Verificação de Programas

Nelma Moreira

Coq: demonstrador interactivo baseado no pCiC - breve introduo
Aula 23

Tipos dependentes

Generalizam os tipos $A \rightarrow B$ que correspondem a funes cujo argumento tem tipo A e os objectos tem tipo B.

Suponhamos o tipo $string(n)$ das strings de tamanho n. Este tipo depende de $n : int$ e pode ser considerado um predicado sobre o tipo int.

Podemos generalizar a outros predicados com mais argumentos e quantificar universalmente. No caso anterior, teramos:

$$\forall n : int, string(n)$$

que pode ser o tipo de uma funo que para qualquer inteiro n retorna uma string de tamanho n .

Produto dependente e Isomorfismo Curry-Howard

$\forall x : \tau, \sigma$ o tipo de uma funo que aplicada a objectos de tipo τ e retorna um objecto de tipo $\sigma[x/a]$ para todo $a : \tau$.

Se x no ocorre em σ, $\forall x : \tau, \sigma$ corresponde a $\tau \rightarrow \sigma$.

Na lgica intuicionista, uma demonstrao de $\forall x : A, Px$ um mtodo p que transforma cada $a \in X$ numa demonstrao de Pa. Isto :

$$\Pi_{x:X} Px = \{f : X \rightarrow \cup_{x:X} Px) \mid \forall a : X,\ fa : Pa\}$$

Exemplos de tipos dependentes

$\forall n : nat, n \leq n$

$\forall n, m : nat, n \leq m \rightarrow n \leq m + 1$

$\forall P, Q : \texttt{Prop},\ P\ \vee\ Q \rightarrow Q\ \vee\ P$

$nat \rightarrow nat \rightarrow \texttt{Prop}$

$\forall n, p : nat, bin\, n \rightarrow bin\, p \rightarrow bin\,(n + p)$

$\forall n : nat, list\, n$

$\forall A : Set, A \rightarrow list\, A \rightarrow list\, A$

$\forall A, B : Set, A \rightarrow B \rightarrow A * B$

$\forall A, B : Set, A * B \rightarrow A$

Inferncia de Tipos

Aplicao

$$\frac{E, \Gamma \vdash t_1 : \forall x : A, B \qquad\qquad E, \Gamma \vdash t_2 : A}{E, \Gamma \vdash t_1\, t_2 : B[x/t_2]}\ (APP)$$

Abstrao

$$\frac{E, \Gamma :: (x : A) \vdash t : B}{E, \Gamma \vdash \texttt{fun}\, x : A \Rightarrow t : \forall x : A, B}\ (ABS)$$

Se x no ocorre em B ento $\forall x : A, B$ equivalente a $A \rightarrow B$

Exemplos

```
Check le_n.
le_n : forall n : nat, n <= n

Check le_S.
le_S: forall n m : nat, n <= m -> n <= S m

Definition le_36_37 := le_S 36 36 (le_n 36).

Check (le_S _ _ (le_S _ _ (le_n 36))).

Theorem le_i_SSi : forall i:nat, i <= S (S i).
Proof (fun i:nat => le_S _ _ (le_S _ _ (le_n i))).
```

Argumentos implcitos

```
Definition compose : forall A B C : Set, (A->B)->(B->C)->A->C
   := fun A B C f g x => g (f x).
```

Podemos usar variveis annimas:

```
Check (fun (A:Set)(f:Z->A) => compose _ _ _ Z_of_nat f).
```

Ou definir logo com

```
Reset compose.
Set Implicit Arguments.
Definition compose (A B C:Set)(f:A->B)(g:B->C)(a:A):= g(f a).
```

Lgica em Coq: deduo natural para lgica de primeira ordem

As conectivas $\to$ e $\forall$ so intrnsecas ao sistema de tipos do Coq e portanto as regras de introduo e eliminao correspondem s regras de inferncia. As tcticas correspondentes so `intro` e `apply`.

```
Theorem imp_trans:forall P Q R:Prop,(P->Q)->(Q->R)->P ->R.
Proof.
intros P Q R H H0 p.
apply H0. apply H. assumption.
Qed.

Theorem all_imp_dist: forall (A:Type)(P Q:A->Prop),
(forall x:A, P x -> Q x)->(forall y:A, P y)->(forall z:A, Q z).
```

As restantes conectivas so definidas indutivamente (vamos ver mais tarde), mas podemos us-las sabendo para j quais os constructores e os seus tipos, e quais as tcticas correspondentes.

