Formalização da Terminação de Especificações Funcionais

Thiago Mendonça Ferreira Ramos
Orientador: Mauricio Ayala-Rincón
Em Parceria com: Cesar Muñoz (NASA LaRC) &
Mariano Moscato (NIA) ...

Grupo de Teoria da Computação - GTC

Departmento de Ciência da Computação - CIC

Universidade de Brasília - UnB

Exame de mestrado

26 de Fevereiro de 2016

Motivação

- A garantia de correção de programa depende de sua terminação.
- O problema da parada é indecidível [1].
- Pode-se criar algoritmos de semi-decisão que respondem "sim" ou "não sei" para a terminação de um programa.
- Esses algoritmos estão cada vez melhores [2].
- O objetivo é formalizar a equivalência das principais definições de terminação para uma linguagem funcional de primeira ordem.

Estudo bibliográfico

- Turing [3] usa uma relação bem fundada nas chamadas de funções.
- Lee et al. [4] mostra terminação partir de um ω-automato.
- Manolius e Vroon [5] usam grafos que representam linhas de execução em funções para analizar terminação.
- Avelar [6] muda a técnica anterior usando matrizes de medida.

Itens de discursão

PVS0: uma linguagem mínima funcional

Terminação semântica

TCC Terminação

PVS0: As Teorias

Provas de equivalência

Uso da teoria

Proposta

PVS0: uma linguagem mínima funcional

- Aplicada a apenas um tipo de entrada e saída $\mathcal{V}al$.
- Um único argumento para recursão.
- Permite uma única função recursiva.
- Uso do assistente de prova PVS.

A gramática PVS0:

$$\mathcal{E}xpr ::= vr \mid cnst \mid op1(\mathcal{E}xpr) \mid op2(\mathcal{E}xpr, \mathcal{E}xpr)$$

$$\mid ite(\mathcal{E}xpr, \mathcal{E}xpr, \mathcal{E}xpr)$$

$$\mid rec(\mathcal{E}xpr)$$

Linguagem funcional de primeira ordem sobre um espaço de valores `T`, que representa $\mathcal{V}al$.

```
PVS0[T:TYPE+] :  DATATYPE WITH SUBTYPES Expr,Def
BEGIN
   cnst(get_val:T): cnst?:  Expr
   vr:  vr?:  Expr
   op1(get_op:nat,get_arg:Expr):  op1?:  Expr
   op2(get_op:nat,get_arg1,get_arg2:Expr):  op2?:  Expr
   rec(get_arg:Expr):  rec?:  Expr
   ite(get_cond,get_if,get_else:Expr):  ite?:  Expr
   def(get_body:Expr):  def?:  Def
END PVS0
```

Exemplo | código PVS0

$$
\begin{array}{l}
\mathit{fibonacci} := \\
\quad \mathit{ite}(\mathit{vr}, \\
\qquad 1_S, \\
\qquad \mathit{ite}(\mathit{op1}_0(\mathit{vr}), \\
\qquad\quad 1_S, \\
\qquad\quad \mathit{op2}_0(\mathit{rec}(\mathit{op1}_0(\mathit{vr})), \mathit{rec}(\mathit{op1}_1(\mathit{vr})))))
\end{array}
$$

```
fibonacci :=
   def(ite(vr,
      cnst(1),
      ite(op1(0,vr),
         cnst(1),
         op2(0,rec(op1(0,vr)),rec(op1(1,vr))))))
```

Terminação semântica

$$
\begin{array}{lrl}
\varepsilon(e, e_f, \beta, \nu) & := & \texttt{CASES } e \texttt{ OF} \\
& cnst: & \nu = cnst^{\mathfrak{I}}; \\
& vr: & \nu = \beta(vr); \\
& op1(e_1): & \exists \nu_1 \in \mathcal{V}al : \varepsilon(e_1, e_f, \beta, \nu_1) \wedge \\
& & \nu = op1^{\mathfrak{I}}(\nu_1); \\
& op2(e_1, e_2): & \exists \nu_1, \nu_2 \in \mathcal{V}al : \varepsilon(e_1, e_f, \beta, \nu_1) \wedge \varepsilon(e_2, e_f, \beta, \nu_2) \wedge \\
& & \nu = op2^{\mathfrak{I}}(\nu_1, \nu_2); \\
& ite(e_1, e_2, e_3): & \exists \nu_1 : \varepsilon(e_1, e_f, \beta, \nu_1) \wedge \\
& & \texttt{IF } \nu_1 \texttt{ THEN } \varepsilon(e_2, e_f, \beta, \nu) \texttt{ ELSE } \varepsilon(e_3, e_f, \beta, \nu); \\
& rec(e_1): & \exists \nu_1 \in \mathcal{V}al : \varepsilon(e_1, e_f, \beta, \nu_1) \wedge \\
& & \varepsilon(e_f, e_f, \beta', \nu), \text{ onde } \beta'(vr) = \nu_1
\end{array}
$$

$$\varepsilon(op1_0(fibonacci), fibonacci, vr \mapsto 1, 0)$$

O predicado:

$$\varepsilon(e, e_f, \beta, \nu)$$

foi especificado como:

$$\texttt{semantic\_rel\_expr}\overbrace{(\mathit{eval\_bool}, \mathit{eval\_op1}, \mathit{eval\_op2})}^{\mathfrak{I}}$$

$$(\texttt{expr}, \texttt{function}, \texttt{beta}, \texttt{val})$$

- Avalia-se a expressão `expr` considerando a função recursiva como a expressão `function`, `beta` como argumento e o retorno deve ser `val`.
- Se `function` é expandida indefinidamente, `semantic_rel_expr` também será e não existirá valor de retorno.

