SPARK - Correctness by Construction

Kim Rostgaard Christensen

DTU – Technical University of Denmark

Nov 30 2010

Outline

1. Correctness by construction
2. Design-by-contract
3. Example
4. Proving correctness
5. Gains
6. Challenges
7. Final words

Correctness by construction

Putting engineering back into software engineering

- CbyC challenges testing and debugging
- Manifested by the SPARK toolkit
- Proves the correctness of software
- - By subsetting the Ada programming language

Subsetting the language

Static programs for static analysis

- No dynamic behaviour
  - Only bounded types
  - No pointers
  - No aliasing
  - No tasking/threading (will be relaxed)
- All sizes known at compile-time

Is subsetting enough?

Warning: trick question

- Introduce a formalism
- Elaborates requirements
- Design-by-contract

Design-by-contract?

The fine print

- Translate requirements to a formal language
- Use this as the specification
- Hold the implementation to this
- Use automated tools
  - Unambiguous
  - Not subject to human error
  - Human code review -> producing proof

Example contract

How does it work

Quick question: What does this procedure do?

```
procedure Increment (X : in out Integer);
```

Example contract

How does it work

Is this implementation correct ?

```
procedure Increment (X : in out Integer) is
begin
   X = X - 1;
end Increment;
```

Example contract

Elaboration

Annotated correctly

```
procedure Increment (X : in out Integer);
--# post X = X~ + 1
```

Proving correctness by static analysis

Quod erat demonstrandum

- Uses pre- and postconditions
- Data flow analysis
  - Starts by the postconditions
  - - and seeks to deduct the preconditions
- Converts postconditions to conclusions
- Constructs hypothesises by visiting all branches
  - The rest is simple reduction
  - (don't worry, there is a tool)
- Internally done by matrix calculations

Outlook

Development phases

1. Convert informal requirements to specification
2. Use a lot of time on design
3. Validate specification
4. Break down implementation modules
5. Write one module at a time, proving it after
6. Test and debug

Gains

Warning: limited information

Productivity is lines of code (loc) per day, and defects are measured per kloc.

Project	Year	Size	Productivity	Defects
CDIS	1992	197 kloc	12.7	0.75
SHOLIS	1997	27 kloc	7.0	0.22
MULTOS CA	1999	100 kloc	28.0	0.04
A	2001	39 kloc	11.0	0.05
NSA	2003	10 kloc	38.0	0

Table: Comparison of projects using SPARK

Gains

The doubleplusgood

- Formal and systematic approach to development
- Produce low defect rate on software
  - Increase reliability and availability
- Proof == 100% correctness (according to specification)
- Forces you to spend time on design
- Provokes a lot of reflection (provability)
- Encourages modularity -> improves maintainability
- Guarantees analysis done in P time

Challenges

The doubleplusungood

- Fairly uncharted waters - no wide adoption
- Steep learning curve, not for the average programmer
- Strong bounds on language features
- Paradigm shift

Real-time domain language

Ada terminology

- Run-time has built-in dispatcher
- Ada task == Real-time task
- Compiler supports locking policies

Real-time domain language

Safety-critical

- Uses profiles to assert conformance
- RavenSPARK profile enables SPARK tasking
- Schedulabiliy analysis is directly applicable
- Check out the Assert project

Questions?