FP

Lez 5

Property-based testing
&
tagged values

Recap: PM su liste

- Sia `f : 'a list → 'b.`
  - Il PM tipico è:

```
let f xs =
 match xs with
    |[] -> e1
    | y::ys -> e2
```

dove `y,ys` tipicamente occorrono in `e2`

- Si possono fare PM più articolati

PM su liste 2

```
let rec ordered xs =
 match xs with
 |[] -> true
 |[x] -> true
 |x::(y::ys) -> x <= y &&
                ordered (y ::ys)
```

- Nota: `[x]` è sy-sugar per `x :: []`

PM su liste 3

- Anche PM non esaustivo da funzioni parziali:

```
let rec last xs =
  match xs with
   | [y] → y
   | _ :: ys → last ys

last : 'a list → 'a
```

Mini F#

Il nostro linguaggio, per adesso

- `p ::= c | id |(p,p)| [] | (p :: p)`
- `e ::= c | id | fun p -> e | e e |(e,e)|()`
  `| let p = e| let p = e in e`
  `| let rec id = e in e | e :: e | []`
  `| match e with p1 → e1 … pn → en`
- `t ::= b | α | t → t | α list | t * t |unit`
- `b ::= int | float | string | bool …`

TESTING

Why it matters …

Go to `fscheck.fsx`

Dijkstra's ghost

“Program testing can at best show the presence of errors, but never their absence” [*Notes On Structured Programming*, 1970]

“None of the program in this monograph, *needless to say*, has been tested on a machine” [Introduction to *A Discipline of Programming*, 1980]

Software testing

Most common approach to SW quality

- Very labour-intensive
  - up to 50% of SW development
- Even after testing, a bug remains on average per 100 lines of code, costing 60 billions $ (2002)
- Need of *automatic* testing tools
  - To complete tests in shorter time
  - To test better
  - To repeat tests more easily
  - *To generate test cases automatically*

Testing: the dominant paradigm

- By far the most widely used style of testing today is *unit testing*.
  - Invent a "state of the world".
  - Run the unit we're testing – a function/method
  - Check the modified state of the world to see if it looks like it should
    - Ericsson's ATM switch controlled by 1.5 mil LOC + 700.000 lines of UT
      - More modestly …

The dominant paradigm

```
public class TestAdder {
    public void testSum() {
        Adder adder = new AdderImpl();
        assert(adder.add(1, 1) == 2);
        assert(adder.add(1, 2) == 3);
        assert(adder.add(2, 2) == 4);
        assert(adder.add(0, 0) == 0);
        assert(adder.add(-1, -2) == -3);
        assert(adder.add(-1, 1) == 0);
        assert(adder.add(1234, 988) == 2222);
    }
}
```

The dominant paradigm

Problem: unit testing is only as good as your patience

- The previous example contains 7 tests.
- Typically we lose the will to continue inventing new unit tests long before we've exhausted our search of the space of possible bugs.

- (One) **solution**: property-based testing – PBT
  - A property is nothing more than a predicate that should always hold.

PBT: Quickcheck

- Quickcheck was introduced by Claessen & Hughes (2000)
  - Voted “most influential ICFP'00 paper”
- A tool for testing Haskell programs automatically.
- The programmer provides a specification of the program, in the form of *properties* that functions should satisfy
- QuickCheck then tests that the properties hold in a large number of *randomly generated cases*.

PBT

Quickcheck is now available for many PLs, including imperative ones, such as *Java, C(++), JavaScript, Objective-C,Perl, Erlang, Prolog, Python, Ruby, Scala …*

- **Quickcheck** is based on *random testing*
- There are alteratives such as (**Lazy) Smallcheck**, based on *exhaustive testing* and *symbolic execution*, but just for FP right no
  - We will see and use **FsCheck**
    - Random testing only

Commercial uses of PBT

- Mostly within QuviQ, Hughes' start-up commercializing Quickcheck for *Erlang* – here a free version
  - See paper "Quickcheck for fun and profit"
- Some success stories:
  - Ericsson's 4G radio base stations
  - Database reliability at Basho
  - Mission-critical gateway at Motorola
  - AUTOSAR Basic Software
  - Google's LevelDB database …

Commercial uses of FsCheck

- Credit Suisse
- BlueMountainCapital
- Tachyus
- Digital Furnace Games
- 15below

Details here

Back to code

Quickcheck: how

- Checking $\forall x : \tau.\ C(x)$ means trying to find an assignment

  $x \rightarrow a$ at type τ such that $\neg C(a)$ holds

  - e.g. checking $\forall xs$: 'a list. rev xs = xs means finding $xs \rightarrow$ [a2,a1] for which rev xs $\neq$ xs
- Quickcheck generates *pseudo-random* values of size *k* and stops when
  - a counterexample is found, or
  - the maximum size of test values has been reached, or
  - a default timeout expires

Conditional laws

- More interesting are *conditional laws*:
  - `∀ x: 'a, xs: 'a list. ordered xs ⟹ ordered (insert x xs)`
- Here we generate random list that may or may not be sorted and then check if insertion preserves ordered-ness
- If a candidate list does not satisfies the condition it is discarded
  - *Coverage is an issue:* what's the likelihood of randomly generating lists (of length > 1) that are *sorted*?
- Quickcheck gives combinator to *monitor* test data distribution – but in the end one has to write an ad-hoc generator, here yielding only ordered lists

Quickcheck's design decisions

- A lightweight tool – originally 300 lines of Haskell code, then extended to deal with the monadic fragment
- Spec are written via a DSL in the very module under test
- Adoption of random testing
- Put distribution of test data in the hand of the user
  - API for writing generators and observe distributions
- Emphasis on *shrinking* failing test cases to facilitate debugging