Chapter 6

Verifying Functions

Motivation

Verification

Verifying properties of functions is a fundamental task in SE. Hence it is an aspect of theorem proving. In particular, functions definitions allow to express recursive algorithms. Our focus here is on the definition of:

- terminiation/well-definedness properties
- functional properties, i.e., properties relating input parameters to the result (PR-properties).
- Example: A compiler can be considered as a partial function.

In general:

- specification = model + properties
- or
- specification = model_1 + model_2 + relationship

Conceptual aspects

Here: specification = function definition + PR-properties

Verify:

- well-definedness of function by:
  - often structural induction according to parameter types
  - more general: well-founded ordering on parameter space “show that parameters become smaller”
- PR-properties:
  - often structural induction according to parameter types
  - in general, proof technique depends on properties

Discussion

Verification

- works for the full parameter space (in contrast to testing)
- checks for consistency of models and properties
  - models may not reflect what programmer had in mind
  - properties may not reflect what programmer had in mind
  - proofs can have errors
- uses redundancy to find errors
- helps to improve the descriptions

Discussion (cont.)

Formal verification

- avoids misunderstanding
- allows using tools
- avoids errors in proofs
- $\rightsquigarrow$ Isabelle and others

‘

Case study: greatest common devisor

see Gcd.thy

Case study: Quicksort

Assumptions

Given:

```
datatype mapping = lt | ge

fun eval :: "mapping => universe => universe => bool"
    where
  "eval ge xa ya = not(eval lt ya xa)" |
  "[|eval lt ya xa|] ==>  eval lt xa ya = False"
```

Modeling in Isabelle using type classes!

Case study: Quicksort

Shallow embedding of the algorithm:

Case study: Quicksort (cont.)

```
fun qsplit  ::
 "mapping =>  universe => universe list => universe
    list"
  where

  "qsplit xf xa Nil      =  Nil"|
  "qsplit xf xa (ya#x)   =
   (if eval xf ya xa then ya#qsplit xf xa x
                     else qsplit xf xa x)"

fun qsort :: "universe list => universe list" where
  "qsort Nil       = Nil" |
  "qsort (p # l) =
   qsort (qsplit lt p l) @ p # qsort (qsplit ge p l)"
```

Properties to prove

Well-definedness/termination of qsort (1) and qsplit (2)

```
primrec counts :: "'a list => 'a => nat" where
"counts [] x      = 0"   |
"counts (y#yl) x = counts yl x +(if x=y then 1 else 0)
    "
```

lemma qsort_counts(3): “counts xl = counts (qsort xl)”

```
fun qsorted :: " universe list => bool" where
 "qsorted [] = True"|
 "qsorted [x] = True"|
 "qsorted (a#b#l) = (ge b a \and qsorted (b#l))"
```

lemma qsort_sort_prop(4): “qsorted (qsort xl)”

Verification of the properties

Ad 1: qsplit is primitive recursive

Ad 2: Idea: length of parameter decreases

```
Auxiliary lemma qsplit_length:
  "length (qsplit f p l) <= length l"
```

$\rightsquigarrow$ Proof termination with “length” as measure

Verification of the properties (cont.)

```
Auxiliary lemma counts_concat:
"counts (l @ m) x = (counts l x) + (counts m x)"
```

```
Auxiliary theorem qsplit_lt_ge_count [iff]:
  "count (qsplit lt p l) x + count (qsplit ge p l) x =
       count l x"
```

Prove lemma “qsort_counts” by induction

Property 4

Order lifting to lists

```
primrec qall :: "mapping => universe => universe list
    => bool" where
  "qall f p [] = True"
| "qall f p (h # t) = (f h p \and qall f p t)"
```

Property 4 (cont.)

Auxiliary Properties

```
theorem qsplit_splits:
   "qall f p (qsplit f p l)"

lemma qall_concat :
  "qall f p (a @ b) = (qall f p a \and qall f p b)"

theorem qsplit_qall :
  "qall f p l ==> qall f p (qsplit g q l)"

theorem qsort_qall :
  "qall f p l  ==> qall f p (qsort l)"
```

prop(4): “sorted (qsort xl)”

Auxiliary lemmatas

```
lemma qsorted_append :
  "[| qsorted l; qall ge p l|] ==> qsorted (p # l)"
```

```
theorem qsorted_concat :
  "[| qsorted a; qsorted b; qall lt p a; qall ge p b
     |] ==> qsorted (a @ p # b)"
```

»> Generic.QSort.thy