PICAT: uma Linguagem Multiparadigma

Claudio Cesar de Sá, Rogério Eduardo da Silva, João Herique Faes Battisti, Paulo Victor de Aguiar

joaobattisti@gmail.com

pavaguiar@gmail.com

claudio.sa@udesc.br

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

Sumário

Histórico

- Criada em 2013 por Neng-Fa Zhou e Jonathan Fruhman.
- Utilizou o B-Prolog como base de implementação, e ambas utilizam a programação em lógica baseada em regras de predicativas>
- Picat 0.1 – Teve seu lançamento em Maio de 2013.
- Picat 1.0 – Foi lançada Abril de 2015.
- Sua atual versão é a 2.0 (9 de novembro de 2016)

Picat é Multiparadigma

- Imperativo – procedural
- Funcional
- Lógico

Linguagem Multiparadigma

Motivo de existencia dos paradigmas?

- Sintaxe $\Rightarrow$ elegância do código
- Velocidade de execução
- Portabilidade

Linguagem Picat

- Terminologia: segue as bases teóricas da linguagem Prolog.
- Na **l**ógica de **p**rimeira-**o**rdem (LPO) os objetos são chamados por **termos**.
- O destaque de Picat é a sua natureza declarativa, funcional, tipagem dinâmica, e sintaxe *açucarada*
- `PICAT` é um anacrônico onde cada letra representa uma característica de sua funcionalidade (operacionalidade).

Características: P-I-C-A-T

Pattern-matching:

- Utiliza o conceito de casamento padrão.
- Um predicado define uma relação entre objetos n-ários
- Uma função é um predicado especial que sempre retorna uma única resposta.
- Ambos são definidos com regras de Picat, e seus predicados e funções seguem as regras de *casamento-de-padrões*.

Características: P-I-C-A-T

Intuitive:

- O Picat oferece atribuições e laços de repetições para a programação dos dias de hoje.
- Uma variável atribuída imita variáveis lógicas, alterado seu valor seguindo o estado da computação.
- As atribuições são úteis para associar os termos, bem como utilizadas nas estruturas de laços repetitivos.

Características: P-I-C-A-T

Constraints:

- Picat suporta a programação por restrições.
- Dado um conjunto de variáveis, cada uma possui um domínio de valores possíveis e restrições para limitar os valores a serem atribuídos nas variáveis.
- O objetivo é atribuir os valores que satisfaçam todas as restrições.

Características: P-I-C-A-T

Actors: **REFAZER em breve ...**

- Atores são chamadas orientadas à eventos.
- Em Picat, as regras de ação descrevem comportamentos dos atores.
- Um ator recebe um objeto e dispara uma ação.
- Os eventos são postados via canais de mensagem e um ator pode ser conectado há um canal, verificar e/ou processar seus eventos postados no canal.

Características: P-I-C-A-T

Tabling:

- Considerando que operações entre variáveis podem ser armazenadas parcialmente em uma tabela na memória, permitindo que um programa acesse valores já calculados.
- Assim, evita-se a repetição de operações já realizadas.
- Com esta técnica de *memoization*, o Picat oferece soluções imediatas para problemas de programação dinâmica.

Comparações

Tabela: Comparativo entre algumas linguagens:
`https://rosettacode.org/wiki/Language_Comparison_Table`

	C	Haskell	Java	Prolog	P.I.C.A.T
Paradigma(s)	procedural	funcional	orientado à objetos	lógico	multi-paradigma
Tipagem	fraca	forte	forte	fraca	fraca
Verificação de tipos	estático	estático	estático	dinâmico	dinâmico
Possui segurança?	não	sim	sim	não	sim
Possui coletor de lixo?	não	sim	sim	sim	sim
Passagem de parâmetros	valor	-	valor	valor	casamento
Legibilidade	baixa	média	média	média	boa

Usos

A linguagem Picat pode ser utilizada para diversas funções:

- Acadêmica
- Industrial
- Pesquisas

Sistema de Programação

- Picat é uma linguagem de multiplataforma, disponível em qualquer arquitetura de processamento e também de sistema operacional
- Utiliza a extensão .pi em seus arquivos de código fonte.
- Existem 2 modos de utilização do Picat: Modo linha de comando e Modo Interativo.

