Executable Calculational Specifications

Francisco Cháves Camilo Rocha

Escuela Colombiana de Ingeniería

10 Congreso Colombiano de Computación – 10CCC
Bogotá, September 25, 2015

Agenda

The Calculational Style (*DS*)

- A semi-formal style of program verification and derivation
- Originated by Dijkstra and Hoare
- Consolidated by Dijkstra and Scholten

Common Uses

- Teaching logic and discrete mathematics
- Program verification and construction
- Symbolic manipulation

Common Uses

- Teaching logic and discrete mathematics
- Program verification and construction
- Symbolic manipulation
- Programming

Agenda

Expressions

In addition to usual mathematical operators and expressions, *DS* has special syntax for:

- Quantifiers
- Collections

Quantifier Notation

The notation:

$$\sum_{i=j}^{k} e$$

in some cases could have ambiguities:

Quantifier Notation

The notation:

$$\sum_{i=j}^{k} e$$

in some cases could have ambiguities:

- Scope:

$$\sum_{i=1}^{3} i + 1$$

means $(1 + 1) + (2 + 1) + (3 + 1)$ or $(1 + 2 + 3) + 1$

Quantifier Notation

Some ambiguities

- Range: the sum of prime numbers below k.
  Solution, relaxing with a predicate:

$$\sum_{\phi(k)} f(k)$$

Quantifier Notation

Some ambiguities

- Range: the sum of prime numbers below k.
  Solution, relaxing with a predicate:

$$\sum_{\phi(k)} f(k)$$

but $\sum_{k \leq i \leq k} i \times k$ reduces to

Quantifier Notation

Some ambiguities

- Range: the sum of prime numbers below k.
  Solution, relaxing with a predicate:

$$\sum_{\phi(k)} f(k)$$

but $\sum_{k \leq i \leq k} i \times k$ reduces to

$$\sum_{i=k} i \times k = \sum_{k=i} i \times k$$

the equal sign $(=)$ is a predicate or bounds a variable?

DS Quantifier Notation

DS notation:

$$(\Sigma \text{ bv} \mid \text{range} : \text{term})$$

Examples:

$$(\Sigma\ k \mid \phi(k) : f(k))$$
$$(\Sigma\ i \mid 1 \leq i \leq 3 : i^2)$$
$$(\Sigma\ i \mid 0 \leq i < n : a[i])$$
$$(\Sigma\ i, j \mid 0 \leq i < j \leq n : i + j)$$

DS Quantifier Generalization

$$(\oplus\ i \mid r : e)$$

where $(\oplus, \tau)$ is an Abelian monoid, i is a variable, r an optional predicate (True if not given) and $e \in \tau$ an expression

DS Quantifier Generalization

$$(\oplus\ i \mid r : e)$$

where $(\oplus, \tau)$ is an Abelian monoid, i is a variable, r an optional predicate (True if not given) and $e \in \tau$ an expression
Examples:

$$(+\ i \mid 1 \leq i \leq 3 : i^2)$$

$$(*\ i \mid 1 \leq i \leq 3 : i^2)$$

$$(\cup\ x \in S \mid\ : x)$$

DS Quantifier Generalization

$$(\oplus\ i \mid r : e)$$

where $(\oplus, \tau)$ is an Abelian monoid, i is a variable, r an optional predicate (True if not given) and $e \in \tau$ an expression
Examples:

$$(+\ i \mid 1 \leq i \leq 3 : i^2)$$
$$(*\ i \mid 1 \leq i \leq 3 : i^2)$$
$$(\cup\ x \in S \mid\ : x)$$

and also Universal and Existential quantification:

$$(\forall\ k \mid 0 \leq k \leq n : a[k] = 0)$$
$$(\exists\ k \mid 0 \leq k \leq n : a[k] = 0)$$

DS Collections

Collections can be defined by extension:

- Sets: $\{1, 2, 3\}$
- Bags: $\{\!|1, 2, 2, 3, 3, 3|\!\}$
- Sequences: $[1, 2, 1, 3, 1, 3]$

DS Collections

Collections can be defined by extension:

- Sets: $\{1, 2, 3\}$
- Bags: $\{\!|1, 2, 2, 3, 3, 3|\!\}$
- Sequences: $[1, 2, 1, 3, 1, 3]$

Collection comprehensions are defined similarly to quantification:

- Sets: $\{\ i \mid -3 \leq i \leq 3 : i^2\}$
- Bags: $\{\!|\ i \mid -3 \leq i \leq 3 : i^2|\!\}$
- Sequences: $[\ i \mid -3 \leq i \leq 3 : i^2]$

Agenda

The *Calculational* Package

Consider the following specification for computing the summation of values in a:

`Precondition:` $0 \leq N$
`Postcondition:` $s = (\Sigma i \mid 0 \leq i < N : a[i])$

The *Calculational* Package

Consider the following specification for computing the summation of values in a:

$$\texttt{Precondition: } 0 \leq N$$
$$\texttt{Postcondition: } s = (\Sigma i \mid 0 \leq i < N : a[i])$$

Mixing with the Haskell syntax, this can be specified in *Calculational* introducing a function:

```
sumA :: (Num e) => Array Int e -> Int -> e
sumA a n = [calc| (Σ i <- [0 .. n-1] | : a!i) |]
```

The *Calculational* Package

Consider the following specification for computing the summation of values in a:

$$\texttt{Precondition: } 0 \leq N$$
$$\texttt{Postcondition: } s = (\Sigma i \mid 0 \leq i < N : a[i])$$

Mixing with the Haskell syntax, this can be specified in *Calculational* introducing a function:

```
sumA :: (Num e) => Array Int e -> Int -> e
sumA a n = [calc| (Σ i <- [0 .. n-1] | : a!i) |]
```

using Haskell array indices function:

```
sumA :: (Ix i,Num e) => Array i e -> e
sumA a = [calc| (Σ i <- indices a | : a ! i) |]
```

Demo

The code shown can be seen as a correct prototype for a more efficient solution.

Conclusion and Remarks

The *Calculational* tool offers:

- A specification can be made executable for a broad class of calculational expressions.

Conclusion and Remarks

The *Calculational* tool offers:

- A specification can be made executable for a broad class of calculational expressions.
- The syntax includes quantifiers (summation, universal, existential, etc.) and collections (sets, bags and lists).

Conclusion and Remarks

The *Calculational* tool offers:

- A specification can be made executable for a broad class of calculational expressions.
- The syntax includes quantifiers (summation, universal, existential, etc.) and collections (sets, bags and lists).
- The executable code could be used as a *correct* prototype for a more efficient solution.

Future Work

- A formal alternative for database querying.

Future Work

- A formal alternative for database querying.
- Derivation of algorithms introducing symbolic variables.

Future Work

- A formal alternative for database querying.
- Derivation of algorithms introducing symbolic variables.
- Implementations in other programming languages.

Thank You!

Francisco Chaves
francisco.chaves@escuelaing.edu.co

Camilo Rocha
camilo.rocha@escuelaing.edu.co