Deduo natural para LP

	Introduo	**Eliminao**
$\wedge$	$\dfrac{\phi \quad \psi}{\phi \wedge \psi} \wedge \mathbf{I}$	$\dfrac{\phi \wedge \psi}{\phi} \wedge \mathbf{E}_1 \qquad \dfrac{\phi \wedge \psi}{\psi} \wedge \mathbf{E}_2$
$\vee$	$\dfrac{\phi}{\phi \vee \psi} \vee \mathbf{I}_1 \qquad \dfrac{\psi}{\phi \vee \psi} \vee \mathbf{I}_2$	$\dfrac{\phi \vee \psi \qquad \begin{array}{c}[\phi]\\ \vdots\\ \gamma\end{array} \quad \begin{array}{c}[\psi]\\ \vdots\\ \gamma\end{array}}{\gamma} \vee \mathbf{E}$
F	$\dfrac{\begin{array}{c}[\phi]\\ \vdots\\ \neg\phi\end{array}}{\mathrm{F}} \mathrm{F}\mathbf{I}$	$\dfrac{\mathrm{F}}{\phi}$
$\rightarrow$	$\dfrac{\begin{array}{c}[\phi]\\ \vdots\\ \psi\end{array}}{\phi \rightarrow \psi} \rightarrow\mathbf{I}$	$\dfrac{\phi \quad \phi \rightarrow \psi}{\psi} \rightarrow\mathbf{E}$
$=$	$\dfrac{}{t=t} =\mathbf{I}$	$\dfrac{t_1=t_2 \qquad \phi[t_1/x]}{\phi[t_2/x]} =\mathbf{E}$ e x substituvel por t_1 e por t_2 em ϕ
$\forall$	$\dfrac{\begin{array}{l}[v]\\ \qquad \vdots\\ \qquad \phi[v/x]\end{array}}{\forall x \ \phi} \forall\mathbf{I}$ onde v uma varivel nova (no ocorre antes)	$\dfrac{\forall x \ \phi}{\phi[t/x]} \ \forall\mathbf{E}$ onde x substituvel por t em ϕ
$\exists$	$\dfrac{\phi[t/x]}{\exists x \ \phi} \ \exists\mathbf{I}$ onde x substituvel por t em ϕ	$\dfrac{\exists x \ \phi \qquad \begin{array}{c}[v \quad \phi[v/x]]\\ \vdots\\ \psi\end{array}}{\psi} \ \exists\mathbf{E}$ onde v uma varivel nova que no ocorre antes nem em ψ

Conectivas lgicas

$\wedge$:

I: `conj` $\forall A\, B : Prop,\ A \rightarrow B \rightarrow A \wedge B$ (`split`)

E: `and_ind` $\forall A\, B\, P : Prop, (A \rightarrow B \rightarrow P) \rightarrow A \wedge B \rightarrow P$ (`elim`)

$\vee$:

I: `or_introl` $\forall A\, B : Prop, A \rightarrow A \ \vee \ B$ (`left`) `or_intror` $\forall A B : Prop, B \to A \lor B$ (`right`)

E: `or_ind` $\forall A\, B\, P : Prop, (A \to P) \to (B \to P) \to A \lor B \to P$ (`elim`)

Conectivas lgicas

F `False`

E: $\forall P : Prop, False \to P$ (`False_ind`)

