High Integrity Computing (F23PS2)

High Integrity Software Development (F29HD2)

Tools and Techniques Overview

Andrew Ireland

Department of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Outline

- Tool support for high integrity software development.
- An overview of the SPARK approach to high integrity Ada.

Special Purpose Tool Support

- The OBJ3 and CafeOBJ projects provide tool support for OBJ family of algebraic specification languages, *i.e.* formal proof and execution (see `http://vl.fmnet.info/obj/`).
- A number of tools have been developed to reason about Z specifications. Of particular note are CADiZ (University of York), Formaliser (Logica) and Z-Eves (Odyssey Research Associates, Ottawa). All have been used to support industrial strengthen case studies.
- VDM-SL (ISO Standard for VDM) has tool support in the form of the VDM-SL Tool Box (IFAD), *i.e.* testing and animation.
- B-Method provides formal proof support for model refinement.

General Purpose Tool Support

- **HOL:** A proof development environment for Higher Order Logic. Has been used extensively within the verification of hardware. Provides and very expressive logic in which other logics can be defined, *e.g.* Z has been embedded within HOL (see also ProofPower - based upon HOL but tailored to Z).
- **PVS:** Prototype Verification Environment. Another proof development environment for higher order logic. Has been used extensively within the verification of formal prototype designs, *e.g.* avionics (Rockwell Avionics) space (NASA).
- **ACL2:** A proof development environment for first order logic, includes significant capabilities in automating proof by mathematical induction.

Trade-Offs within Formal Methods

Two important concerns when developing models and specifications:

- **Abstraction:** crucial for mastering complexity, but care must be taken to ensure that no key characteristic of the "real" system is lost, *e.g.* modelling a railway system without the possibility of a driver passing a signal at danger has obvious limitations in terms of safety predictions.
- **Expressiveness vs analytical power:** the more expressive a representation one uses to model a system the harder the process of analysis becomes, *e.g.* relations are more expressive than functions but formal reasoning becomes harder.

Emerging Challenges

- Certification is the process of judging whether or not a product (software) conforms to a standard or some other form of specification.
- Typically the process of certification is carried out by an organization that is independent of the product developer.
- There is a trend towards the use of commercial-off-the-shelf (COTS) auto code generation technologies, *e.g.* Simulink.
- Formal verification of COTS auto code generators is not a practical option for commercial reasons and rapid product development cycles.
- Consequently, there is push towards the development of formal verification approaches that will under-pin COTS auto code generators - QinetiQ (formally DERA) and NASA Ames.

Code Level Static Analysis

- Program Proof Systems:
  - Stanford Pascal Verifier: Stanford University - 1970s
  - GYPSY: University of Texas at Austin (CLInc) - 1970/80s
  - Penelope: Odyssey Research Associates - 1990s
- Program Analysis Systems:
  - ESC/Java2: University College Dublin (previously ESC/Java: HP, Compaq, DEC) - current
  - Spec#: Microsoft - current
  - SPARK: Praxis Critical Systems Ltd (previously PVL) - current

Note: formal and conventional analysis is required at all levels of the product life-cycle.

Ada: A Historical Perspective

- In response to increased software development and maintenance costs the US Defence Department in 1975 published a set of strict safety criteria for programming languages that were to be used in their systems.
- None of the available languages met the criteria so a competition was set-up to design a new language.
- The winner was Ada (1983) and became a ISO Standard. Only compilers that accepted full Ada, *i.e.* pass a special validation test suit, could be called Ada.
- A major revision of the language was completed in 1995, giving rise to Ada 95: revision included object-oriented constructs, optional annexes, *e.g.* systems programming, real-time systems, distributed systems, security, ...

Ada Applications

- Ada is the most commonly used language for Air Traffic Control Systems worldwide.
- Ada is used increasingly in commercial “fly-by-wire” aircraft such as the Boeing 777 and Airbus.
- Ada is used within advanced avionics, *i.e.* fighter jets, and many military command and control applications.
- Ada is used for embedded systems in nuclear power plants, rail transportation systems, industrial process control, ...

