Protean Code ltd
Process, Technology and Training for Critical Software

Industrial Applications of Static Verification

Roderick Chapman, 25th November 2016

Contents

Protean Code ltd
Process, Technology and Training for Critical Software

- Goals for Static Verification
- SPARK is… (and isn't…)
- Verification - what and how
- Projects…

Contents

Goals…

Protean Code ltd
Process, Technology and Training for Critical Software

- “Design goals…hmmm…yes…you should definitely have some…”
  Guy L Steele Jr (ACM SIGPLAN 1994)

- SPARK has evolved over the years from its first definition in 1987 to the latest release in February 2016.

- BUT..we like to think that the design goals have persisted and have not been compromised.

General SV goals…

Protean Code ltd
Process, Technology and Training for Critical Software

- What would you really like from any SV tool?
- How about these “Big five” goals:

1. SOUND
   - No “False Negatives” please!
   - Sound tool says: “There are definitely no bugs”
   - Unsound tool says: “I’ve done my best and I can’t find any more bugs…”
   - The bad news: all general purpose SV tools for standard unsubsetted programming languages are unsound. See Coverity’s “Billion Lines of Code Later…” article in CACM, for example.

General SV goals…

2. COMPLETE

- Very few “False Positives” or “False Alarms” please.

- Very easy to write a tool that says “Warning: your program might have a bug…” all over the place…

- Very annoying for users. The number 1 cause of customer complaints!

- Zero false alarms is impossible for non-trivial program properties (See “Rice’s Theorem” for example).

General SV goals…

Protean Code ltd
Process, Technology and Training for Critical Software

3. FAST

- How long are you willing to wait for SV tool to run? It seems there are four important milestones:
  - "Now"
  - Coffee Break
  - Lunch
  - Overnight
- For non-trivial programs (e.g. 250kloc)
- Psychological driver: SV must be faster than "let's compile and test our program to see what it does…"

General SV goals…

4. MODULAR and CONSTRUCTIVE

- SV works on incomplete programs, during development.
- Modules can be verified in isolation. Integration of modules does not require re-verification of the “whole thing”.
- You never need to analyse the “whole program.”
  - nb…proper modularity has huge impact on analysis speed as well…
- A bit like “separate compilation” in most languages.

General SV goals…

5. DEEP

- Verification of non-trivial program properties, such as
  - “Type safety” and “Memory safety”
  - Absence of undefined behaviour
  - Partial correctness
  - Termination
  - Invariants
  - Application-specific safety- and/or security properties.

General SV goals…

- The “big five” goals
- The bad news: with any standard unsubsetted language (e.g. C, C++, Ada)…pick any 3.5!
  - Actually, you don’t get to pick…some tool vendor picks for you. Tough luck!
- The good news: there’s another way…

General SV goals…

Protean Code ltd
Process, Technology and Training for Critical Software

- Aside: Why is Soundness so hard to achieve?
- Technical reasons
  - Quiz time!
- Economic reasons

Contents

Protean Code ltd
Process, Technology and Training for Critical Software

- Goals for Static Verification
- SPARK is… (and isn't…)
- Verification - what and how
- Projects…

Protean Code ltd
Process, Technology and Training for Critical Software

SPARK isn't...

- SPARK ≠ Apache SPARK

and

- SPARK ≠ SPARC™

SPARK is…

Protean Code ltd
Process, Technology and Training for Critical Software

- …a programming language,
- a set of static verification tools,
- a discipline for high-integrity software development
- All of the above.

SPARK is…

Protean Code ltd
Process, Technology and Training for Critical Software

- SPARK design process:
  - Start with Ada (remember this all started in 1987…)
  - Soundness is #1 goal and non-negotiable.
  - Remove all undefined and unspecified behaviour by subsetting and introduction of "tighter" language rules. Aim for an unambiguous dynamic semantics.
    - Cool side effect: SPARK just works for all compilers and target machines.
- Remove language features that defy verification.
- Add contracts to enable both efficiency and modularity of analysis.
- Build tools…try to sell them…and use them…
- Do research…make language bigger…make tools more powerful…repeat…Adopt "good stuff" from Ada95, Ada2005, Ada2012…

Protean Code ltd
Process, Technology and Training for Critical Software

The most important language features that SPARK doesn't have?

- No "Access Types" – aka "pointers" !!!
- No Heap…
- No "malloc" or "free"
- No garbage collection.
- oh..and no runtime library to support any of the above…


Huh?!?!