$\neg$ $\neg P$ definido por $P \to \mathrm{F}$

E: (`elim`)

Existencial

$\exists$ `ex`: $\forall A : Type(A \to Prop) \to Prop$

`ex P` $\exists x.Px$

I: `ex_intro` $\forall (A : Type)(P : A \to Prop)(x : A)\,, Px \to ex\, P$

(`exists` v)

E: `ex_ind` $\forall A\, P\, P0\; : (\forall x : A,\; P\, x \to P0) \to ex\, P \to P0$

(`elim`)

```
Lemma ex_imp_ex :
 forall(A:Type)(P Q:A->Prop)(ex P)->
           (forall x:A,Px->Qx)->(ex Q).
```

Igualdade

$=$ `eq`: $\forall (A : Type),\; A \to A \to Prop$

I: `refl_equal` $\forall (A : Type)(x : A),\; x = x$ (`reflexivity`)

E: `eq_ind` (`rewrite`)

$$\forall (A : Type)(x : A)(P : A \to Prop), Px \to \forall (y : A), x = y \to Py$$

Se e um termo de tipo $\forall (x_1 : T_1) \cdots (x_n : T_n), a = b$ e o objectivo da forma $P\, a$ a tctica `rewrite` e, gera um sub-objectivo da forma $P\, b$.

```
Theorem eq_sym' : forall (A:Type)(a b:A), a=b->b=a.
```

Variantes No `rewrite` pode-se orientar a igualdade: `rewrite <-` e ou `rewrite ->` e. Pode-se escolher a hiptese, etc...

Tcticas

Cada conectiva lgica tratada por dois gneros de tcticas, uma para uso em hipteses - tcticas de eliminao - e outras para uso em objectivos - tcticas de introduo.

	$\rightarrow$	$\forall$	$\wedge$	$\vee$	$\exists$
Hipteses	`apply`	`apply`	`elim`	`elim`	`elim`
Objectivos	`intros`	`intros`	`split`	`left` ou `right`	`exists` v
	$\neg$	$=$			
Hipteses	`elim`	`rewrite`			
Objectivos	`intro`	`reflexivity`			

onde:

`split` equivalente a `intros; apply conj`

`left` equivalente a `intros; apply or_introl`

analogo para `right`.

Tipos Indutivos

So definidos por

- um nome
- um tipo (ou famlia de tipos)
- construtores
- processo de computao: por casos ou recurso
- princpios de induo

Pattern-matching permite a descrio de funes por casos

Tipos Indutivos

- Enumeraes

```
Inductive dias: Set := Seg: dias | Ter:dias | Qua:dias
           Qui: dias | Sex: dias | Sab: dias | Dom: dias.

Check dias_ind.
```

- Records

```
Inductive plane: Set := point : Z -> Z -> plane.

Record plane: Set := point { abs: Z; ord:Z}.
```

Tipos Indutivos

- Records com variantes

```
Inductive vehicle : Set :=
  | bicycle : nat -> vehicle
  | motorized : nat -> nat -> vehicle.
```

- Recursivos:

```
Inductive nat : Set := O: nat | S: nat ->nat.
```

- Polimrficos:

```
Inductive list (A : Type) : Type :=
    nil : list A | cons : A -> list A -> list A
```

Tcticas

`pattern m` $\Rightarrow$ $(\lambda x.P(x))m$

case Se t tem um tipo indutivo , `case` t substitui todas as instncias de t no objectivo com todos os casos possveis.

destruct Anloga, mas apenas para tipos no recursivos

apply `apply T_ind`

elim A tctica `elim` faz a ligao entre o tipo indutivo e o princpio indutivo correspondente. Se t um termo de tipo T, de acordo com a espcie s do objectivo `elim t`, escolhe o princpio, `T_ind`, `T_rec` ou `T_rect`. Todos estes princpios tm uma varivel quantificada universalmente P de tipo `T ->s` e terminam com em $\forall x : T.(Px)$. Equivale a