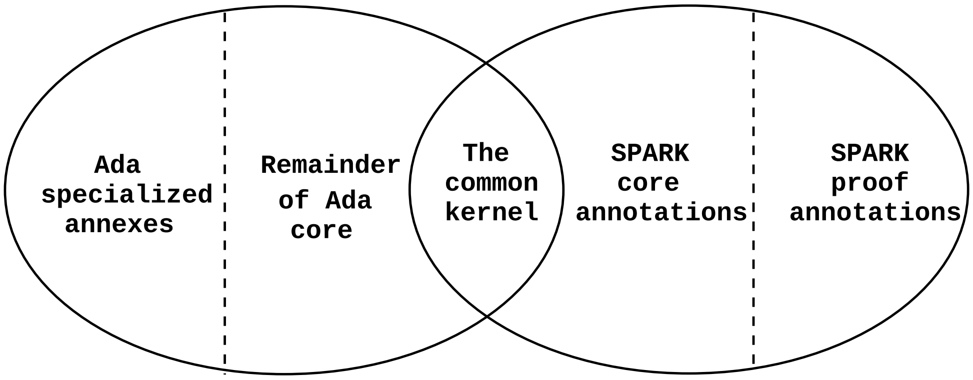
SPARK

- SPARK is a high level programming language aimed at high integrity applications.
- SPARK was designed to exploit the strengths of Ada while eliminating the potential for ambiguities and insecurities, *e.g.*
  - functions in SPARK are true mathematical functions, *i.e.* can not have side-effects, so any ambiguity in terms of order of evaluation is eliminated.
  - Aliasing is prohibited.
- SPARK was designed to support verification, both in terms of mainstream static analysis, *i.e.* flow analysis, and formal proof.
- The amount of space a SPARK program requires at run-time can be predicted via static analysis, *e.g.* guaranteed to have no memory leakage.

Some SPARK Applications

- The UK's next generation strike aircraft, Eurofighter Typhoon, is controlled by SPARK software (Def Standard 00-55).
- The Lockheed Martin C130J military transport aircraft used SPARK – software errors were trapped early leading to cost reduction in DO178B Certification, *i.e.* testing cost a fifth of the normal cost within the industry.
- MONDEX: International smart card security, through the use of SPARK, the highest level in ITSEC was achieved.
- SHOLIS: Ship Helicopter Operating Limits Instrumentation System, UK Ministry of Defence's first Def Standard 00-55 project.

Ada-SPARK Relationship



Based upon diagram from (Barnes, 2003)

SPARK Tool Support

- SPARK Examiner:
  - A static analyzer (written in SPARK) that eliminates various classes of errors.
  - Checks conformance of code against rules of the kernel language.
  - Checks consistency between code and embedded design annotations.
- The SPARK Examiner also provides a language of proof annotations for expressing desired dynamic behaviour:
  - pre- and postconditions
  - assertions such as loop invariants
  - declaration of proof functions

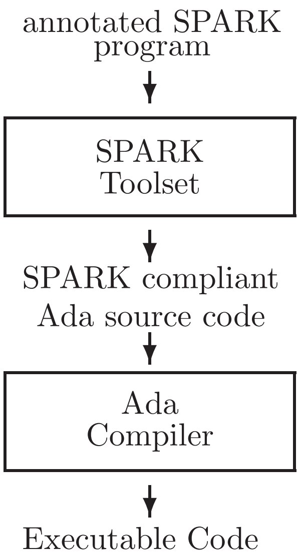
SPARK Tool Support

- The Examiner uses proof annotations to automatically generate mathematical conjectures, known as **verification conditions** (VCs).
- The SPADE Simplifier (automatic) and SPADE Proof Checker (interactive) tools support the proof of VCs.
- The Examiner can also automatically generate VCs that correspond to proving freedom from run-time exceptions, *e.g.* range violations.

