Beyond SPARK2014: the ProofInUse project

J.-C. Filliâtre, C. Marché, A. Paskevich[1]
C. Dross, J. Kanig, Y. Moy[2]

[1]LRI, Univ Paris-Sud, CNRS, INRIA Saclay

[2]AdaCore

IFIP WG 1.9/2.15

July 16th, 2014

Overview

History of SPARK and SPARK2014

(A short one, from my personal point of view)

The SPARK2014 Technology

(What can be done with SPARK2014?)

Beyond SPARK2014: The “ProofInUse” Project

(what should be added to/improved in SPARK2014?)

Case Study: The Patience Game from VSComp 2014

(if we have time)

Overview

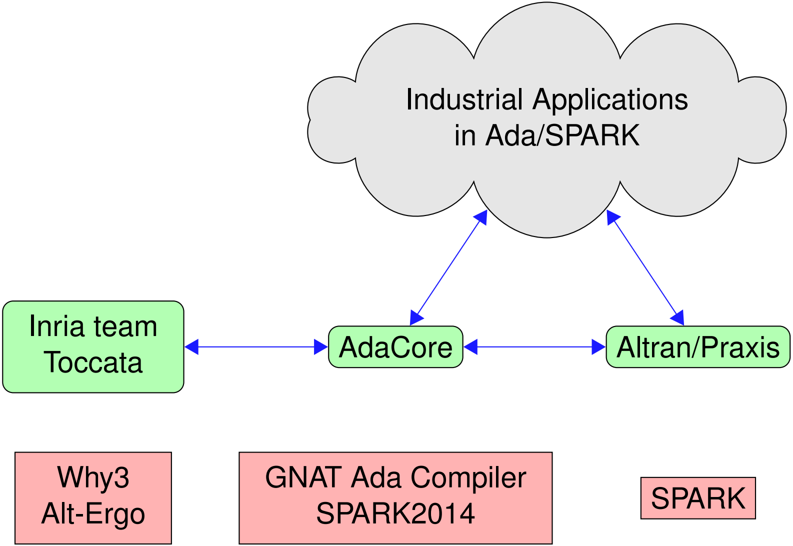
History of SPARK and SPARK2014
(A short one, from my personal point of view)

The SPARK2014 Technology

Beyond SPARK2014: The “ProofInUse” Project

Case Study: The Patience Game from VSComp 2014

Disclaimer: I’m not a SPARK expert at all



Industrial Applications in Ada/SPARK

Inria team Toccata

AdaCore

Altran/Praxis

Why3
Alt-Ergo

GNAT Ada Compiler
SPARK2014

SPARK

Short History of SPARK

SPARK (“SPADE Ada Ratiocinative Kernel”)

- subset of the Ada programming language
- around 20 years old, developed by Praxis and then at Altran
- *dedicated to development of critical systems*

SPARK dedicated to critical systems

- Restrictions w.r.t. Ada (statically checked):
  - strongly *restrictive aliasing rules*
  - data- and information-flow conditions
- *Contracts* specified as special form of comments
- Specification language based on the Z notation
- A VC generator, a dedicated prover (both automated/interactive)
- Several significant case studies

Some case studies

Tokeneer project

- demonstrator for high security software (NSA-funded)
- $\sim$ 10kloc
- 95.8% of the 2623 VCs proved automatically, some more proved interactively
- Microsoft Research Verified Software Milestone Award 2011

iFACTS project

- tools to assist en-route air-traffic controllers in the UK
- most ambitious SPARK project to date
- $\sim$ 250kloc
- 98.8% of the 152927 VCs proved automatically, some more proved interactively

For more details: see *paper in ITP 2014 proceedings*

Experiments with other theorem provers

Around 2010-2012

- Altran/Praxis implemented *another prover backend*, producing *SMT-LIB* syntax
- modern SMT solvers were clearly *increasing the number of VCs proved automatically*
- Last version of SPARK tool suite (2013) is *delivered with the Alt-Ergo SMT solver* (which is freely available)

Ada 2012 and SPARK 2014

Ada 2012:

- Last generation of the Ada standard
- Adds *contracts into the language*, under the form of aspects
- Can be checked *at run-time*

SPARK2014:

- A new SPARK language version, now a *subset of Ada2012*
- Completely *redesigned technology*, development leaded by AdaCore
  - as an add-on to the GNAT Ada compiler
  - *Why3* as intermediate language

Overview

History of SPARK and SPARK2014

The SPARK2014 Technology

(What can be done with SPARK2014?)

Beyond SPARK2014: The “ProofInUse” Project

Case Study: The Patience Game from VSComp 2014

SPARK 2014 in a nutshell

`http://www.spark-2014.org/`

- Inside the Ada tool suite (compiler, etc.)
- Static rules checks if subprograms belong to the SPARK2014 subset
  - Rejects unsupported feature (pointers, objects)
  - Data-flow analysis, overapproximation of side-effects
  - Name aliasing rejected
- Translation into Why3 programming language

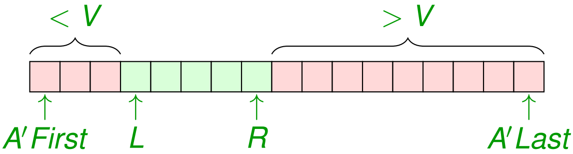
Importance of non-aliasing rules

no need for complex memory modeling like separation logic, dynamic frames, etc.