`pattern m; apply T_ind`

`match`

Construir funes com anlise de casos.

```
Definition month_length (leap:bool)(m:month) : nat :=
  match m with
  | January => 31 | February => if leap then 29 else 28
  | March => 31   | April => 30    | May => 31  | June => 30
  | July => 31    | August => 31   | September => 30
  | October => 31 | November => 30 | December => 31
  end.
```

```
Definition nb_wheels (v:vehicle) : nat :=
  match v with
  | bicycle x => 2
  | motorized x n => n
  end.
```

Mais Tcticas

`induction` `induction t` permite que t no esteja no contexto: `intros until t; elim t` seguida de `intros`...

`simpl` realiza redues $\beta, \iota, ...$

`change` permite substituir um objectivo por outro convertvel

`discriminate` e `injection`

Para lidar com termos que se pretendem ou no equivalentes

Relaciona igualdade com os tipos indutivos:

- os dois construtores no so iguais e
- os construtores so injectivos

No podem ser usados com termos da classe `Prop`.

`rewrite`

Se t do tipo $\forall (x_i : T_i)_{i=1..n}, a = b$ e o objectivo da forma Pa, a tctica `rewrite t` dar um sub-objectivo Pb. Pode-se indicar a direco em que a reescrita aplicada.

Tipos Recursivos

Funes recursivas

$$\texttt{Fixpoint}\ f(x_1 : T_1) : T = expr$$

Mais geral

$$\texttt{Fixpoint}\ f(x_1 : T_1)...(x_n : T_n)\ \{\texttt{struct}\ x_i\} : T = expr$$

mas necessrio que a recurso termine!

```
Fixpoint mult2 (n:nat) : nat :=
   match n with
     0 => 0
   | S p => S (S (mult2 p))
   end.
```

Tipos Indutivos – `nat`

```
nat

Inductive nat : Set :=  O : nat | S : nat -> nat


Fixpoint plus (n m:nat) {struct n} : nat :=
  match n with
  | O => m
  | S p => S (p + m)
  end

Lemma plus_n_O : forall n:nat, n = n + 0.
Lemma plus_n_Sm: forall n m:nat, S n + m = S (n+m).

Fixpoint mult (n m:nat) {struct n} : nat :=
  match n with
  | O => 0
  | S p => m + mult (p  m)
  end
```

Relaes como tipos indutivos

```
Inductive le (n : nat) : nat -> Prop :=
    le_n : n <= n
  | le_S : forall m : nat, n <= m -> n <= S m
```

Tcticas: `constructor n.` Provar que

```
Theorem zz: 0<= 0.
Proof.
constructor 1.
Qed.

Theorem ut: 1<= 3.
Proof.
constructor 2.
constructor 2.
constructor 1.
Qed.
```

ou usar s `repeat constructor`.

Tipos Indutivos Polimrficos - `List`, `option`, `prod`

```
Inductive list (A:Type): Type := nil : list A
 | cons : A -> list A -> list A.
```

Exerc. 23.1. *1. Concatenao de listas*

2. Dada uma lista retornar uma lista com os 2 primeiros elementos

3. Dada uma lista e n retornar uma lista com os n primeiros elementos

4. Dada uma lista de inteiros retornar a sua soma.

5. Dado n retornar uma lista com os n primeiros inteiros.

◇

Tipos Polimrficos

```
Fixpoint app (A:Set)(l m:list A){struct l} : list A :=
  match l with
  | nil => m
  | cons a l1 => cons a (app A l1 m)
  end.
```

`option` –**funes parciais**

```
Inductive option (A : Type) : Type :=  Some : A -> option A
| None : option A
```

`prod` – **pares**

```
Inductive prod (A : Type) (B : Type) : Type :=
 pair : A -> B -> A * B
```

Tipos Indutivos Dependentes - `ltree`

rvores binrias

```
Inductive Z_btree : Set :=
Z_leaf: Z_btree | Z_bnode: Z-> Z_btree-> Z_btree ->Z_btree.
```

Exerc. 23.2.

