Coqゼミ　第8回　（第7回のおまけ）

浅井研究室（担当：廣田知子）

2008年7月31日

1　はじめに

今回は第七回のおまけです。（こっちの方が重要な気もしますが … 。）

前回定義した型システムが**健全性**（type safety）を持つことを、Coqを使って証明するのが今回の目的です。（証明の仕方は、「Types and Programming Languages」の8章に書かれてあったのと、全く同じです。）

2　type safety

型システムの健全性とはtype safetyのことです。（type soundnessとも言います。）

type safetyの性質を満たす型システムでは、プログラムが型チェックを通ったら、プログラムの実行時には型エラーが決して起こりません。もっと単純に言い換えれば、well-typed term（well-typed program）がstuck状態に到達することはないということです。well-typed termとは、きちんとした型が付けられているtermのことです。stuckとは、これ以上評価規則を適用出来ず、かつ意味を成さない・誤りのあるterm（program）のことです。（例えば、`(tm_succ tm_true)`などがstuckに当たります。）

ですので、型システムがこの性質を持つかどうかは重要な問題になります。

type safetyは、preservationとprogressという二つの定理から成り立っています。この二つの定理がその型システムにおいて成立することを証明すれば、その型システムはtype safetyであることを示したことになります。

type safety = preservation + progress

preservation:

well-typed termはstuckではない。

progress:

もしwell-typed termがさらに評価出来るならば、評価後のtermもwell-typedである。

以下に、前回定義した、booleanとnumberのみの型システムにおいて上二つの定理がどのように定義出来るかを示します。

2.1　type preservation

もしterm tがTという型を持ち（つまり`t ~: T`であり）、tはt'へと評価出来る（`t --> t'`）ならば、t'はT型を持つ（`t' ~: T`）。

2.2　type progress

term tがTという型を持つならば（つまり`t ~: T`ならば）、tはvalueであるか、又は`t --> t'`を満たすt'が存在するかのどちらかである。

演習問題

問 1

「もし`t`が`Bool`型を持つ`value`であるならば、`t`は`tm_true`又は`tm_false`のどちらかである。　」という補題`canonical_forms_Bool`を証明せよ。

問 2

「もし`t`が`Nat`型を持つ`value`であるならば、`t`は`nv`である。」という補題`canonical_forms_Nat`を証明せよ。

問 3

定理 *preservation* を *Coq* 上で定義・証明せよ。

ヒント：`generalize dependent`を用いる。

問 4

定理 *progress* を *Coq* 上で定義・証明せよ。

ヒント：必要とあらば、

補題 `canonical_forms_Bool` と

補題 `canonical_forms_Nat` を用いよ。

3 証明手順の方針

3.1 type preservation の場合：

`t --> t'` の導出における帰納法を用いる。（「Types and Programming Languages」の Exercise 8.3.4 と同じ方法。）

つまり、`t --> t'` の規則（evaluation rule）について場合分けを行い、帰納法の仮定により、t より小さい term については preservation の定理が成り立つものと仮定して証明を行う。

3.2 type progress の場合：

`t ~: T` の導出における帰納法を用いる。

つまり、`t ~: T` の規則（typing rule）について場合分けを行う。帰納法の仮定により、t より小さい term については progress の定理が成り立つものとして証明を行う。

4 補足（tactic）

assert (式).

(式) を仮定に導入するための tactic。ただし、その式が真であることを証明しなくてはいけない。（つまり Goal が一つ増える。）

又、例えば`assert（式）as Hv.` 等とすると、その式の名前を自分で付けることが出来る。（この例では式の名前を Hv と名付けた。）

subst.

仮定や Goal の置換を勝手に行い、仮定をシンプルにしてくれる。

destruct H.

1. 仮定 `H: exists w, A w` に対して用いると、仮定が `w` と `H0: A w` となる。
2. 仮定 `H: A /\ B` に対して用いると、仮定が `He1: A` と `He2: B` になる。
3. 仮定が偽の場合、例えば `H: False` や `H: 1 = 2` のときに用いれば、Goal は完了する。
4. 再帰的なデータ型や命題の場合分けをしてくれる。例えば、`H: nvalue v` に対して用いれば、v に対して場合分けをしてくれる。

generalize dependent t.

仮定として上げられた forall 変数 t を Goal に引き戻す tactic.

下に例を示す：

```
----------------------------------------
forall (x y:tm), A x y -> B y -> C x y

Coq < intros.

x:tm
y:tm
T:typ
H0: A x y
H1: B y
-----------
C x y

Coq < generalize dependent x.

y:tm
H1: B y
----------------------------
forall x:tm, A x y -> C x y
```

注：仮定に入っていても、x が含まれる式は勝手に下に（Goal に）降りてくるので注意。