Vantagens

- Enfatiza uma visão moderna e controlável em seu mecanismo de backtracking.
- Clareza em construir regras declarativas.
- Funções disponíveis numa sintaxe análoga a Haskell com um ambiente de programação análogo ao Python.
- Biblioteca é organizada em módulo a exemplo de Haskell e Python.

Desvantagens

- Manteve as letras maiúsculas para variáveis, como feito no B-Prolog.
- A geração de um código executável ainda não é puro, ela ainda se encontra em desenvolvimento
- As estruturas de repetição, comparadas com outras imperativas, ficam com uma sintaxe diferente.

Tipos de Dados

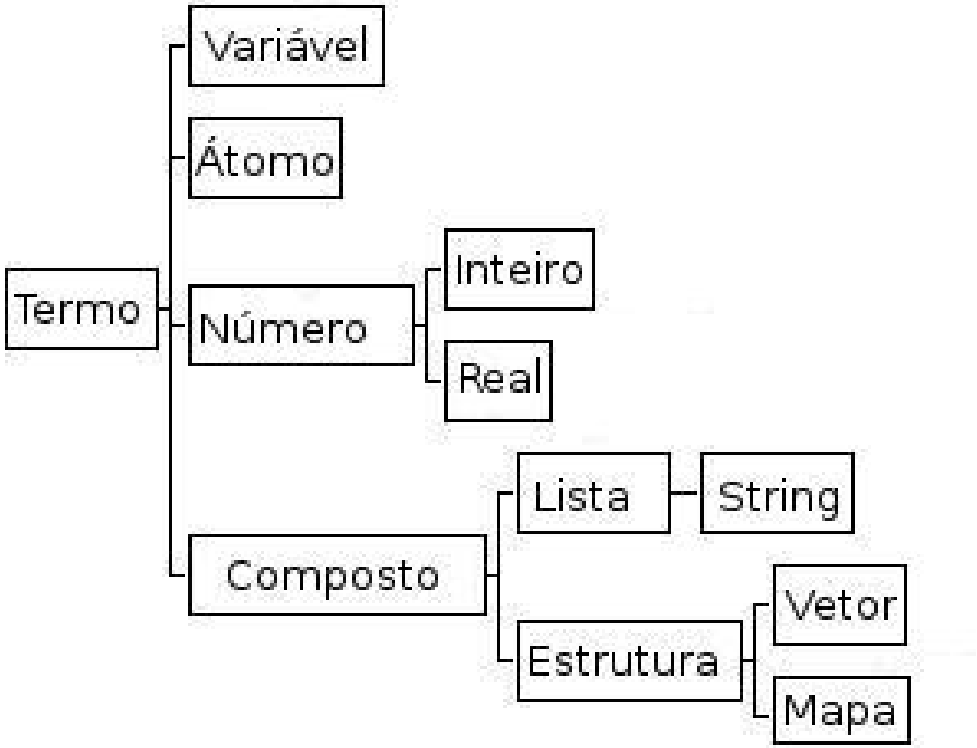


Figura: Hierarquia dos Tipos de dados

Variável

- As variáveis em Picat são similares as variáveis das matemática, pois ambas guardam valores. Diferentemente das linguagens imperativas, as variáveis em Picat não possuem um endereço simbólico na memória do computador.
- Quando uma variável ainda não foi instanciada com um valor, ela fica em um estado livre. Uma vez quando for instanciada com um valor, ela terá a mesma identidade como se fosse um valor até que ela seja liberada de novo.

Átomos

- Um átomo é uma constante simbólica e seu nome pode ser representado tanto com aspas simples ou sem.
- Um átomo não pode ultrapassar uma linha de comando e seu nome tem um limite de mil caracteres.
  Ex: x, x_1, 'a', 'b1'

Número

- Um número é um átomo inteiro ou real. Um número inteiro pode ser representado na forma decimal, binária, octal ou hexadecimal.
- Já o número real usa o ponto no lugar da virgula para separar os valores depois de zero como: 3.1415.

Número

```
Picat> A = 5, B = 7, number(A), number(B), max(A, B) =
Maximo, min(A, B) = Minimo.
A = 5
B = 7
Maximo = 7
Minimo = 5
yes.
```

Termos Compostos

Um termo composto se divide entre listas, estruturas e outros tipos compostos derivado destes são: *strings*, vetores e mapas. Entretanto, ambos tem seus elementos acessados via casamento de padrões de fatos, predicados e funções.

