq 2 \text{ e } n \in \mathbb{N} \end{cases}$$

Ou seja:

$$S(n) = \underbrace{1 + 2 + 3 + \dots + (n-1)}_{S(n-1)} + n$$

Conceitos de Recursividade via Exemplos – II

Fatorial

O Fatorial de um número n é definido recursivamente pela multiplicação do fatorial do termo $n - 1$ por n . O fatorial só pode ser calculado para números positivos. Adicionalmente, o fatorial de 0 é igual a 1 por definição.

$$Fat(n) = \begin{cases} 1 & \text{para } n = 0 \\ Fat(n - 1) \cdot n & \text{para } n \geq 1 \text{ e } n \in \mathbb{N} \end{cases}$$

Portanto:

$$Fat(n) = \underbrace{1 * 2 * 3 * \dots * (n - 1)}_{Fat(n - 1)} \cdot n$$

Conceitos de Recursividade via Exemplos – III

Sequência Fibonacci

A sequência Fibonacci é uma sequência de números calculada a partir da soma dos dois últimos números anteriores desta. Ou seja o n – *esimo* termo da Sequência Fibonacci é definido como a soma dos termos $n - 1$ e $n - 2$. Como fato ou definição: os dois primeiros termos, $n = 0$ e $n = 1$, são respectivamente, 0 e 1.

$$Fib(n) = \begin{cases} 0 & \text{para } n = 0 \\ 1 & \text{para } n = 1 \\ Fib(n - 1) + Fib(n - 2) & \text{para } n \geq 1 \text{ e } n \in \mathbb{N} \end{cases}$$

Conceitos de Recursividade via Exemplos – IV

- Podemos perceber algo em comum entre estas três regras, todas tem uma ou mais condições que sempre tem o mesmo valor de retorno, ou seja, todas tem uma *regra de aterramento*.

Conceitos de Recursividade via Exemplos – IV

- ▶ Podemos perceber algo em comum entre estas três regras, todas tem uma ou mais condições que sempre tem o mesmo valor de retorno, ou seja, todas tem uma *regra de aterramento*.
- ▶ Uma condição de *aterramento* é uma condição onde a chamada recursiva da regra acaba (para ou termina).

Conceitos de Recursividade via Exemplos – IV

- ▶ Podemos perceber algo em comum entre estas três regras, todas tem uma ou mais condições que sempre tem o mesmo valor de retorno, ou seja, todas tem uma *regra de aterramento*.
- ▶ Uma condição de *aterramento* é uma condição onde a chamada recursiva da regra acaba (para ou termina).
- ▶ Caso uma regra não tenha uma *regra de aterramento*, poderá ocorrer uma recursão infinita deste regra, ou seja, são feitas infinitas chamadas recursivas da regra.

Exemplos

Numa visão funcional, estas regras matemáticas podem ser transcritas em Picat como:

```
1 fatorial(0) = 1.
2 fatorial(1) = 1.
3 fatorial(n) = n * fatorial(n-1).
4
```

```
1 fibonacci(0) = 0.
2 fibonacci(1) = 1.
3 fibonacci(n) = fibonacci(n-1) + fibonacci(n-2).
4
```

```
1 somatorio(0) = 0.
2 somatorio(1) = 1.
3 somatorio(n) = n + somatorio(n-1).
```

Recursão Infinita

- ▶ Caso a definição do fatorial fosse modificada para:

$$Fat(n) = Fat(n - 1) * n, \quad \forall n \in \mathbb{N} \text{ ou } \forall n \geq 0$$

Recursão Infinita

- ▶ Caso a definição do fatorial fosse modificada para:

$$Fat(n) = Fat(n - 1) * n, \quad \forall n \in \mathbb{N} \text{ ou } \forall n \geq 0$$

- ▶ Teríamos um caso de *recursão infinita*, pois a regra Fatorial continuaria a ser chamada com $n < 0$
- ▶ Nesse caso haveria um erro, pois estaria tentando executar algo indefinido.

Exercício

Para os exemplos anteriores, reescreva-os as formulações sob uma visão *lógica* e *procedural*.