- *Determinar a soma de todos os ns duma rvore binria*
- *Determinar se uma rvore binria tem algum zero.*

◇

rvore com ns menores que um dado n

```
Inductive ltree (n:nat) : Set:=
| lleaf : ltree n
| lnode : forall p:nat, p<= n -> ltree n -> ltree n -> ltree n.
```

Tipos Indutivos Dependentes - `htree`

rvores com ramos do mesmo comprimento

```
    Inductive htree (A:Set) : nat->Set :=
  | hleaf : A->htree A 0
  | hnode : forall n:nat, A -> htree A n -> htree A n
               -> htree A (S n).
```

```
Fixpoint htree_to_btree (n:nat)(t:htree Z n){struct t}:
Z_btree :=
  match t with
  | hleaf x => Z_bnode x Z_leaf Z_leaf
  | hnode p v t1 t2 =>
      Z_bnode v (htree_to_btree p t1)(htree_to_btree p t2)
  end.
```

Predicados Indutivos (propriedades)

Par

```
    Inductive even : nat->Prop :=
  | O_even : even 0
  | plus_2_even : forall n:nat, even n -> even (S (S n)).
```

Lista ordenada

```
Inductive sorted (A:Set)(R:A->A->Prop) : list A -> Prop :=
  | sorted0 : sorted A R nil
```

```
  | sorted1 : forall x:A, sorted A R (cons x nil)
  | sorted2 :
      forall (x y:A)(l:list A),
        R x y ->
        sorted A R (cons y l)-> sorted A R (cons x (cons y l)).
```

Construo de Predicados indutivos

- Os construtores so axiomas
- Os construtores devem corresponder a casos mutuamente exclusivos

Demonstraes por induo usando Predicados indutivos

```
Theorem sum_even : forall n p:nat, even n -> even p
               -> even (n+p).
Proof.
 intros n p Heven_n; elim Heven_n.
 trivial.
 intros x Heven_x Hrec Heven_p; simpl.
 apply plus_2_even; auto.
Qed.
```

```
Theorem lt_le : forall n p:nat, n < p -> n <= p.
Proof.
 intros n p H; elim H; repeat constructor; assumption.
Qed.
```

Conectivas lgicas

Verifica os princpios de induo `_ind`

True

```
Inductive True : Prop :=  I : True
```

False

```
Inductive False : Prop :=.
```

$\wedge$

```
Inductive and (A : Prop) (B : Prop) : Prop :=
conj : A -> B -> A /\ B
```

$\vee$

```
Inductive or (A : Prop) (B : Prop) : Prop :=
    or_introl : A -> A \/ B | or_intror : B -> A \/ B
```

Conectivas lgicas

Verifica os princpios de induo `_ind`

$\exists$

```
Inductive ex (A : Type) (P : A -> Prop) : Prop :=
    ex_intro : forall x : A, P x -> ex P
```

$=$

```
Inductive eq (A : Type) (x : A) : A -> Prop :=
refl_equal : x = x
```

Funes recursivas como Predicados

Um dos problemas garantir a terminao!

As definies indutivas permitem introduzir restries que garantam a terminao:

$f : A \to B$ pode ser descrita pelos pares $(x, f(x))$ por um predicado de tipo $A \to B \to Prop$.

```
Inductive Pfact : Z->Z->Prop :=
  Pfact0 : Pfact 0 1
| Pfact1 : forall n v:Z, n <> 0 -> Pfact (n-1) v -> Pfact n (n*v).
```

$Pfact\, n\, m$ a computao do factorial de n termina e retorna m.

```
Theorem pfact3 : Pfact 3 6.
```

Podemos determinar o domino e o contradomnio(***):

```
Theorem fact_def_pos:forall x y:Z,Pfact x y ->  0 <= x.
Theorem Zle_Pfact:forall x:Z,0<= x -> exists y:Z,Pfact x y.
```

`inversion`

```
Theorem not_even_1 : ~even 1.
Proof.
 unfold not; intros H.
 inversion H.
Qed.
```

Se se usasse `elim H` no era possvel unificar objectivo (`False` com o predicado P do principio indutivo `even_ind`). Para aplicar `inversion` necessrio que uma hiptese tenha o tipo indutivo. No exemplo, os construtores `O_even` e `plus_2_even` so considerados para `1`, e ambos falham (`discriminate`).