Uma das definições de terminação semântica:

$$T_\varepsilon(e_f) := \forall\beta\ \exists\nu\ \varepsilon(e_f, e_f, \beta, \nu)$$

$$
\begin{array}{lll}
\chi(m, e, e_f, \beta) & := & \texttt{IF } m = 0 \texttt{ THEN } \diamondsuit \\
& & \texttt{ELSE CASES } e \texttt{ OF} \\
& & \qquad cnst: \quad cnst^{\mathfrak{I}}; \\
& & \qquad vr: \quad \beta(vr); \\
& & \qquad op1(e_1): \quad \texttt{IF } \chi(m, e_1, e_f, \beta) \neq \diamondsuit \\
& & \qquad\qquad \texttt{THEN } op1^{\mathfrak{I}}(\chi(m, e_1, e_f, \beta)) \\
& & \qquad\qquad \texttt{ELSE } \diamondsuit; \\
& & \qquad op2(e_1, e_2): \quad \texttt{IF } \chi(m, e_1, e_f, \beta) \neq \diamondsuit \wedge \chi(m, e_2, e_f, \beta) \neq \diamondsuit \\
& & \qquad\qquad \texttt{THEN } op2^{\mathfrak{I}}(\chi(m, e_1, e_f, \beta), \chi(m, e_2, e_f, \beta)) \\
& & \qquad\qquad \texttt{ELSE } \diamondsuit; \\
& & \qquad ite(e_1, e_2, e_3): \quad \texttt{IF } \chi(m, e_1, e_f, \beta) \neq \diamondsuit \texttt{ THEN} \\
& & \qquad\qquad \texttt{IF } \chi(m, e_1, e_f, \beta) \texttt{ THEN } \chi(m, e_2, e_f, \beta) \\
& & \qquad\qquad \texttt{ELSE } \chi(m, e_3, e_f, \beta); \\
& & \qquad\qquad \texttt{ELSE } \diamondsuit; \\
& & \qquad rec(e_1): \quad \texttt{IF } \chi(m, e_1, e_f, \beta) \neq \diamondsuit \texttt{ THEN} \\
& & \qquad\qquad \chi(m-1, e_f, e_f, \beta'), \\
& & \qquad\qquad \text{onde } \beta'(vr) = \chi(m, e_1, e_f, \beta) \\
& & \qquad\qquad \texttt{ELSE } \diamondsuit
\end{array}
$$

$$\chi(4, op1_0(fibonacci), fibonacci, vr \mapsto 4) = 4$$

$$\chi(2, op1_0(fibonacci), fibonacci, vr \mapsto 4) = \diamondsuit$$

A função:

$$\chi(\mathit{depth}, e, e_f, \beta)$$

foi especificada como:

$$\texttt{eval\_expr}\overbrace{(\mathit{eval\_bool}, \mathit{eval\_op1}, \mathit{eval\_op2})}^{\mathfrak{I}}$$

$$(\texttt{depth}, \texttt{expr}, \texttt{function}, \texttt{beta})$$

- Avalia-se a a expressão `expr` considerando como função recursiva `function`, `beta` como argumento e `depth` como o limite de profundidade de chamadas recursivas.
- Se o número de chamadas ultrapassa o limite, então retorna-se `None` ($\diamondsuit$), caso contrário, o valor `value` em `Some(value)`.

$$T_\chi(e_f) := \forall\beta\ \exists m\ \texttt{Seja}\ \nu = \chi(m, e_f, e_f, \beta)\ \texttt{em}\ \nu \neq \diamondsuit \wedge \varepsilon(e_f, e_f, \beta, \nu)$$