Backtracking

- ▶ O mecanismo de *backtracking* é bem conhecido por algumas linguagens de programação

Backtracking

- ▶ O mecanismo de *backtracking* é bem conhecido por algumas linguagens de programação
- ▶ Em Picat, o *backtracking* é controlável e é habilitado pelo símbolo `?=>` no escopo da regra.

Ilustrando o *Backtracking* – 01

Backtracking Implementation

Backtracking is a modified depth-first search of the solution-space tree. In the case of the maze the start location is the root of a tree, that branches at each point in the maze where there is a choice of direction.



Backtracking I

Basicamente o procedimento do *Backtracking* é definido por:

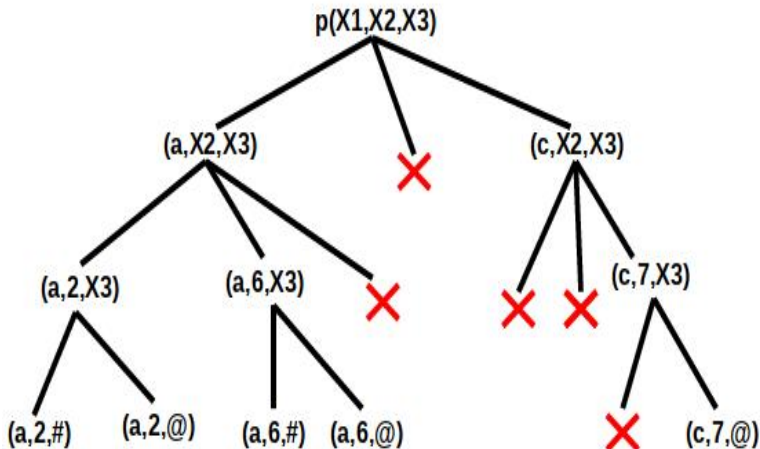
1. Inicia-se por um casamento de um predicado *backtrackable* p com um outro predicado p .
2. Segue-se a execução da regra p , executando a instância das variáveis da esquerda para direita. **Exemplo** (ilustrativo):
$$p(X_1, X_2, X_3, \dots, X_n) \text{ ?} \Rightarrow q_1(X_1), q_2(X_2), \dots, q_n(X_n).$$

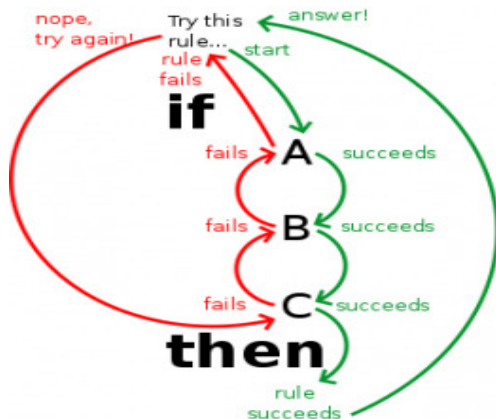
Backtracking II

3. Caso ocorra uma falha durante a execução da regra p , o compilador busca re-instanciar as variáveis do corpo de p que falharem. Esta tentativa segue uma ordem:
 $q1(X1) \rightarrow q2(X2) \rightarrow \dots \rightarrow qn(Xn)$, até a variável Xn
4. Caso Xn seja instanciada com sucesso, tem-se uma resposta consistente para p

Backtracking III

5. No caso de uma falha completa na regra corrente p , segue-se para uma próxima regra p ($p \dots ? \Rightarrow \dots$), a qual é avaliada com novas instâncias as suas variáveis.
6. Este processo é completo (exaustivo) e se repete até não for mais possível a reinstanciação de variáveis, ou ocorrer uma falha durante a execução.

