PICAT: Uma Linguagem de Programação Multiparadigma

Claudio Cesar de Sá

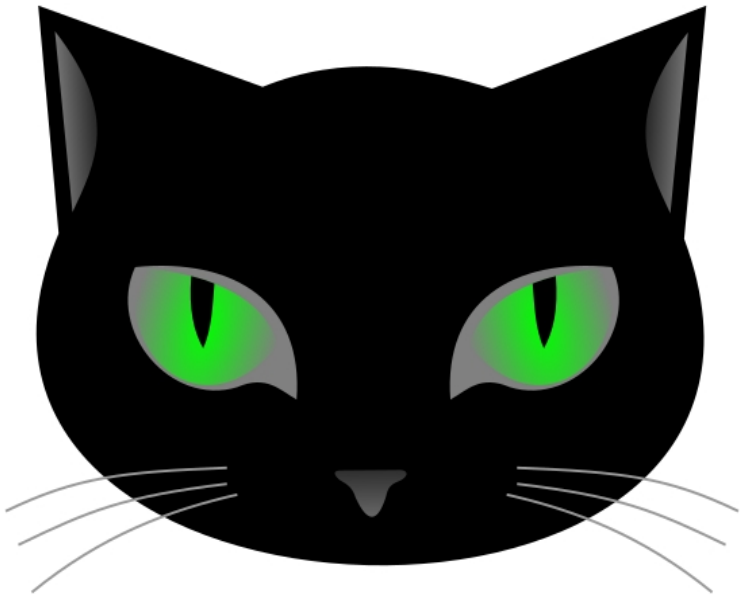
✉ claudio.sa@udesc.br

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

3 de junho de 2019

Depuração

- O que é depuração?
- Como funciona no Picat?
- Exemplo de uso



Depuração – Quando usar?

- Depuração, *trace* ou *debug* – o que é?

Depuração – Quando usar?

- Depuração, *trace* ou *debug* – o que é? Mostrar estado de variáveis, funções e predicados durante a execução de um código
- Depuração ou *trace* – quando usar?

Depuração – Quando usar?

- Depuração, *trace* ou *debug* – o que é? Mostrar estado de variáveis, funções e predicados durante a execução de um código
- Depuração ou *trace* – quando usar? **Aprender detalhes da linguagem**

Depuração – Quando usar?

- Depuração, *trace* ou *debug* – o que é? Mostrar estado de variáveis, funções e predicados durante a execução de um código
- Depuração ou *trace* – quando usar? **Aprender detalhes da linguagem**
- Depuração ou *trace* $\Rightarrow$ **para descobrir o erro!**

Depuração – Quando usar?

- Depuração, *trace* ou *debug* – o que é? Mostrar estado de variáveis, funções e predicados durante a execução de um código
- Depuração ou *trace* – quando usar? **Aprender detalhes da linguagem**
- Depuração ou *trace* $\Rightarrow$ **para descobrir o erro!**
- Em Picat não há um ambiente gráfico de depuração, ao contrário do SWI-Prolog.