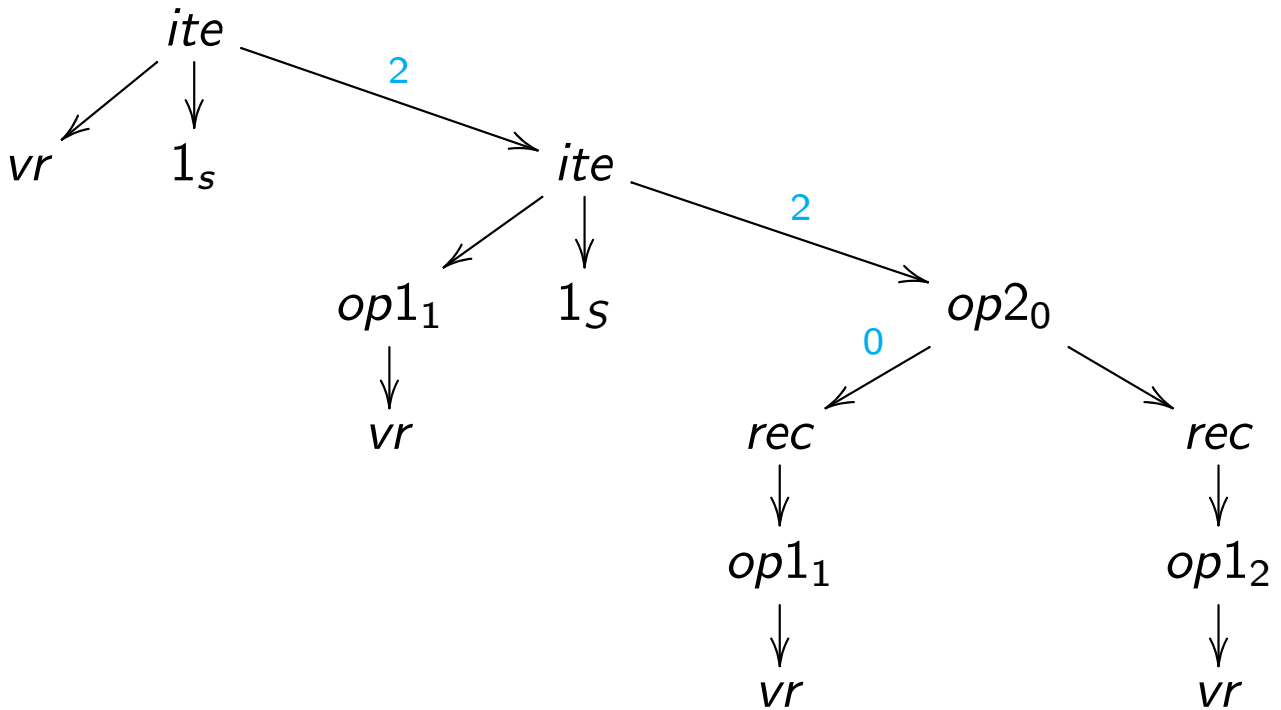
TCC Terminação

- Um contexto de chamado (*Calling Context*) é um registro que representa quais condições precisam ser validadas para se chegar a um chamado recursivo.
- Um contexto é válido quando contém:
  - Uma subexpressão recursiva.
  - A lista condições do caminho da linha de execução da subexpressão.
  - O caminho da linha de execução até a subexpressão.

```
PVS0_CC: TYPE = [# rec_expr:  (rec?)  % PVS0 recursive
call:  R(...)
, cnds:  Conditions % Path conditions
, path:  list[nat] #] % Position of rec_expr
```

$$cc : \langle path : [\mathbb{N}],\ rec\_expr : \{a : \mathcal{E}xpr | a = rec(b)\},\ cnds : [\mathcal{E}xprbool]\rangle$$

Exemplo | caminhos em Fibonacci (PVS0)



$$\begin{pmatrix} path := [022], \\ rec\_expr := rec(op1_1(vr)), \\ cnds := [exprnot(op1_1(vr)), exprnot(vr)] \end{pmatrix}$$

Dados:

- Um espaço de medida $\mathcal{MT}$.
- Uma relação bem-fundada $\prec$ sobre $\mathcal{MT}$.
- Uma função de medida $\mu : \mathcal{V}al \to \mathcal{MT}$.

Define-se TCC Terminação como:

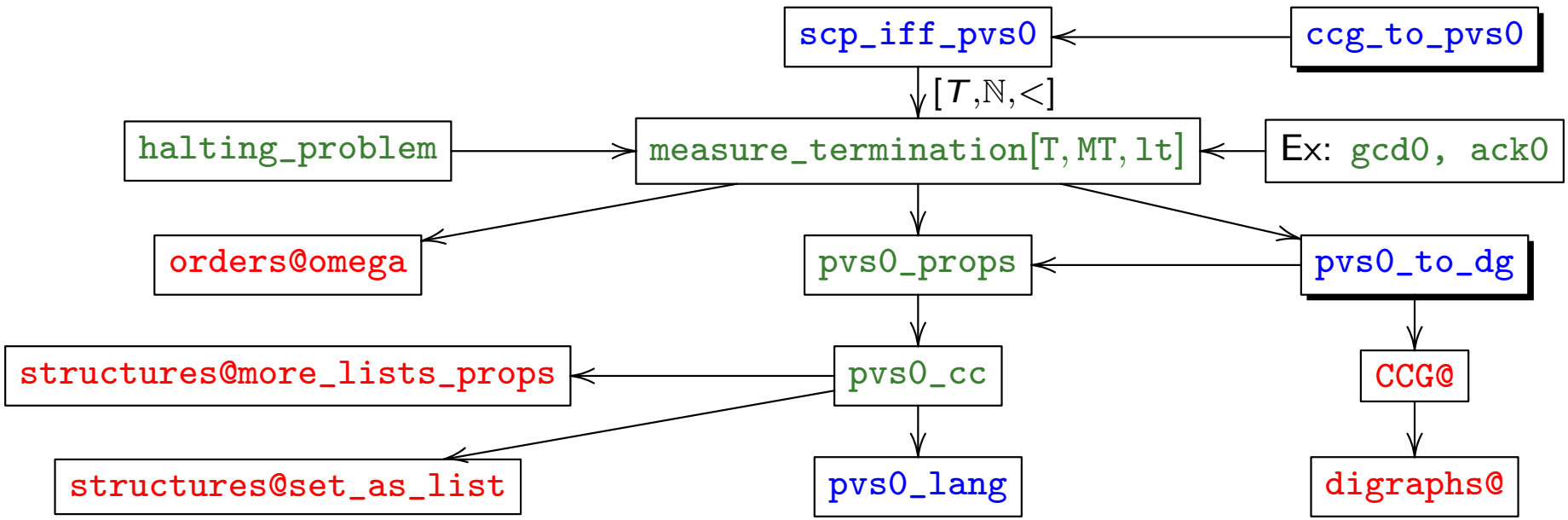
$$\begin{aligned} T_\zeta(e_f) \quad := \quad & \exists\, \mu,\ \forall\, (\beta,\ cc : Valid\_cc(e_f),\ \nu) : \\ & \varepsilon(get\_arg(rec\_expr(cc)), e_f, \beta, \nu) \wedge \\ & EC(e_f, cnds(cc), \beta) \Longrightarrow \mu(\nu) \prec \mu(\beta(vr)) \end{aligned}$$

