Formalizando uma linguagem simples

Introdução ao assistente de provas Coq

Rodrigo Ribeiro

Formalizando uma linguagem de expressões

- Definição da sintaxe.
- Definição da semântica.
  - Semântica de passo pequeno
  - Semântica de passo grande
- Sistema de tipos
  - Algoritmo para verificação de tipos.
- Propriedades

Sintaxe

- Definida pela seguinte gramática

$$
\begin{array}{rcl}
e & ::= & \mathit{zero} \\
  & \mid & \mathit{true} \\
  & \mid & \mathit{false} \\
  & \mid & \mathit{suc}\ \ e \\
  & \mid & \mathit{pred}\ \ e \\
  & \mid & \mathit{isZero}\ \ e \\
  & \mid & \mathit{if}\ \ e\ \ \mathit{then}\ \ e\ \ \mathit{else}\ \ e
\end{array}
$$

Sintaxe como tipo indutivo

```
Inductive exp : Set :=
| Zero   : exp
| Succ   : exp -> exp
| T      : exp
| F      : exp
| Pred   : exp -> exp
| If     : exp -> exp -> exp -> exp
| IsZero : exp -> exp.
```

Valores

- Primeiro definimos os resultados da computação, isto é os valores da linguagem.

$$
\begin{array}{rcl}
b & ::= & \mathit{true} \mid \mathit{false} \\
n & ::= & \mathit{zero} \mid \mathit{suc}\ \ n \\
v & ::= & b \mid n
\end{array}
$$

Valores como predicados em Coq

```
Inductive bvalue : exp -> Prop :=
| btrue  : bvalue T
| bfalse : bvalue F.

Inductive nvalue : exp -> Prop :=
| nzero  : nvalue Zero
| nsucc  : forall n, nvalue n ->
 nvalue (Succ n).

Inductive value : exp -> Prop :=
| Bvalue : forall e, bvalue e ->
value e
| Nvalue : forall e, nvalue e ->
value e.
```

Semântica operacional

- Especifica a execução de uma linguagem como regras de transição entre estados.
  - Passo pequeno: Cada regra mostra como uma expressão reduz em um passo.
  - Passo grande: Cada regra mostra como uma expressão reduz até o valor correspondente.
- Especificamos a semântica como predicados Coq.

Semântica de passo pequeno — 1

```
Reserved Notation "e '==>' e1" (at level 40).

Inductive step : exp -> exp -> Prop :=
| ST_If_T
  : forall e e', (If T e e') ==> e
| ST_If_F
  : forall e e', If F e e' ==> e'
| ST_If
  : forall e e' e1 e2,
    e ==> e'                ->
    If e e1 e2 ==> If e' e1 e2
```

Semântica de passo pequeno — 2

```
| ST_Succ
  : forall e e',
    e ==> e'                    ->
    (Succ e) ==> (Succ e')
| ST_Pred_Zero
  : Pred Zero ==> Zero
| ST_Pred_Succ
  : forall e,
    nvalue e          ->
    Pred (Succ e) ==> e
| ST_Pred
  : forall e e',
    e ==> e'              ->
    (Pred e) ==> (Pred e')
```

Semântica de passo pequeno — 3

```
| ST_IsZeroZero
  : IsZero Zero ==> T
| ST_IsZeroSucc
  : forall e,
    nvalue e            ->
    IsZero (Succ e) ==> F
| ST_IsZero
  : forall e e',
    e ==> e'                  ->
    (IsZero e) ==> (IsZero e')
```

Forma normal

```
Definition normal_form e :=
  ~ exists e', step e e'.

Definition stuck e :=
  normal_form e /\ ~ value e.
```

Relacionando formas normais e valores

```
Lemma value_is_nf'
    : forall e, value e -> normal_form e.
Proof.
  intros e Hv.
  unfold normal_form.
  intro contra.
  induction e.
  +
    inverts contra.
    inverts H.
  +
    inverts contra.
    inverts Hv.
    ....
```

Relacionando formas normais e valores

- A prova anterior é tediosa...
  - Repetição de táticas...
- Solução usar automação de provas para tratar casos repetitivos.