`inversion`

Usada quando se pretende raciocinar negativamente sobre os construtores de um predicado indutivo.

Esta tctica equivalente a usar `generalize e` (que modifica o objectivo `O` para `e -> O`) seguida de `pattern` (para obter P) e **discriminate**s (caso e seja uma igualdade).

No entanto, novos objectivos podem ser gerados:

```
Theorem plus_2_even_inv: forall n:nat,even (S (S n))->even n.
Proof.
 intros n H; inversion H.
 assumption.
Qed.
```

Tcticas (resumo)

`intros`: transforma um objectivo com implicaes e/ou quantificao universal num objectivo mais simples, onde as hipteses e variveis quantificadas vo para o contexto.

`apply H`: aplica `H` ao objectivo corrente, adicionando as premissas de H como subobjectivos.

`change e` Se o objectivo `e'`, muda o objectivo para `e`. Mas `e` e `e'` tm de ser equivalentes por computaes.

Tcticas para a igualdade

`reflexivity`: igualdade entre dois termos equivalentes

`rewrite H`: Reescreve o objectivo usando a igualdade da hiptese `H`. `rewrite H1 with H2` efectua a reescrita em `H2`. `rewrite <- H1` utiliza a igualdade da direita para a esquerda.

Tcticas

Tcticas de simplificao

`simpl`: simplifica o objectivo usando regras de reduo computacionais (converses). `simpl in H` simplifica a hiptese `H` ou `simpl in *` para simplificar o objectivo e todas as hipteses.

`unfold nome`: Sendo `nome` uma definio, expande todas as ocorrncias do nome no objectivo. Pode ser usada nas hipteses como `simpl`. Cuidado que pode complicar o objectivo...

Tcticas para tipos indutivos

`case`: raciocno por casos em `e`

`elim`: raciocnio indutivo

`discriminate` Demonstra uma frmula desde que nas hipteses exista uma igualdade entre construtores de um tipo indutivo (que devem corresponder a termos distintos...)

`injection H` Se `H` uma hiptese que iguala dois valores com o mesmo construtor, adiciona as igualdades que so geradas por ela (construtores so injectivos).

Tcticas

Tcticas compostas

`desctruct e` raciocnio por casos em `e`, criando um sub-objectivo para cada constructor do tipo de `e` e substituindo `e` pela aplicao desse construtor a uma varivel nova. (Pode utilizar `intros` antes).

`induction e` Igual ao anterior, mas os sub-objectivos podem ter hipteses adicionais. `e` pode ser uma varivel quantificada no objectivo.

Tcticas automticas

`trivial` Demonstra objectivos simples

`tauto` Demonstra tautologias proposicionais `auto` e `eauto`

`Hint Resolve` adiciona termos a uma base de dados de tcticas

`intuition`

`autorewrite`

`Hint Rewrite`

`congruence` Demonstra objectivos que so consequncia de igualdades e propriedades bsicas dos construtores

Tcticas

Tcticas numricas

Correspondem a resolutores automticos nos respectivos conjuntos.

Naturais `nat`, Inteiros $\mathbb{Z}$ e Reais $\mathbb{R}$

`ring` Para inteiros, resolve equaes polinomiais em anis ou semi-anis

`omega` Sistemas de equaes lineares e inequaes em `nat` e $\mathbb{Z}$

`fourier` Sistemas de equaes lineares e inequaes para reais.