Ou seja

$$\begin{array}{ccc} f(M) & \overset{\text{chama}}{\longrightarrow} & f(m) \\ \downarrow & & \downarrow \\ \mu(M) & \succ & \mu(m) \end{array}$$

Onde $\succ$ é converso de $\prec$

PVS0: As Teorias



- 233 propriedades formalizadas
- Há outras bibliotecas de terminação:
  - CCG
  - MWG

Equivalências Formalizadas

$$\boxed{\texttt{MWG(def)} \iff \texttt{CCG(def)}}$$

$$\iff$$

$$\boxed{T_{\varepsilon}\texttt{(def)} \iff T_{\zeta}\texttt{(def)} \iff T_{\chi}\texttt{(def)}}$$

Prova de equivalência | definições semânticas

Provou-se a equivalência entre:

$$T_{\varepsilon}(e_f) \iff T_{\chi}(e_f)$$

Ou seja:

$$\forall \beta \; \exists \nu \; \varepsilon(e_f, e_f, \beta, \nu)$$

$$\iff$$

$$\forall \beta \; \exists m \; \texttt{Seja} \; \nu = \chi(m, e_f, e_f, \beta) \; \texttt{em} \; \nu \neq \diamondsuit \wedge \varepsilon(e_f, e_f, \beta, \nu)$$

Para isso, mostraram-se os fatos:

$$\underbrace{\left\{\begin{array}{rl} \text{def. de } \chi & \text{Para todo } m \text{ maior ou igual a um } n \text{ qualquer e } e_f = rec(x_1): \\ & \chi(n, e, e_f, \beta) \neq \diamondsuit \ \wedge \ \chi(n, e, e_f, \beta) = \nu \ \Rightarrow \ \chi(m, e, e_f, \beta) = \nu \\ \text{Ind. em } e & \varepsilon(e, e_f, \beta, \nu) \Rightarrow \exists\, m \in \mathbb{N}: \ \chi(m, e, e_f, \beta) \neq \diamondsuit \ \wedge \ \chi(m, e, e_f, \beta) = \nu \\ \text{conseq.} & \varepsilon(e, e_f, \beta, \nu_1) \ \wedge \ \varepsilon(e, e_f, \beta, \nu_2) \ \Rightarrow \ \nu_1 = \nu_2 \end{array}\right.}$$

Para provar $T_\varepsilon(e_f) \Longleftrightarrow T_\chi(e_f)$

$T_\varepsilon(e_f) \Longrightarrow T_\chi(e_f)$

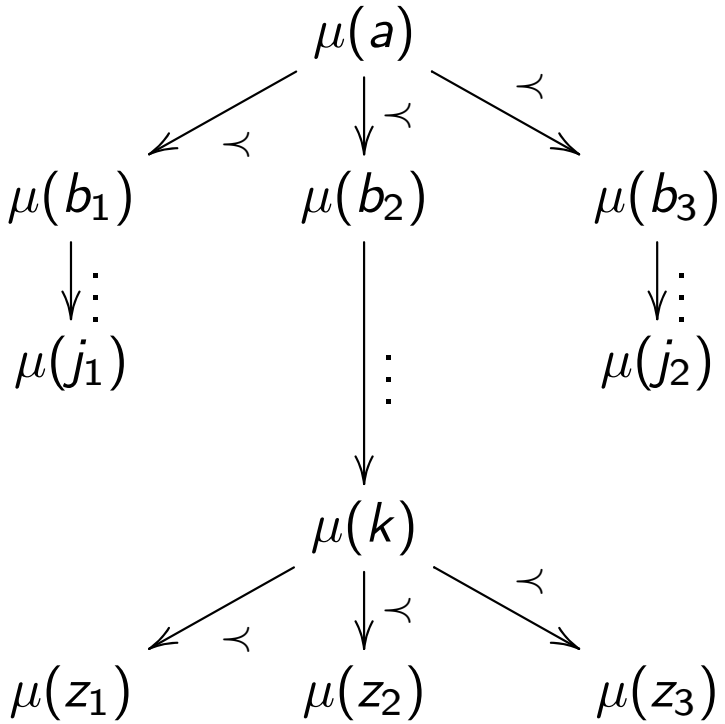
- $\forall \beta \; \exists \nu \; \varepsilon(e_f, e_f, \beta, \nu)$
- Por
  $\varepsilon(e, e_f, \beta, \nu) \Rightarrow \exists \; m \in \mathbb{N} : \; \chi(m, e, e_f, \beta) \neq \diamondsuit \; \wedge \; \chi(m, e, e_f, \beta) = \nu$
- Chega-se a
  $\forall \beta \; \exists m \; \texttt{Seja} \; \nu = \chi(m, e_f, e_f, \beta) \; \texttt{em} \; \nu \neq \diamondsuit \wedge \varepsilon(e_f, e_f, \beta, \nu)$