Ilustrando o *Backtracking* – 02

Ilustrando o *Backtracking* – 03

Exemplos

Tomando como exemplo uma relação de parentesco, como a seguinte:

```
1 index(-,-) (+,-) (-,+)
2 antecedente(ana,maria).
3 antecedente(pedro,maria).
4 antecedente(maria,paula).
5 antecedente(paula,lucas).
6 antecedente(lucas,eduarda).
7
8 index(-)
9 mulher(ana).
10 mulher(maria).
11 mulher(paula).
12 mulher(eduarda).
13 homem(pedro).
14 homem(lucas).
15
16 mae(X,Y) ?=> antecedente(X,Y), mulher(X).
17 pai(X,Y) ?=> antecedente(X,Y), homem(X).
18 avos(X,Y) ?=> antecedente(X,Z), antecedente(Z,Y).
```

Exercícios

- ▶ Uma chamada do tipo $mae(maria, X)$, seria como perguntar ao compilador "Maria é mãe de quem ?".
- ▶ Nesse caso o compilador iria testar cada possível valor que pudesse ser unificado com X que pudesse satisfazer a regra $mae(maria, X)$.
- ▶ Ou seja, seria como se estivéssemos perguntando:
 - ▶ "Maria é mãe de Ana ?".
 - ▶ "Maria é mãe de Paula ?".
 - ▶ "Maria é mãe de Pedro ?".

Reflexões

- ▶ A recursão é o paradigma das linguagens declarativas como Haskell, Prolog, Picat, ... etc

Reflexões

- ▶ A recursão é o paradigma das linguagens declarativas como Haskell, Prolog, Picat, ... etc
- ▶ As regras recursivas são construídas com uma ou mais *regras aterradas*, que **sempre vem antes** das demais regras recursivas, as quais podem ou não terem o *backtracking* habilitados (\Rightarrow)

Reflexões

- ▶ A recursão é o paradigma das linguagens declarativas como Haskell, Prolog, Picat, ... etc
- ▶ As regras recursivas são construídas com uma ou mais *regras aterradas*, que **sempre vem antes** das demais regras recursivas, as quais podem ou não terem o *backtracking* habilitados (\Rightarrow)
- ▶ A avaliação destas regras **são sempre da esquerda para direita**, ocorrendo o *backtracking* em caso de falha ou de uma nova resposta

Reflexões

- ▶ A recursão é o paradigma das linguagens declarativas como Haskell, Prolog, Picat, ... etc
- ▶ As regras recursivas são construídas com uma ou mais *regras aterradas*, que **sempre vem antes** das demais regras recursivas, as quais podem ou não terem o *backtracking* habilitados ($?=>$)
- ▶ A avaliação destas regras **são sempre da esquerda para direita**, ocorrendo o *backtracking* em caso de falha ou de uma nova resposta
- ▶ As regras recursivas com *backtracking* habilitados ($?=>$), apenas para regras predicativas. As funções não admitem *backtracking*!

Reflexões

- ▶ A recursão é o paradigma das linguagens declarativas como Haskell, Prolog, Picat, ... etc
- ▶ As regras recursivas são construídas com uma ou mais *regras aterradas*, que **sempre vem antes** das demais regras recursivas, as quais podem ou não terem o *backtracking* habilitados ($?=>$)
- ▶ A avaliação destas regras **são sempre da esquerda para direita**, ocorrendo o *backtracking* em caso de falha ou de uma nova resposta
- ▶ As regras recursivas com *backtracking* habilitados ($?=>$), apenas para regras predicativas. As funções não admitem *backtracking*!
- ▶ A metodologia destas regras e sua construção, seguem esquemas mais avançados da programação declarativa

Reflexões

- ▶ A recursão é o paradigma das linguagens declarativas como Haskell, Prolog, Picat, ... etc
- ▶ As regras recursivas são construídas com uma ou mais *regras aterradas*, que **sempre vem antes** das demais regras recursivas, as quais podem ou não terem o *backtracking* habilitados ($?=>$)
- ▶ A avaliação destas regras **são sempre da esquerda para direita**, ocorrendo o *backtracking* em caso de falha ou de uma nova resposta
- ▶ As regras recursivas com *backtracking* habilitados ($?=>$), apenas para regras predicativas. As funções não admitem *backtracking*!
- ▶ A metodologia destas regras e sua construção, seguem esquemas mais avançados da programação declarativa
- ▶ O **main** é uma regra predicativa que pode conter ou não