Listas

A forma de uma lista reúne um conjunto de termos e os coloca dentro de colchetes: [t1; t2; :::; tn]. Veja o exemplo:

Listas

```
Picat> A=[1,2,3], list(A), length(A)=L_A, B= [4,5,6],
list(B),
length(B) = L_B, A ++ B = C, list(C), length(C) = L_C.
A = [1,2,3]
L_A = 3
B = [4,5,6]
L_B = 3
C = [1,2,3,4,5,6]
L_C = 6
yes.
```

Estruturas

A forma de uma estrutura é definida como $s(t1, t2, ..., tn),onde s é um átomo e $ é usado para diferenciar uma função. Seus principais elementos são o nome da estrutura que é o átomo que fica na frente e a aridade (número de argumentos do predicado). Veja o exemplo:

Estruturas

```
Picat> N = $nome(1,2,3,4,5), struct(N), arity(N) = Aridade,
to_list(N) = Lista.
N = nome(1,2,3,4,5)
Aridade = 5
Lista = [1,2,3,4,5]
yes.
```

Estruturas

```
Picat> N = $(1,2,3,4,5), struct(N), arity(N) = Aridade,
to_list(N) = Lista.
N = (1,2,3,4,5)
Aridade = 2
Lista = [1,(2,3,4,5)]
yes.
```

Exemplos

Atribuição

```
Picat> X := 7, X := X + 7, X := X + 7.
X = 21
```

Estruturas de Controle

```
ex1 =>
X:=3, Y:=4,
if(X >= Y)
then printf("%d", X)
else printf("%d", Y)
end.
```

Entradas e Saídas

```
main =>
printf("Digite dois números:  "),
N_real01 = read_real(),
N_real02 = read_real(),
Media = (N_real01 + N_real02)/2,
printf("A média é:  %6.2f", Media),
printf("\n.....FIM.....\ n").
```

Dirigido aos estudantes de LMA da UDESC – 9 de novembro de 2016

- Ainda fora de ordem este material, mas acompanhe as explicações em sala de aula
- Traga o seu notebook para aula
- Dúvidas exemplos etc: `https://github.com/claudiosa/CCS`

Dirigido aos estudantes de LMA da UDESC

- Instalem o PICAT a partir de `http//www.picat-lang.org`
- Windows, Mac ou Linux
- Tenham um editor de código de programa.
  Sugestão: *geany* ou *sublime*

Fatos em Lógica – Exemplo 01

- `nome(joao)`, `nome(maria)`, etc, *aridade* $= 1$
- `idade_nome(18, joao)`, `idade_nome(19, maria)`, etc, *aridade* $= 2$
- `pai(pedro, joao)`, `pai(pedro, maria)`, etc, *aridade* $= 2$
- `idade_nome_sexo(18, joao, 'm')`, `idade_nome_sexo(19, maria, 'f')`, etc, *aridade* $= 3$
- `dados(futebol, 18, joao, 'm', joinville)`, `dados(natacao, 19, maria, 'f', blumenau)`, etc, *aridade* $= 5$

Fatos em Lógica – Generalizações

- `nome(joao)`
  `nome(maria)`
  $\therefore \exists x.nome(x)$ ou $\forall x.nome(x)$
- `idade_nome(18, joao)`
  `idade_nome(19, maria)`
  $\therefore \exists x \exists y.idade\_nome(x, y)$ ou $\forall x \exists y.idade\_nome(x, y)$
- `pai(pedro, joao)`
  `pai(pedro, maria)`
  $\therefore \exists x \exists y.pai(x, y)$ ou $\exists x \forall y.pai(x, y)$

Fatos em Lógica – Generalizações

- `nome(joao)`
  `nome(maria)`
  $\therefore \exists x.nome(x)$ ou $\forall x.nome(x)$
- `idade_nome(18, joao)`
  `idade_nome(19, maria)`
  $\therefore \exists x \exists y.idade\_nome(x, y)$ ou $\forall x \exists y.idade\_nome(x, y)$
- `pai(pedro, joao)`
  `pai(pedro, maria)`
  $\therefore \exists x \exists y.pai(x, y)$ ou $\exists x \forall y.pai(x, y)$
- Cuidar nas **generalizações** ... há muitas regras!
- Principalmente nas regras (fórmulas com conectivos) e fatos com $aridade \geq 2$