$T_\varepsilon(e_f) \Longleftarrow T_\chi(e_f)$

- $\forall \beta \; \exists m \; \texttt{Seja} \; \nu = \chi(m, e_f, e_f, \beta) \; \texttt{em} \; \nu \neq \diamondsuit \wedge \varepsilon(e_f, e_f, \beta, \nu)$
- $\forall \beta \; \exists \nu \; \varepsilon(e_f, e_f, \beta, \nu)$

Equivalência com TCC Terminação

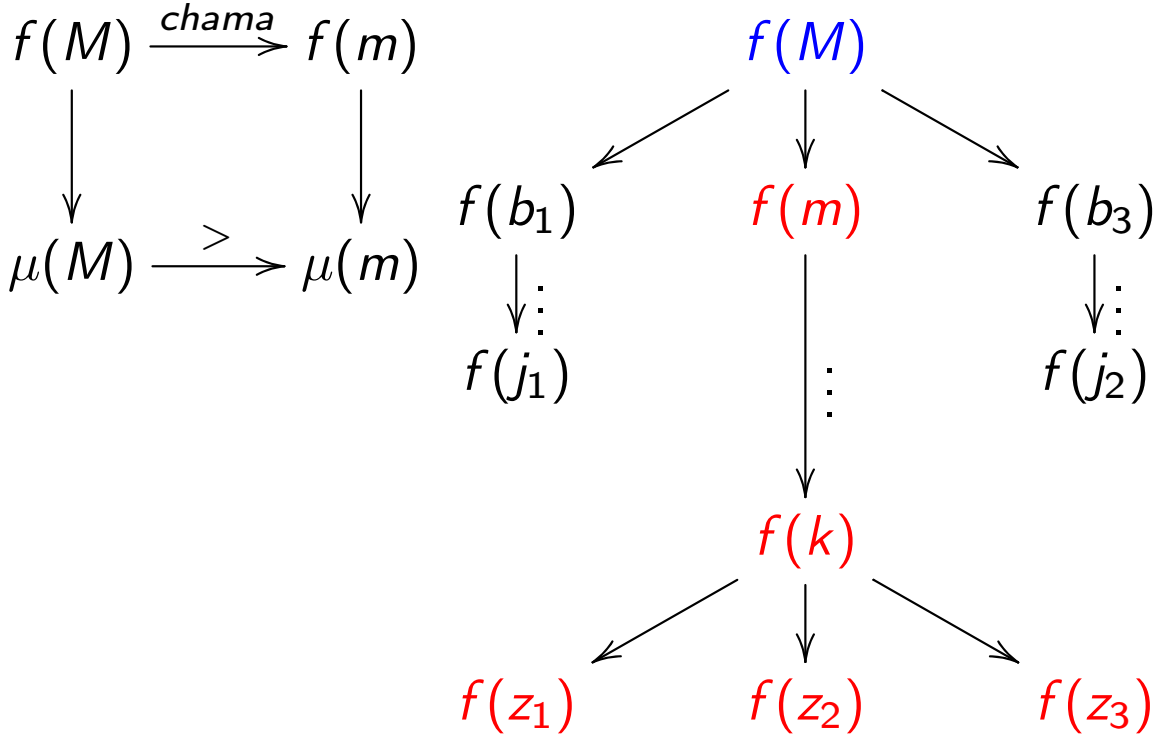
$T_\zeta(e_f) \Longrightarrow T_\chi(e_f)$
Uso do lema de König:



Vale $\chi(h, e_f, e_f, \beta)$, onde h é altura e $\beta = vr \mapsto a$

$T_\zeta(e_f) \Longleftarrow T_\chi(e_f)$

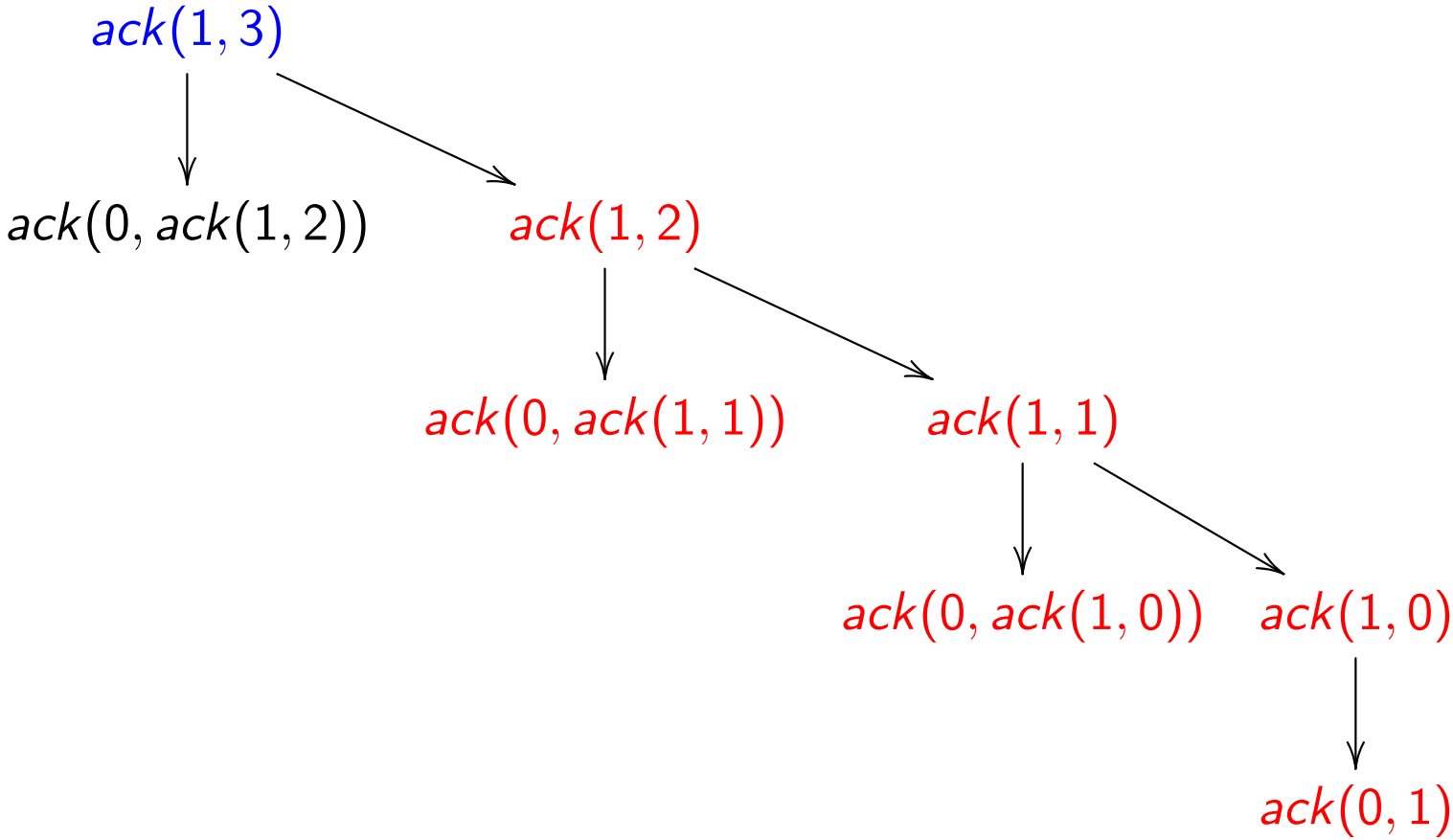
- $\forall\beta\ \exists m\ \chi(m, e_f, e_f, \beta) \neq \diamondsuit$
- $\mu(a) := min\{m \mid \chi(m, e_f, e_f, vr \mapsto a) \neq \diamondsuit\}$

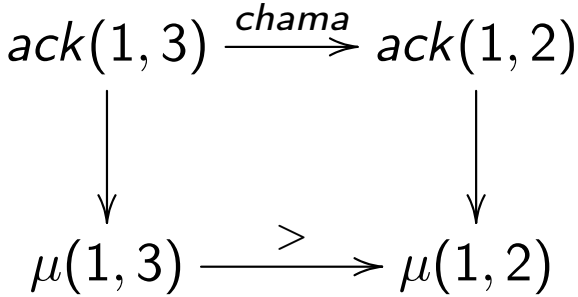


- $\mathcal{MT} := \mathbb{N}$ e $\prec :=<$

Exemplo | altura μ

$$
\begin{array}{l}
gcd(64,24) \\
\downarrow \\
gcd(24,64) \\
\downarrow \\
gcd(24,40) \\
\downarrow \\
gcd(24,16) \\
\downarrow \\
gcd(16,24) \\
\downarrow \\
gcd(16,8) \\
\downarrow \\
gcd(8,16) \\
\downarrow \\
gcd(8,8) \\
\downarrow \\
gcd(8,0)
\end{array}
\qquad
\begin{array}{ccc}
gcd(64,24) & \xrightarrow{chama} & gcd(24,64) \\
\downarrow & & \downarrow \\
\mu(64,24) & \xrightarrow{>} & \mu(24,64)
\end{array}
$$


ack(1, 3)
ack(0, ack(1, 2))
ack(1, 2)
ack(0, ack(1, 1))
ack(1, 1)
ack(0, ack(1, 0))
ack(1, 0)
ack(0, 1)