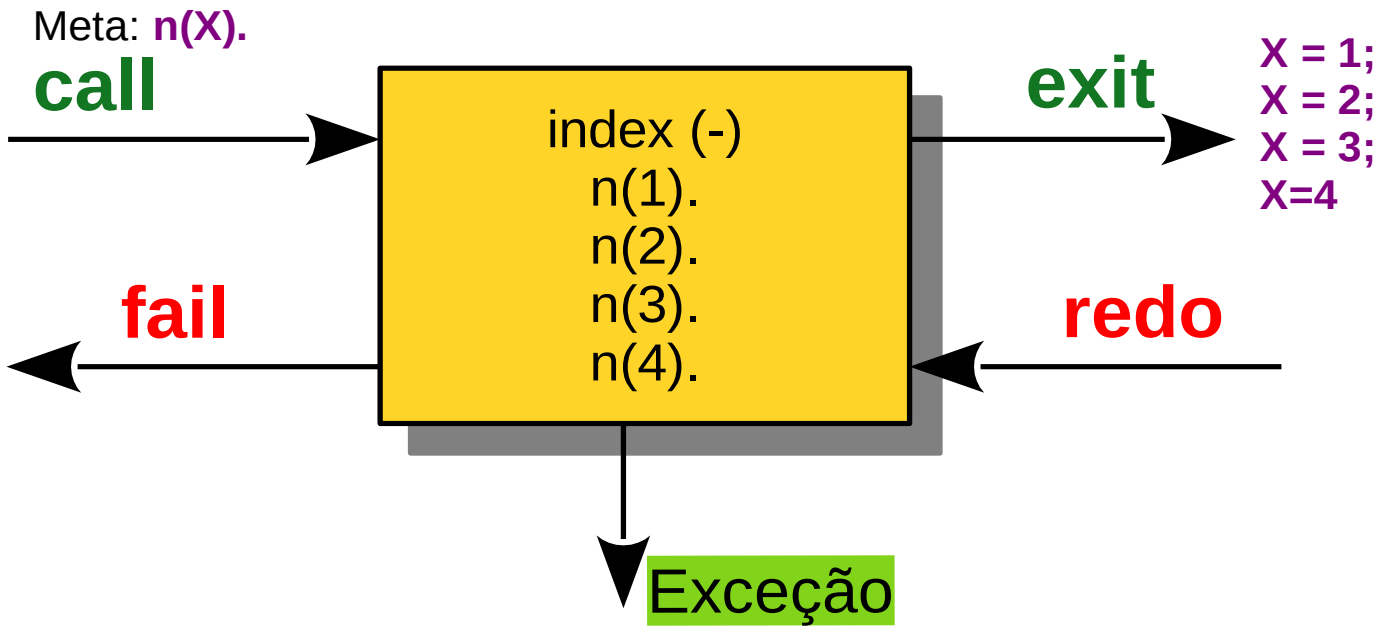
Depuração – Quando usar?

- Depuração, *trace* ou *debug* – o que é? Mostrar estado de variáveis, funções e predicados durante a execução de um código
- Depuração ou *trace* – quando usar? **Aprender detalhes da linguagem**
- Depuração ou *trace* $\Rightarrow$ **para descobrir o erro!**
- Em Picat não há um ambiente gráfico de depuração, ao contrário do SWI-Prolog. Eis uma oportunidade de um projeto aqui....
- Contudo, o *trace* do Picat faz tudo ...

Depuração – Quando usar?

- Depuração, *trace* ou *debug* – o que é? Mostrar estado de variáveis, funções e predicados durante a execução de um código
- Depuração ou *trace* – quando usar? **Aprender detalhes da linguagem**
- Depuração ou *trace* $\Rightarrow$ **para descobrir o erro!**
- Em Picat não há um ambiente gráfico de depuração, ao contrário do SWI-Prolog. Eis uma oportunidade de um projeto aqui....
- Contudo, o *trace* do Picat faz tudo ...

Depuração – Como funciona um predicado?



Depuração – Internamente tem o *backtracking*!