Fatos em PICAT (1)

```
1  %%% FATOS ... = Prolog
2  index(-)       % these facts are not ordered
3          nome(joao).
4          nome(maria).
5          nome(marcia).
6
7  index(-,-)      % these facts are not ordered
8      idade_nome(18, joao).
9      idade_nome(19, maria).
10 index(-,-)
11     pai(pedro, joao).
12     pai(pedro, maria).
13 index(-,-,-)
14     idade_nome_sexo(18, joao, 'm').
15     idade_nome_sexo(19, maria, 'f').
16 index(-,-,-,-,-)
17     dados(futebol, 18, joao, 'm', joinville).
18     dados(natacao, 19, maria, 'f', blumenau).
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20 listar_nomes  ?=>      %%% this rule is backtrackable
21           nome(X)                          ,   %% and
```

Fatos em PICAT (2)

```
22            printf("\n Nome: %w ", X)   ,    %% and
23            false.
24
25 listar_nomes =>
26            printf("\n ")   ,
27            true. %% the final rule of above
28 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29 listar_ida_nomes   ?=>
30            idade_nome(Y,X) ,
31            printf("\n Nome: %w  Idade: %w", X, Y)   ,
32            false.
33
34 listar_ida_nomes =>
35            printf("\n ")   ,
36            true. %% the final rule of above
37 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
38 listar_dados   ?=>
39            dados(X1,X2,X3,X4,X5) ,
40            %%dados(futebol, 18, joao, 'm', joinville).
41            printf("\n Nome: %w  Idade: %d Sexo: %c Joga: %w Mora: %w",
42            X3, X2, X4, X1, X5),
43            false.
```

Fatos em PICAT (3)

```
44
45 listar_dados =>
46           printf("\n ")   ,
47           true. %% the final rule of above
48
49 main => listar_nomes,
50         listar_ida_nomes,
51         listar_dados.
52
53 /*
54 1. Ponto  final  no final da linha de regras er fatos
55 2. Termos ou atomos: letras minusculas ... SEMPRE
56 3. Letras maiusculas = variaveis das outras Linguagens
57 4. A maquina anda de cima para baixo ....
58 5.  Embora a ordem do codigo nao interessa
59 6.   :- do Prolog e agora ?=>
60 7.  Contudo so assim "=>" nao tem backtracking
61 */
```

Experimente: `$ picat fatos_ex_01.pi`

$\Rightarrow$ as saídas são **particularizações** (PU e PE)

Regras em Lógica – Exemplo 01: os mortais!

- *homem*(*adao*)
- *homem*(*platao*)
- *homem*(*socrates*)

leia-se: *Adão é um homem*

leia-se: *Platão é um homem*

leia-se: *Sócrates é um homem*

Regras em Lógica – Exemplo 01: os mortais!

- *homem*(*adao*)
- *homem*(*platao*)
- *homem*(*socrates*)
- Todos homens sao mortais

leia-se: *Adão é um homem*

leia-se: *Platão é um homem*

leia-se: *Sócrates é um homem*

Regras em Lógica – Exemplo 01: os mortais!

- $homem(adao)$ leia-se: *Adão é um homem*
- $homem(platao)$ leia-se: *Platão é um homem*
- $homem(socrates)$ leia-se: *Sócrates é um homem*
- Todos homens sao mortais
- $\forall x.(homem(x) \rightarrow mortal(x))$
- A LPO usa um raciocínio dedutivo $\Rightarrow$ pesquise sobre isto!

Regras em PICAT (1)

```
1  %%% FATOS ...
2  index(-) % These facts are not ordered
3  homem( platao )        .
4  homem( socrates )      .
5  homem( adao )          .
6
7  %% uma regra da LPO: Todos homens sao mortais
8  mortal(X)  => homem(X)  .
9
10 listar_todos  ?=>     %%% this rule is backtrackable
11           mortal(X) ,
12           printf("\n Homem mortal: %w ", X) ,
13           false.
14
15 listar_todos =>
16           printf("\n The End \n ")  ,
17           true. %% the final rule of above
18
19 main  => listar_todos.
```

Regras em Lógica – Exemplo 02: os pais!