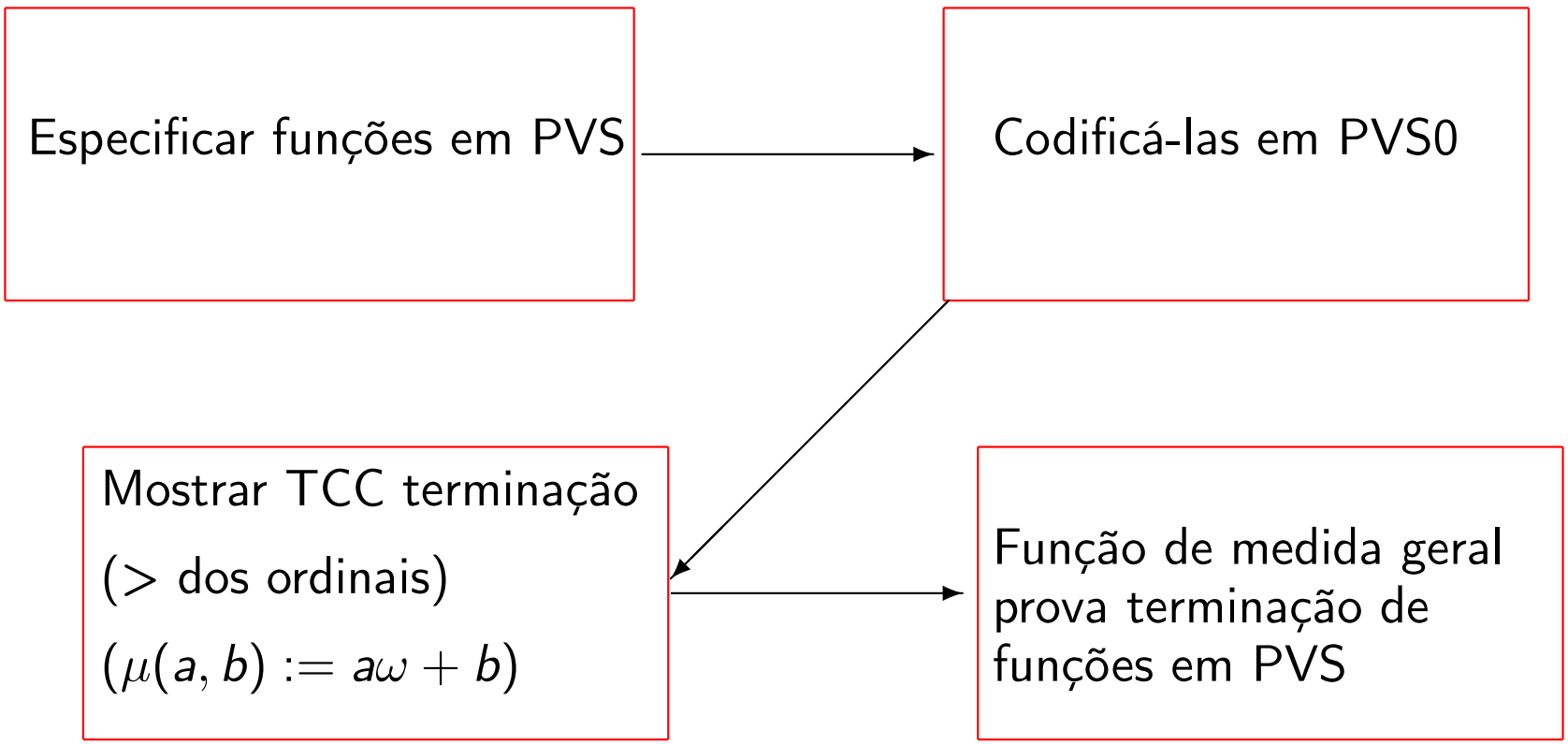

ack(1, 3)
chama
ack(1, 2)
μ(1, 3)
>
μ(1, 2)

Uso da teoria

Dois exemplos de uso da teoria:

- Uso das definições PVS0 para provar terminação de funções em PVS:
  - gcd.
  - Ackermann.
- Formalização da indecidiblidade do problema da parada.

Metodologia para provar terminação de funções



Especificar a função

```
ack(m,n:nat) :  RECURSIVE {a :  nat | a = ack_pvs0(m,n)} =
   IF m = 0 THEN n+1
   ELSIF n = 0 THEN ack(m-1,1)
   ELSE ack(m-1,ack(m,n-1))
ENDIF
```

Codificar em PVS0

```
pvs0_ack :  Def = def(ite(op1(0,vr),
   op1(2,vr),
   ite(op1(1,vr),
      rec(op1(3,vr)),
      rec(op2(0,vr,rec(op1(4,vr)))))))
```