No Pointers?!?!

Protean Code ltd
Process, Technology and Training for Critical Software

- Yes…but…
- Low-level programming *just works*… control of representation and data layout is easy in SPARK and Ada.
- Composite types are *first class* and we have high-level parameter passing modes.
  - Pass-by-reference *mechanism* is permitted for efficiency, but that’s OK and doesn’t break anything…

No Pointers?!?!

Protean Code ltd
Process, Technology and Training for Critical Software

- No heap -> predictable timing behaviour for real-time/embedded systems. (nb most coding standards for embedded/critical C say “no malloc”)

- Building linked data structures?
  - use arrays and array index values as “pointers” – works rather well!
  - or…we have formal, generic container libraries.

No Pointers?!?!

Protean Code ltd
Process, Technology and Training for Critical Software

- Implications for Verification…
- Massive simplification in modelling of variables and their values – no need to model “store” or “address” stuff…
- Calculation and verification of modsets is trivial.
- Aliasing analysis is trivial (and sound in P-time…)
- No need at all for a separation-logic. “Classic” Hoare logic worked all along…

Why contracts?

Protean Code ltd
Process, Technology and Training for Critical Software

- In C, consider, the following function prototype declaration:

```
int Sqrt (int x);
```

- What does this mean? What does the function promise to do? What is left un-stated?

Why contracts?

Protean Code ltd
Process, Technology and Training for Critical Software

- In C, consider, the following function prototype declaration:

```
int Sqrt (int x);
```

- Answer: not much. There is enough info here for the compiler to know how to call Sqrt, but not much else!

- Moral: Don’t let compiler-writers design programming languages! This trend has crippled program verification for decades…

Why contracts?

Protean Code ltd
Process, Technology and Training for Critical Software

- In SPARK 2014…

```
subtype Sqrt_Domain is Natural range 0 .. 1_000_000;

subtype Sqrt_Range  is Natural range 0 .. 1_000;
```

```
procedure Sqrt (X : in     Sqrt_Domain;
                Y :    out Sqrt_Range)
with
   Global => (In_Out => Call_Count);
```

Why contracts?

Protean Code ltd
Process, Technology and Training for Critical Software

- In SPARK 2014…going further…

```
subtype Sqrt_Domain is Natural range 0 .. 1_000_000;

subtype Sqrt_Range  is Natural range 0 .. 1_000;
```

```
procedure Sqrt (X : in     Sqrt_Domain;
                Y :    out Sqrt_Range)
with
   Global => (In_Out => Call_Count),
   Post   => (Y * Y) <= X and
             (Y + 1) * (Y + 1) > X;
```

Why contracts?

Protean Code ltd
Process, Technology and Training for Critical Software

- Summary

- Contracts tell the verification system exactly what is needed on the specification of a unit, not the body.

- They say what a unit does and doesn’t do.

- Contracts are the key to modularity, efficiency and abstraction in verification.

Contents

Protean Code ltd
Process, Technology and Training for Critical Software

- SPARK is… (and isn't…)
- Goals for Static Verification
- Verification - what and how
- Projects…

Verification

Protean Code ltd
Process, Technology and Training for Critical Software

- SPARK is designed to be amenable to many forms of verification
- Static
  - Subset and “type checking”
  - Data- and Information-flow analysis (essential to get rid of uninitialised variables before “proof” can start).
  - “Proof” - by generation verification conditions and then using a theorem-prover.
  - Worst-case memory-usage analysis.
  - Worst-case execution-time analysis.

Verification

Protean Code ltd
Process, Technology and Training for Critical Software

- SPARK is designed to be amenable to many forms of verification

- Dynamic
  - Testing, based on contracts. (like in Eiffel)
  - Structural coverage analysis.

- Note - in SPARK 2014, contracts can be used statically (for proof), dynamically (for test) or both. Allows “mixed language” development and verification.

Verification

Protean Code ltd
Process, Technology and Training for Critical Software

- **What can be “proven”???**
- “Type safety” - this really means:
  - No undefined behaviour.
  - No invalid data read or generated.
  - No “exceptions” - i.e. buffer overflow, arithmetic overflow, division by zero etc.
  - Program “never crashes” for any input data and state.

Verification

Protean Code ltd
Process, Technology and Training for Critical Software

- **What can be “proven”???**
- User-defined Assertions - anything you like!
  - Partial correctness against Pre- and Post-conditions.
  - Invariants (for types or package state)
  - Top-level safety and/or security properties. (Really just an assertion at the “main loop” level in your program.)

