

PICAT: Uma Linguagem de Programação Multiparadigma

Claudio Cesar de Sá

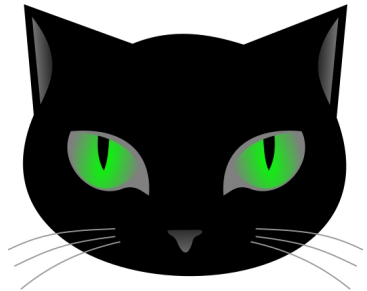
✉ claudio.sa@udesc.br

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

2 de junho de 2019



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



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



- 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, *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, *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, *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.



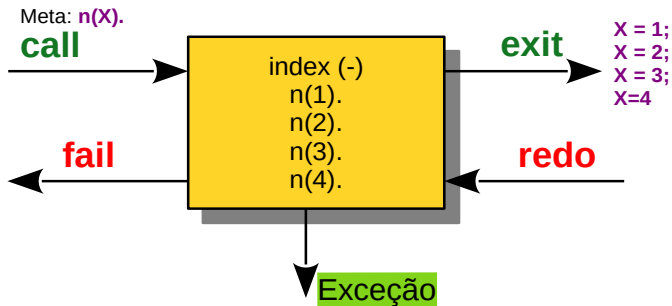
- Depuração, *trace* ou *debug* – o que é? Mostrar estado de variáveis, funções e predicados durante a execução de um código
- Depuração ou *trace* – quando usar? **Aprender detalhes da linguagem**
- Depuração ou *trace* \Rightarrow **para descobrir o erro!**
- Em Picat não há um ambiente gráfico de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



- Depuração, *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*!

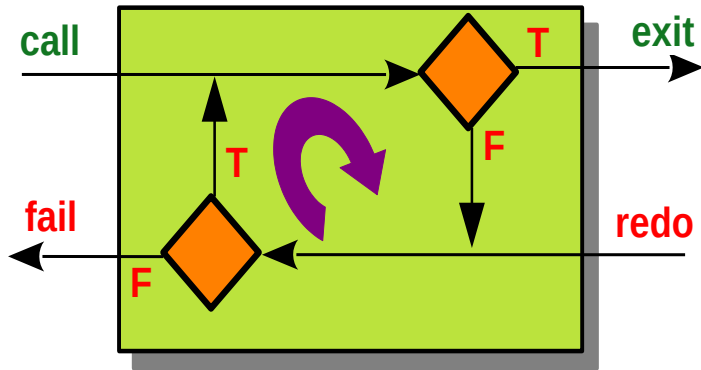
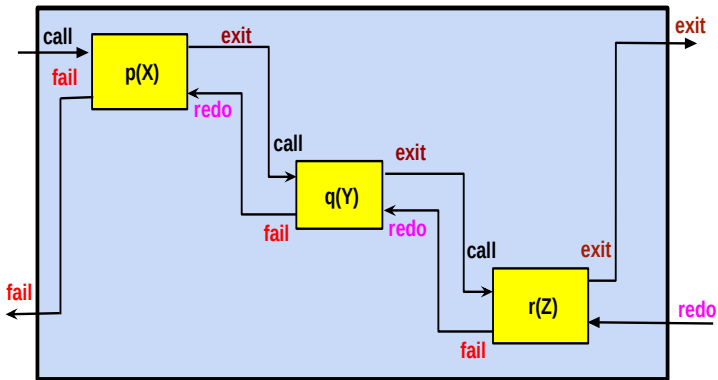


Figura 1: Levemente diferente do que é encontrado na bibliografia!



Depuração – uma regra composta

$\text{regra}(X,Y,Z) \Rightarrow p(X), q(Y), r(Z).$



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



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



- 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



- 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.



- 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*



- 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



- 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



```

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

```



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



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



- Este comando é muito útil e se encontra em modo bem primitivo 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!



- Este comando é muito útil e se encontra em modo bem primitivo 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!



- Este comando é muito útil e se encontra em modo bem primitivo 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!
- Ao se ganhar confiança com a linguagem, voce vai omitindo alguns pontos!



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



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



- 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 `c1('.....')`.
 - ③ Execute o que desejares do código



- 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 `c1('.....')`.
 - ③ Execute o que desejares do código
- Em caso de mudança no código, repita os 3 passos acima



- 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 `c1('.....')`.
 - ③ Execute o que desejares do código
- Em caso de mudança no código, repita os 3 passos acima
- Finalmente, *boas depurações!*