Provar terminação sobre PVS0 e encontrar uma função de medida

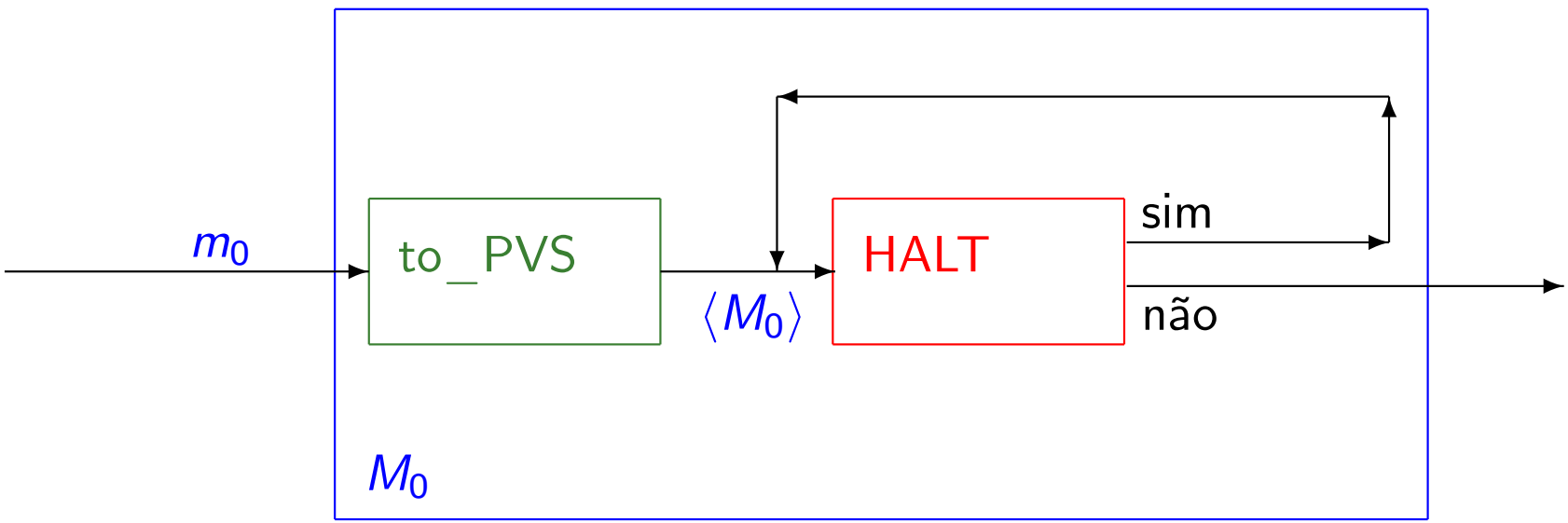
```
R : MACRO PRED[[MT,MT]] = ordinals.<

pvs0_ack_tcc_termination:  JUDGEMENT
   pvs0_ack HAS_TYPE
(pvs0_tcc_termination(ebool,eop1,eop2))

ack_wfm :
(pvs0_tcc_termination_pred(ebool,eop1,eop2)(pvs0_ack))
```

Provar terminação da função especificada

```
ack(m,n:nat) :  RECURSIVE {a :  nat | a = ack_pvs0(m,n)} =
   IF m = 0 THEN n+1
   ELSIF n = 0 THEN ack(m-1,1)
   ELSE ack(m-1,ack(m,n-1))
ENDIF
MEASURE ack_wfm BY R
```

Indecibilidade do problema da parada

- Assumindo to_PVS : $\mathcal{V}al \mapsto \mathcal{E}xpr$ é sobrejetiva.
- Supondo que existe a expressão PVS0 HALT : $\mathcal{E}xpr \mapsto \mathcal{B}ool$ que determina terminação de outras expressões PVS0.
- to_PVS(m_0) = M_0



- Exemplo de uso de TCC Terminação

Proposta

- Extender PVS0 para PVS1.
- Recursos:
  - suporte a vários tipos
  - operadores com mais argumentos
- Adaptar definições de terminação.
- Mostrar equivalência usando transformação conservativa.

$$
\begin{aligned}
\mathcal{E}xpr \quad ::= \quad & Type\ rec(Type\ \mathcal{E}xpr, \cdots, Type\ \mathcal{E}xpr) \\
& \mid Type\ op(Type\ \mathcal{E}xpr, \cdots, Type\ \mathcal{E}xpr) \\
& \mid Type\ ite(\mathcal{E}xpr, \mathcal{E}xpr, \mathcal{E}xpr) \\
& \mid Type\ vr \\
& \mid Type\ cnst
\end{aligned}
$$

- Mostrar que PVS0 é Turing completa.
- Mostrar a decidibilidade do problema da parada para tipos finitos.

Referências I

[1] Alan M. Turing.
On computable numbers, with an application to the Entscheidungsproblem.
Proceedings of the London Mathematical Society, 2(42):230–265, 1936.

[2] Byron Cook, Andreas Podelski, and Andrey Rybalchenko.
Proving program termination.
Commun. ACM, 54(5):88–98, May 2011.

[3] Alan M. Turing.
The early British computer conferences.
chapter Checking a Large Routine, pages 70–72. MIT Press, Cambridge, MA, USA, 1989.

Referências II

[4] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram.
The size-change principle for program termination.
In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, pages 81–92, New York, NY, USA, 2001. ACM.

[5] Panagiotis Manolios and Daron Vroon.
Termination analysis with calling context graphs.
In Thomas Ball and RobertB. Jones, editors, *Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 401–414. Springer Berlin Heidelberg, 2006.

[6] Andréia B. Avelar.
Formalização da automação da terminação através de grafos com matrizes de medida.
PhD thesis, Universidade de Brasília, Departamento de Ciência Da Computação e Matemática, Brasília, Distrito Federal, Brasil, 2015.