Verification - How

Protean Code ltd
Process, Technology and Training for Critical Software

- Under the hood - SPARK 2014

- SPARK 2014 is a complete reboot of the language design and tooling, based on Ada2012…

- Tools are structured like a compiler, but with a very different “back end.”

- “Front-end” is GNAT Pro Ada Compiler (GCC) and AdaCore’s GPS IDE.

Verification - How

Protean Code ltd
Process, Technology and Training for Critical Software

- “Middle-end”
  - Expands language constructs - e.g. generics.
  - Subset checking and extended legality rules.
  - Information-flow analysis, based on Program Dependence Graphs (PDGs).
  - Translation to “Why3ML” language for proof.

Verification - How

Protean Code ltd
Process, Technology and Training for Critical Software

- “Back-end”
  - Why3 VC Generator
    - Multiple “proof engines”, using SMTLib format
      - CVC4, Alt-Ergo, Z3 provers…
      - More to come…
- Oh…it’s all “FLOSS” (Freely Licensed/Open Source).

Contents

Protean Code ltd
Process, Technology and Training for Critical Software

- SPARK is… (and isn't…)
- Goals for Static Verification
- Verification - what and how
- Projects…

Projects

Protean Code ltd
Process, Technology and Training for Critical Software

- Not an exhaustive list, but a selection of significant projects, with data where available…

The first big industrial SPARK project…

Protean Code ltd
Process, Technology and Training for Critical Software

SHOLIS - 1995ish

Protean Code ltd
Process, Technology and Training for Critical Software

SHOLIS - 1995ish

Protean Code ltd
Process, Technology and Training for Critical Software

LM-130J - 1996-

Protean Code ltd
Process, Technology and Training for Critical Software

Tokeneer - 2003-

Protean Code ltd
Process, Technology and Training for Critical Software

- NSA-funded demonstrator for high-security software engineering.
- Developed by Praxis (now Altran UK) in Bath.
- Fully formal “Correctness by Construction” development.
- Delivered about 10kloc SPARK. Zero bugs found by customer until 2008.
- 2008 - Unprecedented open-source release of the entire project archive.
- Only 4 or 5 bugs ever found since…

NATS iFACTS 2006-now

Protean Code ltd
Process, Technology and Training for Critical Software

Before iFACTS…

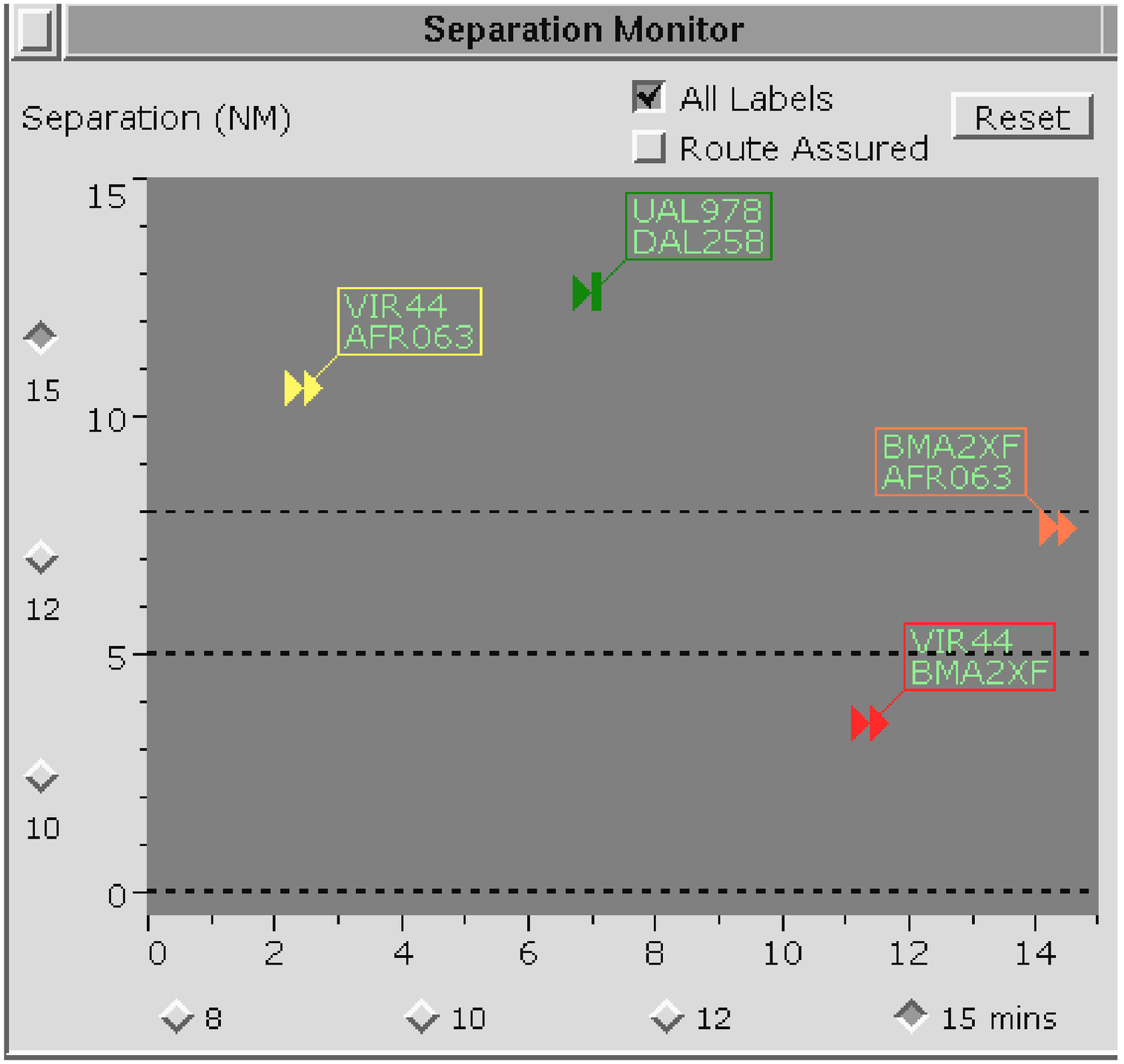
Protean Code ltd
Process, Technology and Training for Critical Software