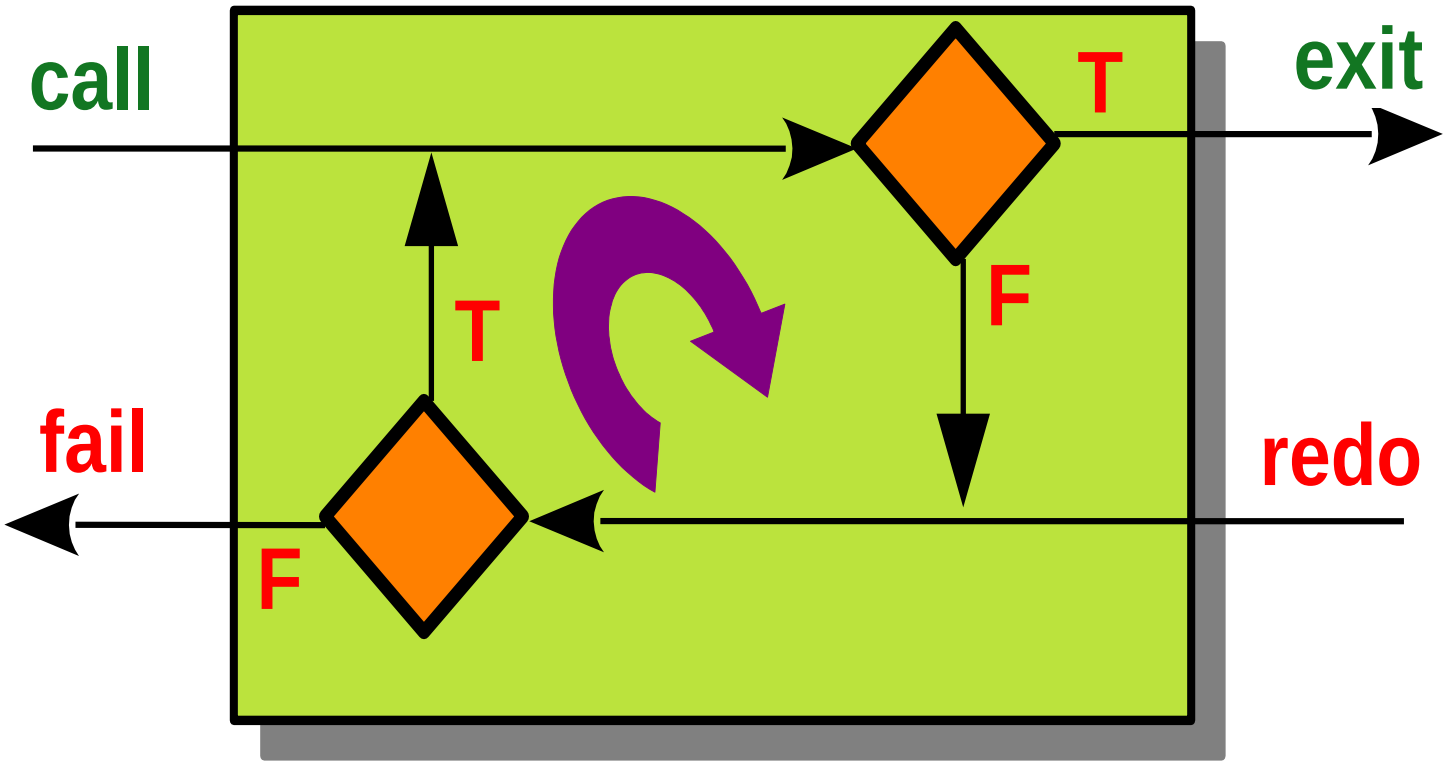
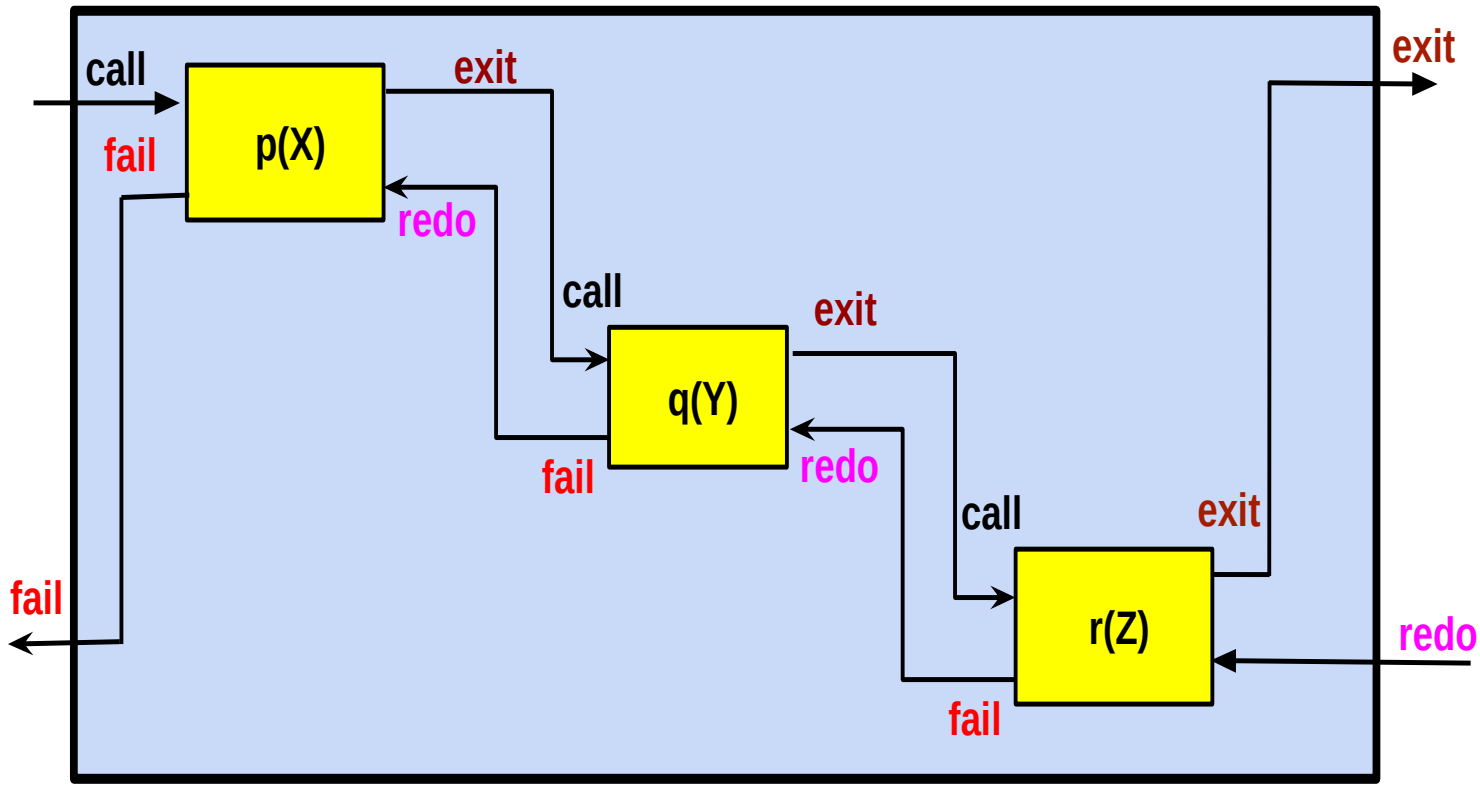