- $pai(platao, luna)$
- $pai(platao, pricles)$
- $pai(epimenides, platao)$

leia-se: *Platão é o pai de Luna*

leia-se: *Platão é o pai de Péricles*

leia-se: *Sócrates é o pai de Platão*

Regras em Lógica – Exemplo 02: os pais!

- $pai(platao, luna)$ leia-se: *Platão é o pai de Luna*
- $pai(platao, pricles)$ leia-se: *Platão é o pai de Péricles*
- $pai(epimenides, platao)$ leia-se: *Sócrates é o pai de Platão*
- Alguém que é avô tem um filho que é um pai de alguém

Regras em Lógica – Exemplo 02: os pais!

- $pai(platao, luna)$ leia-se: *Platão é o pai de Luna*
- $pai(platao, pricles)$ leia-se: *Platão é o pai de Péricles*
- $pai(epimenides, platao)$ leia-se: *Sócrates é o pai de Platão*
- Alguém que é avô tem um filho que é um pai de alguém
- Alguém que é irmão tem o mesmo pai e não é irmão consigo mesmo

Regras em Lógica – Exemplo 02: os pais!

- $pai(platao, luna)$ leia-se: *Platão é o pai de Luna*
- $pai(platao, pricles)$ leia-se: *Platão é o pai de Péricles*
- $pai(epimenides, platao)$ leia-se: *Sócrates é o pai de Platão*
- Alguém que é avô tem um filho que é um pai de alguém
- Alguém que é irmão tem o mesmo pai e não é irmão consigo mesmo
- As regras estão nos slides de lógica
- $\Rightarrow$ as saídas são **particularizações** (PU e PE)

Regras em PICAT (1)

```
1  %%% FATOS ...  desenhe a arvore geneologica
2  index(-,-)
3          pai(platao, luna).
4          pai(platao, pericles).
5          pai(platao, eratostenes).
6          pai(epimenides, platao).
7  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8  listar_pais  ?=>
9            pai(X,Y) ,
10           printf("\n ==> %w  e pai de  %w", X , Y)  ,
11           false.
12
13 listar_pais =>
14           printf("\n ")  ,
15           true. %% the final rule of above
16
17 listar_avos  ?=>
18           avo(X,Y) ,
19           printf("\n ==> %w  e avo de  %w", X , Y)  ,
20           false.
21
```

Regras em PICAT (2)

```
listar_avos =>
          printf("\n ")  ,
          true. %% the final rule of above
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
avo(X,Y) ?=> pai(X,Z), pai(Z,Y).

irmao(X,Y) ?=> pai(Z,X), pai(Z,Y), X !== Y.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% main padrao
main ?=> listar_pais,
        avo(X,Y), printf("\n ==> %w  eh avo de  %w", X , Y)  ,
    irmao(Z,W), printf("\n ==> %w  eh irmao de  %w", Z , W),
    false.
main => true.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Regras em Lógica – Exemplo 03: entendendo *backtracking*

- O exemplo a seguir é simples para entender o que é *backtracking*

Regras em Lógica – Exemplo 03: entendendo *backtracking*

- O exemplo a seguir é simples para entender o que é *backtracking*
- Há vários conceitos embutidos, logo, preste atenção nas expicações em aula

Regras em Lógica – Exemplo 03: entendendo *backtracking*

- O exemplo a seguir é simples para entender o que é *backtracking*
- Há vários conceitos embutidos, logo, preste atenção nas expicações em aula
- Finalmente, faça variações sobre o predicado chamado: `regra`

Regras em Lógica – Exemplo 03: entendendo *backtracking*

- O exemplo a seguir é simples para entender o que é *backtracking*
- Há vários conceitos embutidos, logo, preste atenção nas expicações em aula
- Finalmente, faça variações sobre o predicado chamado: `regra`
- Veja os resultados diversos e se voce entendeu!
  Ai sim, avance!

Regras em PICAT (1)

```
%%% FATOS ...
index(-)       % fatos instanciados como retorno
     f1(a).
     f1(b).

index(-)       % fatos instanciados como retorno
    f2(1).
    f2(2).

