Programozási nyelvek és paradigmák

Programhelyesség

Kozsik Tamás (2020)

Megközelítési módok

- Bizonyítottan helyes kódok előállítása (correct-by-construction)
- Kód formális statikus helyességbizonyítása
- Modellellenőrzés (model-checking)
- Tulajdonságalapú tesztelés (*QuickCheck*)
- Futási idejű helyességvizsgálat

Statikus helyességellenőrzés

- Típuselmélet alapján, pl. Coq, Agda, Idris, Epigram...
- Logikai eszközkészlettel
  - Külső nyelv felhasználásával, pl. Isabelle, B-módszer
  - Metaprogramozással, nyelvfeldolgozó eszközzel, pl. ESC2/Java
  - Nyelven belül, pl. Ada/SPARK

Ada-2012

```
generic
    type Item is private;

package Stacks is

    type Stack( Capacity: Positive ) is tagged limited private with
        Invariant => Size(Stack) <= Stack.Capacity;

    function Size( S: Stack ) return Natural;

    procedure Push( S: in out Stack; I: in Item ) with
        Pre  => Size(S) < S.Capacity,
        Post => Size(S) = Size(S)'Old + 1 and Top(S) = I;

    procedure Pop( S: in out Stack ) with
        Pre  => Size(S) > 0,
        Post => Size(S) = Size(S)'Old - 1;

    function Top( S: Stack ) return Item with
        Pre  => Size(S) > 0;

private ... end Stacks;
```

Ada/Spark - 2014

```
generic
    type Item is private;
    Default_Value: in Item;
    --Capacity: in Positive := 5;

package Stacks is

    Capacity: constant := 5;

    type Stack is limited private; -- with Invariant => Capacity >= Size(Stack);

    procedure Empty( S: out Stack ) with
        Post => Size(S) = 0;

    function Size( S: Stack ) return Natural;

    procedure Push( S: in out Stack; I: in Item ) with
        Pre  => Size(S) < Capacity,
        Post => Size(S) = Size(S)'Old + 1 and Top(S) = I;

    procedure Pop( S: in out Stack ) with
        Pre  => Size(S) > 0 and Size(S) <= Capacity,
        Post => Size(S) = Size(S)'Old - 1;

    function Top( S: Stack ) return Item with
        Pre => Size(S) > 0 and Size(S) <= Capacity;

private ... end Stacks;
```

Szerződések ellenőrzése Eiffelben

- tervezés során a programozó által (dokumentációba írt állítások!)
- statikus (?)
- dinamikus (!)
  - vö. Java `assert`
  - no/require/ensure/invariant/all

vö: "`debug(`*Debug_key*`,...)` *utasítások* `end`" utasítás

Szerződések Eiffelben

- elő- és utófeltétel
- osztályinvariáns
- ciklusinvariáns, ciklus variánsfüggvénye
- check

Szerződések Eiffelben

- elő- és utófeltétel
- osztályinvariáns
- ciklusinvariáns, ciklus variánsfüggvénye
- check

Specifikációs nyelv:

- `require`, `require else`, `ensure`, `ensure then`, `invariant`, `variant`, `old`, `strip`
- logikai műveletek, `implies` is
- klózok: *és*-előd és, opcionális név és opcionális “;”

Öröklődés

- osztályinvariánsok *és*-elődnek
- felüldefiniálás
  - előfeltételek *vagy*-olódnak (`require else`)
  - utófeltételek *és*-elődnek (`ensure then`)

Osztálydefiníció helyessége

- osztály-helyes
- ciklus-helyes
- check-helyes
- kivétel-helyes

Osztálydefiníció helyessége

- osztály-helyes
- ciklus-helyes
- check-helyes
- kivétel-helyes

Felelősségkezelés:

- sok nyelvben kivételekkel, itt a szerződésekkel
- nem a kivétel terjedése az érdekes, hanem a felelősség terjedése (előfeltétel)
- a kivétel csak a legvégső eszköz