Figura 1: Levemente diferente do que é encontrado na bibliografia!

Depuração – uma regra composta

```
regra(X,Y,Z) => p(X), q(Y), r(Z).
```



- Picat tem **3 modos de execução**: *trace*, *non-trace* e *spy*

Depuração

- Picat tem **3 modos de execução**: *trace*, *non-trace* e *spy*
- *non-trace*: é a execução normal ou *default* – vista até o momento

Depuração

- Picat tem **3 modos de execução**: *trace*, *non-trace* e *spy*
- *non-trace*: é a execução normal ou *default* – vista até o momento
- *trace* ou *debug*: vê passo-a-passo de cada variável, predicado ou função, do código inteiro

Depuração

- Picat tem **3 modos de execução**: *trace*, *non-trace* e *spy*
- *non-trace*: é a execução normal ou *default* – vista até o momento
- *trace* ou *debug*: vê passo-a-passo de cada variável, predicado ou função, do código inteiro
- *spy*: especifica um predicado ou função – útil para habilitar e desabilitar partes do código

Depuração

- Picat tem **3 modos de execução**: *trace*, *non-trace* e *spy*
- *non-trace*: é a execução normal ou *default* – vista até o momento
- *trace* ou *debug*: vê passo-a-passo de cada variável, predicado ou função, do código inteiro
- *spy*: especifica um predicado ou função – útil para habilitar e desabilitar partes do código
- Devem ser habilitado e desabilitados no **modo interpretado**
- Há funções e predicados que não tem *backtracking*.

Depuração

- Picat tem **3 modos de execução**: *trace*, *non-trace* e *spy*
- *non-trace*: é a execução normal ou *default* – vista até o momento
- *trace* ou *debug*: vê passo-a-passo de cada variável, predicado ou função, do código inteiro
- *spy*: especifica um predicado ou função – útil para habilitar e desabilitar partes do código
- Devem ser habilitado e desabilitados no **modo interpretado**
- Há funções e predicados que não tem *backtracking*.
  Exemplos: `nl`, `printf`, `read`, etc

Uso básico do *trace*

1. Entre no interpretador do Picat
2. Ative o modo *trace* (ou *debug*), com o comando: *trace*
3. Compile e carregue o programa desejado a depurar, com o comando: `cl('....')`.
4. Execute a parte desejada do código
5. Vá dando `Enter` no modo *passo-a-passo*
6. Digite '**h**' para o `help` e ver opções disponíveis
7. Digite '**a**' para o `abortar` o modo trace

Em qualquer modificação do código fonte no editor, repita os passos acima, desde o início!