index(-)        % fatos instanciados como retorno
    f3('#').
    f3('@').
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
regra(XXX, YYY, ZZZ)  =>      %%% this rule is NOT backtrackable
                f1(ZZZ),
                f2(XXX),
                f3(YYY).
%% EXERCICIO: TROQUE AS ORDENS das VARIAVEIS e dos fatos
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lst_vars ?=> %%% this rule is  backtrackable
   regra(X,Y,Z),
```

Regras em PICAT (2)

```
22      printf("\n X: %w \t Y: %w \t Z: %w ", X,Y,Z)   ,    %% and
23      false.
24
25  lst_vars =>
26          printf("\n\n FIM DOS FATOS \n\n ")   ,
27          true. %% the final rule of above
28  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29  main => lst_vars.
```

$\Rightarrow$ as saídas são **particularizações** (PU e PE)

Regras em Lógica – Exemplo 04: os conexões entre cidades

- as estradas existentes:

(1)	$estrada(joinville, itajai)$
(2)	$estrada(joinville, blumenau)$
(3)	$estrada(itajai, balneariocamboriu)$
(4)	$estrada(blumenau, balneariocamboriu)$
(5)	$estrada(balneariocamboriu, florianopolis)$
(6)	$\forall x \; \exists y : estrada(x, y) \rightarrow caminho(x, y)$
(7)	$\forall x \; \exists z \; \exists y : estrada(x, z) \wedge caminho(z, y) \rightarrow caminho(x, y)$

- Veja os comentários nos slides de lógica

Regras em PICAT (1)

```
1
2  %%% FATOS ... = Mapa
3  index(-, -)
4  estrada(joinville , itajai )          .
5  estrada(joinville , blumenau )        .
6  estrada(itajai    , camboriu )        .
7  estrada(blumenau  , camboriu)         .
8  estrada(camboriu  , floripa)          .
9
10 % As regra da LPO agora em PICAT
11 caminho(X,Y) ?=> estrada(X,Y) .
12 caminho(X,Y)  => estrada(X,Z) ,   caminho(Z,Y) .
13
14 listar_todos  ?=>     %%% this rule is backtrackable
15           caminho(X,Y)                              ,   %% and
16           printf("\n X: %w ---> Y: %w ", X,Y)   ,   %% and
17           false.
18
19 listar_todos =>
20           printf("\n ")   ,
21           true. %% the final rule of above
```

Regras em PICAT (2)