Protean Code ltd
Process, Technology and Training for Critical Software

After iFACTS…

NATS iFACTS

Protean Code ltd
Process, Technology and Training for Critical Software



NATS iFACTS

Protean Code ltd
Process, Technology and Training for Critical Software

- In full operational service since December 2011.
- About 250kloc SPARK
- Proved “type safe” (i.e. crash proof etc.) *as a matter of course…*
- 152,000 VCs. 98.76% discharged automatically by tools. Remainder proved with user-defined lemmas, so 100% automation.
- Complete re-proof takes about 15 minutes on a standard PC, using distributed persistent proof caching. i.e. we’re down to “coffee break” for the whole system.
- Small changes are re-proved by developers before commit to CM system.

SPARKSKein - 2010

Protean Code ltd
Process, Technology and Training for Critical Software

- SPARK Reference Implementation of the “Skein” hash algorithm - one of the (then) contenders to become SHA-3.

- Implementation is: fast, formal, proven, portable, and (we think) readable.

- Debunks the myth that “formal is slow”.

- Released Open-Source.

- Also (en-passant) spotted a nasty corner-case overflow bug in the designer’s own (C) reference implementation.

Muen - 2013

Protean Code ltd
Process, Technology and Training for Critical Software

- Muen is a small separation kernel for x86_64 architecture.

- Proved type-safe and “crash proof”

- Open source

- Latest version now in SPARK 2014.

Protean Code ltd
Process, Technology and Training for Critical Software

To end…

- Static Verification really does work!
- A disciplined mindset can bring huge benefits.
- Soundness enables us to modify later verification activities (e.g. do less testing!). This can save lots of money.
- Many industrial projects do this for real, right now.

Homework…

Protean Code ltd
Process, Technology and Training for Critical Software

- Have a play…
- All the tools are GPL and freely available.
  - (Support for Professors, too… :-))
- Download Muen, SPARKSkein, or Tokeneer and see what you think.

Resources

Protean Code ltd
Process, Technology and Training for Critical Software

- SPARK 2014 - www.spark-2014.org (including language definition, blog, community projects page etc.)
- GPL tools: libre.adacore.com
- Tokeneer: www.adacore.com/tokeneer
- SPARKSkein: www.skein-hash.info
- Muen: muen.codelabs.ch
- Universities using SPARK: www.adacore.com/academia
- me: rod@proteancode.com
- SPARK Team in Bath: www.altran.co.uk