Exemplo: X + Y = 22

```
main ?=>
    %%%%% ACHE X e Y tal que  X + Y = 22
    regra(X,Y, 22) ,
    printf("SAIDAS ==> X: %w Y: %w \n", X, Y)  ,
    false.
main =>  printf("\n Não há mais soluções! \n").
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
regra(X,Y, R) =>
      p(X),
      q(Y),
      R = X + Y.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
index(-)
     p(2).     p(1).     p(0).
index(-)
     q(20).    q(22).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Saída

```
Picat> trace
Note: you need to recompile programs in debug mode for tracing
yes
{Trace mode}
Picat>  cl('trace_exemplo_01')
Compiling:: trace_exemplo_01.pi
trace_exemplo_01.pi compiled in 3 milliseconds
loading...
yes
{Trace mode}
Picat> q(X), writeln(x = X), false.
   Call: (1) q(_10b38) ?
?  Exit: (1) q(20) ?
   Call: (2) writeln(x = 20) ?
x = 20
   Exit: (2) writeln(x = 20) ?
   Redo: (1) q(20) ?              >>>> DEVIDO o false
   Exit: (1) q(22) ?
   Call: (3) writeln(x = 22) ?
x = 22
   Exit: (3) writeln(x = 22) ?
no
{Trace mode}
```

Considerações de Uso do *trace*

- Este comando é muito útil e se encontra em modo *beta* de uso
- Mas, tem **TUDO** que se precisa para descobrir um erro

Considerações de Uso do *trace*

- Este comando é muito útil e se encontra em modo *beta* de uso
- Mas, tem **TUDO** que se precisa para descobrir um erro
- A idéia de se programar com Picat e outras linguagens declarativas é:

Considerações de Uso do *trace*

- Este comando é muito útil e se encontra em modo *beta* de uso
- Mas, tem **TUDO** que se precisa para descobrir um erro
- A idéia de se programar com Picat e outras linguagens declarativas é:
  a cada linha de código escrita segue-se por um teste isolado!

Considerações de Uso do *trace*

- Este comando é muito útil e se encontra em modo *beta* de uso
- Mas, tem **TUDO** que se precisa para descobrir um erro
- A idéia de se programar com Picat e outras linguagens declarativas é:
  a cada linha de código escrita segue-se por um teste isolado!
- Ou seja, use o *trace* em partes do código!

Considerações de Uso do *trace*

- Este comando é muito útil e se encontra em modo *beta* de uso
- Mas, tem **TUDO** que se precisa para descobrir um erro
- A idéia de se programar com Picat e outras linguagens declarativas é:
  a cada linha de código escrita segue-se por um teste isolado!
- Ou seja, use o *trace* em partes do código!
- **Os números após a chamadas, representam a sequência na pilha de cada predicado!**

Considerações de Uso do *trace*

- Este comando é muito útil e se encontra em modo *beta* de uso
- Mas, tem **TUDO** que se precisa para descobrir um erro
- A idéia de se programar com Picat e outras linguagens declarativas é:
  a cada linha de código escrita segue-se por um teste isolado!
- Ou seja, use o *trace* em partes do código!
- **Os números após a chamadas, representam a sequência na pilha de cada predicado!**
- Ao se ganhar confiança com a linguagem, voce vai omitindo alguns pontos!

Reflexões

- Se aprende os detalhes da linguagem com um *trace*

Reflexões

- Se aprende os detalhes da linguagem com um *trace*
- Não se assuste com modo `trace` na console

Reflexões

- Se aprende os detalhes da linguagem com um *trace*
- Não se assuste com modo `trace` na console
- Atenção na sequência de uso:
  1. Ative o modo *trace*
  2. Compile e carregue o seu fonte no modo trace, com o comando `cl('.....').`
  3. Execute o que desejares do código

Reflexões

- Se aprende os detalhes da linguagem com um *trace*
- Não se assuste com modo `trace` na console
- Atenção na sequência de uso:
  1. Ative o modo *trace*
  2. Compile e carregue o seu fonte no modo trace, com o comando `cl('.....').`
  3. Execute o que desejares do código

  Em caso de mudança no código, repita os 3 passos acima

Reflexões

- Se aprende os detalhes da linguagem com um *trace*
- Não se assuste com modo `trace` na console
- Atenção na sequência de uso:
  1. Ative o modo *trace*
  2. Compile e carregue o seu fonte no modo trace, com o comando `cl('.....').`
  3. Execute o que desejares do código

  Em caso de mudança no código, repita os 3 passos acima
- Finalmente, *boas depurações*!