```
22
23 %% aconselhavel o ... uso do main
24 main =>
25      listar_todos  .
```

$\Rightarrow$ as saídas são **particularizações** (PU e PE)

Exercícios

Resolva e implemente em PICAT os seguintes problemas:

- Seja o conjunto das seguintes fórmulas em lógica de primeira-ordem (LPO), as quais descrevem o comportamento de um adversário autônomo (NPC–*nerd por computador*) em um videogame:

1.	agente(oponente)
2.	estado(oponente, fome)
3.	fruta(banana)
4.	fruta(laranja)
5.	sanduiche(bigmac)
6.	$\forall x \exists y : (fruta(x) \vee sanduiche(y) \rightarrow alimento(x))$
7.	$\exists z \forall x : (agente(z) \wedge estado(z, fome) \wedge alimento(x) \rightarrow decisao(z, comer, x))$

Demonstre as possíveis decisões que o **agente** pode executar quando está no estado "`fome`".

$\Rightarrow$ as saídas são **particularizações** (PU e PE)

Exercícios

continuação:

- Seja o conjunto das seguintes fórmulas em lógica de primeira-ordem (LPO), as quais descrevem uma estória:

1.	$\forall y \exists x(pessoa(y) \wedge pet(x) \wedge vacinado(x) \rightarrow ama(y,x))$
2.	$\forall x(pet(x) \wedge saudavel(x) \rightarrow vacinado(x))$
3.	$pessoa(mickey)$
4.	$pet(pluto)$
5.	$pet(garfield)$
6.	$saudavel(pluto)$
7.	$saudavel(garfield)$

Responda quem é saudável, vacinado, e quem *ama* o quem?

$\Rightarrow$ as saídas são **particularizações** (PU e PE)

Recursão em PICAT

- A recursão já foi usada nos exemplos da cidade, dos pais, etc

Recursão em PICAT

- A recursão já foi usada nos exemplos da cidade, dos pais, etc
- O objetivo é buscar a solução n-ésima instância na solução da $(n-1)$-ésima instância

Recursão em PICAT

- A recursão já foi usada nos exemplos da cidade, dos pais, etc
- O objetivo é buscar a solução n-ésima instância na solução da $(n-1)$-ésima instância
- Raciocínio análogo (e o contrário) da hipótese indutiva

Recursão em PICAT

Exemplo da soma de 0 a N (inteiro, positivo)

- $0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$

Recursão em PICAT

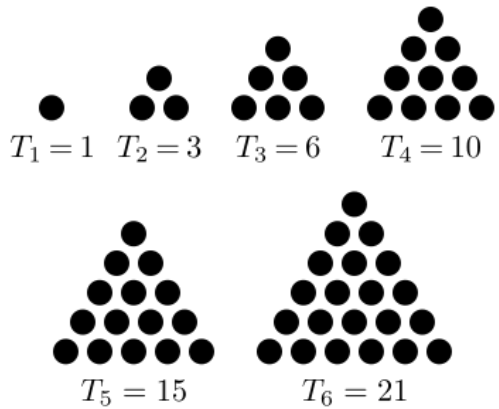
Exemplo da soma de 0 a N (inteiro, positivo)

- $0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$
- Ilustrando:

Recursão em PICAT

Exemplo da soma de 0 a N (inteiro, positivo)

- $0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$
- Ilustrando:

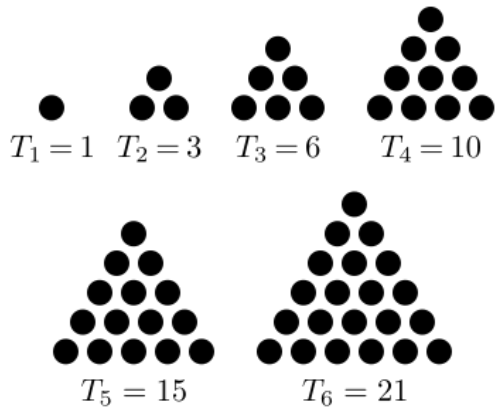


- Aqui temos a fórmula da soma, e se não tivéssemos?

Recursão em PICAT

Exemplo da soma de 0 a N (inteiro, positivo)

- $0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$
- Ilustrando:



- Aqui temos a fórmula da soma, e se não tivéssemos?
- Eis a recursão ...

Recursão em PICAT

Exemplo da soma de 0 a N (inteiro, positivo)

$$S(n) = 0 + 1 + 2 + 3 + 4 + ... + (n-1) + n$$

Recursão em PICAT

Exemplo da soma de 0 a N (inteiro, positivo)

$$S(n) = 0 + 1 + 2 + 3 + 4 + ... + (n-1) + n$$

Reformulado sob uma visão da indução finita como:

Recursão em PICAT

Exemplo da soma de 0 a N (inteiro, positivo)

$$S(n) = 0 + 1 + 2 + 3 + 4 + ... + (n-1) + n$$

Reformulado sob uma visão da indução finita como:

$$S(n) = \begin{cases} 0 & \text{para } n = 0 \\ S(n-1) + n & \text{para } n \geqslant 1 \end{cases}$$

Recursão em PICAT

Explicando

- O procedimento é recursivo,
- Contudo, deve-se encontrar a definição para a "**parada**" da recursividade.
- Como n não tem limite superior, então n, inicia-se pelo que se conhece, $n = 0$

Recursão em PICAT

Explicando

- O procedimento é recursivo,
- Contudo, deve-se encontrar a definição para a "**parada**" da recursividade.
- Como n não tem limite superior, então n, inicia-se pelo que se conhece, $n = 0$
- Em um procedimento algoritmico:

#1. A soma até 0 é 0, logo: `soma(0,0).`
#2. Para soma dos n-ésimos termos, é necessário a soma do $(n-1)$-ésimos termos
Assim uma fórmula recursiva é deduzida como:

```
soma(N,S) = Nant := (N-1) ∧
soma(Nant, S_Nant) ∧
S := (N + S_Nant).
```

Recursão em PICAT

Explicando

- O procedimento é recursivo,
- Contudo, deve-se encontrar a definição para a "**parada**" da recursividade.
- Como n não tem limite superior, então n, inicia-se pelo que se conhece, $n = 0$
- Em um procedimento algoritmico:

#1. A soma até 0 é 0, logo: `soma(0,0).`
#2. Para soma dos n-ésimos termos, é necessário a soma do $(n-1)$-ésimos termos
Assim uma fórmula recursiva é deduzida como:

```
soma(N,S) = Nant := (N-1) ∧
soma(Nant, S_Nant) ∧
S := (N + S_Nant).
```

- Entendido?

Recursão em PICAT

Explicando

- O procedimento é recursivo,
- Contudo, deve-se encontrar a definição para a "**parada**" da recursividade.
- Como n não tem limite superior, então n, inicia-se pelo que se conhece, $n = 0$
- Em um procedimento algoritmico:

#1. A soma até 0 é 0, logo: `soma(0,0).`
#2. Para soma dos n-ésimos termos, é necessário a soma do $(n-1)$-ésimos termos
Assim uma fórmula recursiva é deduzida como:

```
soma(N,S) = Nant := (N-1) ∧
soma(Nant, S_Nant) ∧
S := (N + S_Nant).
```

- Entendido? $\Rightarrow$ **Agora basta escrever em PICAT**

Recursão em PICAT (1)

```
1  % Soma de 0 + 1 + 2 + 3 + .... + N
2  main => soma_p1(7, X) ,
3          printf("\n soma_P1: %d " , X) ,
4          soma_p2(7, Y) ,
5          printf("\n soma_P2: %d " , Y) .
6
7  /* recursividade CLASSICA -- prolog like*/
8  soma_p1(0,S) ?=> S = 0.    % regra backtrackable
9  soma_p1( N, S ) => N > 0, %%% regra recursiva
10                    Ant  =  (N - 1),
11                    soma_p1( Ant , Parcial ),
12                    S = (N + Parcial) .
13
14 /* simplificada */
15 soma_p2(0,S) ?=> S = 0.
16 soma_p2(N,S) , N > 0 =>
17               soma_p2( N-1 , Parcial ),
18               S = N + Parcial.
```

Variações do Problema da Soma (1)

Reusando todo aprendizado o exemplo anterior:

```
1  % Soma como predicado
2
3  main => soma_p1(7, X),
4          printf("\n soma_P1: %d " , X),
5          soma_p2(7, Y),
6          printf("\n soma_P2: %d " , Y),
7          printf("\n soma_f1: %d " , soma_f1(7) ),
8          printf("\n soma_f2: %d " , soma_f2(7) ).
9
10 /* recursividade CLASSICA -- prolog like*/
11 soma_p1(0,S) ?=> S = 0.    % regra backtrackable
12 soma_p1( N, S ) => N > 0, %%% regra recursiva
13                    Ant  =  (N - 1),
14                    soma_p1( Ant , Parcial ),
15                    S = (N + Parcial) .
16
17 /* simplificada */
18 soma_p2(0,S) ?=> S = 0.
19 soma_p2(N,S) , N > 0 =>
20              soma_p2( N-1 , Parcial ),
```

Variações do Problema da Soma (2)

```
21              S = N + Parcial.
22
23 % Soma como funcao -- visao classica
24 soma_f1(0) = S => S = 0.
25 soma_f1(N) = S, N >= 1 =>
26           S := N + soma_f1(N-1).
27
28 % Soma como funcao de fatos -- proximo a Haskell
29 soma_f2(0) = 0.
30 soma_f2(N) = N + soma_f2( N-1 ).
```

Conclusão

- PICAT é uma linguagem nova (2013), desconhecida, revolucionária e com um futuro promissor para áreas de pesquisas e utilização comercial.
- Atualmente há pouco material disponível e uma comunidade pequena de usuários, mas existe um site atualizado e mantido por Hakan Kjellerstrand e um fórum de discussão no próprio site que está cada dia mais ativo, graças ao crescimento de usuários desta linguagem.

Referências

- https://github.com/claudiosa/CCS/tree/master/picat
- http://picat-lang.org/

Questionário

1. Qual característica do P.I.C.A.T é mais chamativa?
2. Em quais aplicações você usaria P.I.C.A.T?
3. Quais são os pontos positivos e negativos do P.I.C.A.T que você identifica?
4. Se pudesse melhorar algo no P.I.C.A.T, o que melhoraria?
5. O P.I.C.A.T pode substituir alguma linguagem?