Uma primeira tática

```
Ltac s :=
match goal with
| [ H : ex _ |- _ ] => destruct H
| [ H : Zero ==> _ |- _] => inverts H
| [ H : T ==> _ |- _] => inverts H
| [ H : F ==> _ |- _] => inverts H
| [ H : value (Pred _) |- _] => inverts H
| [ H : bvalue (Pred _) |- _] => inverts H
| [ H : nvalue (Pred _) |- _] => inverts H
... lots of boring cases
end.
```

Usando automação

```
Lemma value_is_nf
: forall e, value e -> normal_form e.
Proof.
    unfold normal_form ; intros e H contra ;
    induction e ; try (repeat s) ; eauto.
Qed.
```

Determinismo da semântica

```
Lemma step_deterministic
: forall e e', e ==> e' ->
forall e'', e ==> e'' ->
e' = e''.
Proof.
    intros e e' H ; induction H ;
   intros e'' H' ;
   inverts H' ; f_equal ;
   try repeat s ;
   auto ; try repeat s1.
Qed.
```

Semântica de passo grande — 1

```
Reserved Notation "e '==>>' e1" (at level 40).

Inductive big_step : exp -> exp -> Prop :=
| B_Value
  : forall v, value v -> v ==>> v
| B_If_True
  : forall e e1 e11 e2,
    e ==>> T ->
    e1 ==>> e11 ->
    (If e e1 e2) ==>> e11
| B_If_False
  : forall e e1 e2 e22,
    e ==>> F ->
    e2 ==>> e22 ->
    (If e e1 e2) ==>> e22
```

Semântica de passo grande — 2

```
| B_Succ
  : forall e nv,
    nvalue nv ->
    e ==>> nv ->
    (Succ e) ==>> (Succ nv)
| B_PredZero
  : forall e,
    e ==>> Zero ->
    (Pred e) ==>> Zero
| B_PredSucc
  : forall e nv,
    nvalue nv ->
    e ==>> (Succ nv) ->
    Pred e ==>> nv
```

Semântica de passo grande — 3

```
| B_IsZeroZero
  : forall e,
    e ==>> Zero ->
    (IsZero e) ==>> T
| B_IsZeroSucc
  : forall e nv,
    nvalue nv ->
    e ==>> (Succ nv) ->
    (IsZero e) ==>> F
```

Sistema de tipos — 1

```
Inductive has_type : exp -> type -> Prop :=
| T_True
  : T <<- TBool
| T_False
  : F <<- TBool
| T_Zero
  : Zero <<- TNat
| T_Succ
  : forall e,
    e <<- TNat ->
    (Succ e) <<- TNat
```

Sistema de tipos — 2

```
| T_Pred
  : forall e,
    e <<- TNat  ->
    (Pred e) <<- TNat
| T_If
  : forall e e' e'' t,
    e <<- TBool ->
    e' <<- t    ->
    e'' <<- t   ->
    (If e e' e'') <<- t
| T_IsZero
  : forall e,
    e <<- TNat ->
    (IsZero e) <<- TBool
```

Propriedades

```
Theorem progress : forall e t, e <<- t ->
    value e \/ exists e', e ==> e'.
Proof.
  induction 1 ; try solve [left ; auto] ; ...
Qed.

Theorem preservation : forall e t, e <<- t ->
     forall e', e ==> e' -> e' <<- t.
Proof.
  induction 1 ; intros ; repeat (s ; eauto) ; ...
Qed.
```

Definindo um type checker — 1

- Usar um tipo dependente para garantir correção e completude com respeito ao sistema de tipos.

```
typecheck : forall e, {t | e <<- t} + {forall t, ~ (e <<- t
```

Definindo um type checker — 2

```
refine (fix tc (e : exp) : {t | e <<- t} + {forall t, ~ (e
    match e as e' return e = e' -> {t | e' <<- t} + {forall
        | T  => fun _ => [|| TBool ||]
        | F  => fun _ => [|| TBool ||]
        | Zero  => fun _ => [|| TNat ||]
        | Succ e => fun _ =>
            ty <-- tc e ;
            eq_ty_dec ty TNat ;;;
            [|| TNat ||]
            ....
```

Definindo um type checker — 3

- Demais casos são similares e seguem a estrutura das regras do sistema de tipos.
- Obrigações de provas produzidas são resolvidas por uma tática.

Conclusão — 1

- Um “aperitivo” de como formalizar resultados utilizando Coq.
- Automação é fundamental para garantir manutenibilidade e escalabilidade de formalizações.
- Formalizem resultados de seus trabalhos usando Coq!

Conclusão — 2

- Para saber mais:
  - Bertot, Yves; Casterrán, Pierre - Interactive Theorem Proving and Program Development; The Coq'art - The calculus of inductive constructions
- Chlipala, Adam - Certified Programming with Dependent Types
- Pierce, Benjamin - Software Foundations.

Conclusão — 3

- Dúvidas?
  - Fiquem a vontade para me enviar e-mails!
- Outras fontes:
- Stackoverflow e #coq no #freenode

Conclusão — 4

Figure 1: Obrigado pela atenção!