- Calls Why3 VC generator
- Uses Why3's *multi-prover output*: Alt-Ergo as default prover, but also CVC4, Isabelle, etc.

Demo: Binary Search

- Example: search a value *V* in an array *A* of integers
- Algorithm:
  - binary search
  - assuming *A* sorted in non-decreasing order



About using Why3 as intermediate language

- Aliasing restriction of SPARK2014 allow a quite straightforward translation
- But *differences in design choices* get into the way

Why3	SPARK2014
Distinct logic and programming language	Specifications are executable Any side-effect free function can be used in specification Quantification is always bounded
Functions in logic are total (statically check)	VC are generated to ensure that expressions in specs are totally defined

See also [“Why Hi-Lite Ada”, Boogie Workshop 2011]

Overview

History of SPARK and SPARK2014

The SPARK2014 Technology

Beyond SPARK2014: The “ProofInUse” Project

(what should be added to/improved in SPARK2014?)

Case Study: The Patience Game from VSComp 2014

What is “ProofInUse”

- 3 years project, starting April 2014
- Funded by French National Research Agency “ANR”

Double objective:

- push the technology forward
- advance academic knowledge

ProofInUse task 1: Improve Usability

- Help in the design of specifications
  - *debugging* specifications
  - Exploit the *counter-examples*, i.e. the models generated by SMT solvers in case of proof failure

  Inspiration: Microsoft's Dafny tool

  [Leino, FIDE 2014]

- Improve the degree of automation
  - Improve the encoding of *numerical data*
    - Machine integers: bitwise operations, unsigned wraparound arithmetic
    - Floating-point and Fixed-point numbers

ProofInUse task 2: Extend the supported features

- Extend the supported language
  - Ada2012's *type invariants*

  Absence of aliasing should make things easier than solutions adopted e.g. in OO languages

- Allow some kind of interactive proof
  - dedicated environment
  - dedicated “simple” proof tactics

  Inspiration: *Click'n Prove* in Atelier B

  [Abrial, TPHOLs 2003]

Overview

History of SPARK and SPARK2014

The SPARK2014 Technology

Beyond SPARK2014: The “ProofInUse” Project

Case Study: The Patience Game from VSComp 2014
(if we have time)

SPARK2014 capabilities?

- *Is SPARK 2014 already for solving problems from Verified Software competitions?*
- For this year's VSCOMP, we set up a "ProofInUse" team
  - C. Marché (Inria)
  - Y. Moy (AdaCore)
  - D. Mentré (Mitsubishi Electric R&D, Rennes, France)
- There was another "Purely AdaCore" team

The Patience Solitaire Game

Rules:

- take cards one-by-one from a deck of cards
- arrange cards face up in a sequence of stacks (from left to right):
  - first card: form the first singleton stack
  - each subsequent card: placed on the leftmost stack where its card value is no greater than the topmost card on that stack. If no such stack: new stack started to the right of others.

The Patience Solitaire Game

Example:

- input sequence: $9, 7, 10, 9, 5, 4, 10$
- result:

4		
5		
7	9	
9	10	10

Verification task

Verify the claim that the number of stacks at the end of the game is the length of the longest (strictly) increasing subsequence in the input sequence.

Proof strategy

During the competition:

- First, develop specifications *directly within Why3*
- Find appropriate specifications without bothering about integer overflow
- Then move the solution to a SPARK2014 program

Today:

- Show a preliminary code in SPARK
- Move to Why3 afterwards

SPARK code

Stacks represented by a record

- number of cards already seen
- an array of the values of cards (in the order they appeared)
- number of stacks
- sizes of these stacks: an array
- the stacks: array of arrays (of indexes of cards in the "values" array)

Second step: augment this record with *"ghost"* fields (next slide)

SPARK code: ghost fields and invariants

Ghost fields:

- array giving for each card (index) the stack number where it lies...
- ...and at which height (another array)
- an array of *predecessors* of cards: for each card, gives an index of a card in the stack on the immediate left, whose value is smaller. (-1 if card on the leftmost stack)

Add an quite large *invariant* on this complete record (see the code)

Why3 code

- Definition of the notion of (increasing) subsequence
- Proof of the claims, expressed as post-conditions to the main function
  - quantified over all subsequence (unbounded quantification)
  - needs the pigeon-hole lemma, proven in Why3 via a lemma function
- *these could not be done using SPARK2014* (yet!)

Conclusions

- SPARK “market” is a *very good target for dissemination of formal verification into industry*
- “ProofInUse” project is definitely in the aim of an IFIP WG 1.9 objective:

  > *“To contribute to a coherent toolset that automates the theory and scales up to the analysis of industrial-strength software.”*

- Yet *improvements are needed* for solving challenges like the VSCOMP ones
- Another on-going effort: *Operational Semantics of SPARK* (and maybe all Ada2012) *formalized in Coq*
  - Towards a verified compiler (a la CompCert)
  - Possibly a verified VC generator

[Herms et al., VSTTE 2012]