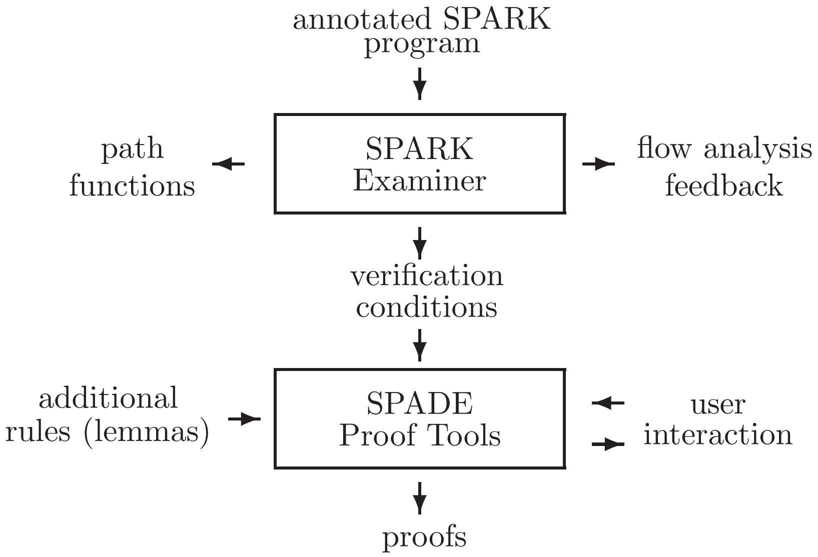
Z and SPARK

- Z can be used to model the dynamic behaviour of systems.
- There is a relatively small semantic gap between a Z system model and a SPARK annotation level description, *i.e.* pre- and postconditions and state invariants.
- The Examiner is applicable before coding is complete, *i.e.* the annotations should proceed the coding.
- SPARK approach advocates "correctness-by-construction".
- Note that adding annotations to existing Ada code, known as "Sparking the Ada" – is **not recommended!**

The SPARK Approach



The SPARK Toolset



Based upon diagram from (Barnes, 2003)

SHOLIS: A Closer Look

- Ship Helicopter Operating Limits Instrumentation System (SHOLIS).
- Decision support system that aids the safe operation of helicopters on navel ships.
- Major functions classed as SIL4

SHOLIS Development Process

- Requirements written in English [4000 statements]
- Software Requirements Specification (SRS) written in Z and English [300 pages]
- Software Design Specification (SDS) written in SPARK, Z and English.
- SPARK code [27 KLOC].

SHOLIS: Formal Methods

- Formal proof at the level of Z and SPARK.
- Z proofs were rigorous, with some use of CADiZ: *consistency of global variables and constants; existence of initial states; checking preconditions; checking safety properties.*
- SPARK program proofs were carried out using the SPADE proof tools (more details in later lecture).

SHOLIS: Lessons Learned

- Z Proof was the most efficient technique for finding faults, followed by System Validation Test (incorrect interpretation or translation of a requirements).
- SPARK Program Proof was found to be more efficient that Unit Testing, although the unit testing proceeded the proof phase.

What Next?

- Crash course in the SPARK approach
- Some of the theory behind the SPARK Examiner
- Theory and practice of SPARK Program Proof

Summary

Learning outcomes:

- Gain insight into the types of tools that support formal methods and the need to combine formal and conventional tools and techniques in practice.
- Gain an appreciation of the role SPARK can play in developing high integrity Ada.

Recommended reading:

- `http://secure.ucd.ie/products/opensource/ESCJava2` [ESC/Java2]
  `http://research.microsoft.com/specsharp` [Spec#]
  `http://www.sparkada.com/` [SPARK]
- "High Integrity Software: The SPARK Approach to Safety and Security" Barnes, J. Addison-Wesley 2003.
- "Is Proof More Cost-Effective Than Testing?" King, S. Hammond, J. Chapman, J. Pryor, A. IEEE Transactions on Software Engineering, Vol 26(8), 2000.