SPARK 2014 and GNATprove

Roadmap and Challenges

Roadmap

- May 29 2013: GPL 2013 release (called SPARK-HiLite)
- June 2013: hi-lite project on Open-DO moved to spark2014 project (also public)
- June 2013: finalization of SPARK 2014 RM
  - most of SPARK 2005 supported (not yet supported: RavenSPARK + Object Oriented programs)
  - generation of Global is not described in RM
- November 2013: beta release
- Q1 2014: release 1 of SPARK 2014
  - flow analysis, non-aliasing analysis, proof

Assumptions (1/3)

- Why?
  - Allow mixing different verification methods
  - Allow mixing of SPARK and non-SPARK code
  - Allow mixing of Ada and C Code
- How?
  - Two phases
  - Modular generation of explicit assumptions
  - Aggregation of assumptions and verification results
- Existing approaches
  - Evidential Tool Bus (SRI)
  - Frama-C collaboration of plug-ins

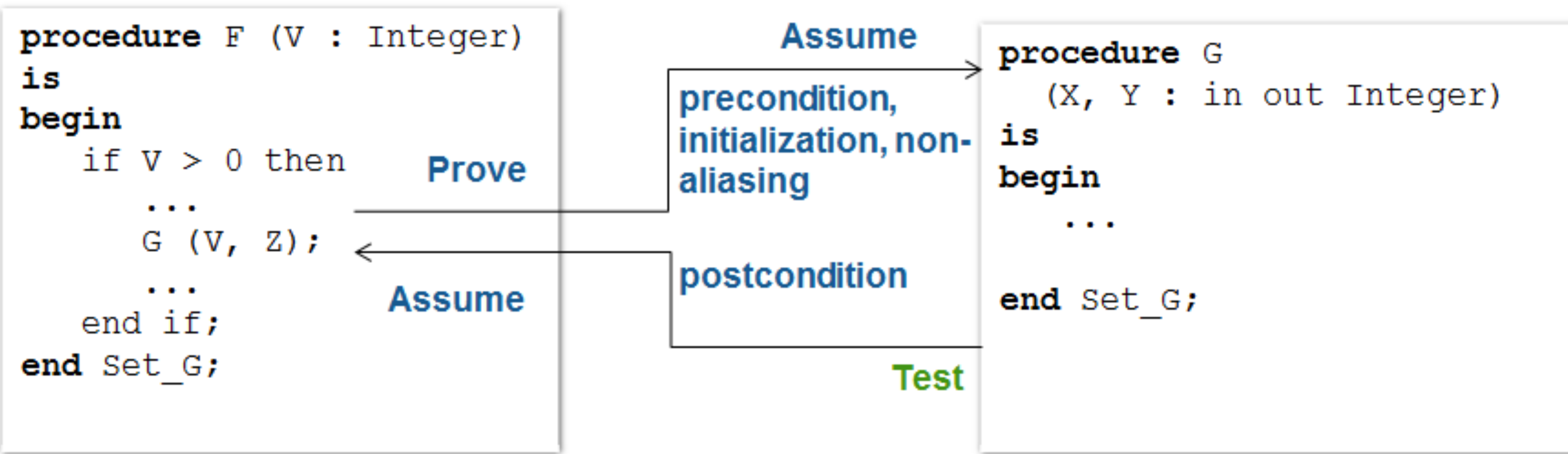
Assumptions (2/3)

```
procedure F
   (V : in out Integer)  is
begin
    if V > 0 then
      ...
      G (V, Z);
      ...
    end if;
end F;
```

Tool output:

```
file.adb:12:7: precondition of G proved
file.adb:15:6: postcondition of F proved
file.adb:12:7: postcondition of G assumed
```

Assumptions (3/3)



Christakis, Müller, Wüstenholz: Collaborative Verification and Testing with Explicit Assumptions, FM 2012

Object Oriented Code

- Support for behavioral subtyping only
- Check Liskov Substitutability Principle (LSP)
  - weaker Pre and stronger Post
  - Less Global Input and Global In_Out
  - same Global Ouput
- Subprogram checked against Pre/Post
- Which contract for dispatching?
  - Pre / Post ?
  - Pre'Class / Post'Class ?
- Global'Class / Depends'Class ?

RavenSPARK

- RavenSCAR is a subset of Ada for safe usage of tasking features (schedulability)
  - only top-level tasks
  - fixed priorities
- RavenSPARK is a subset of RavenSCAR compatible with SPARK
  - tasks communicate only through protected objects
- Proof of protected objects & tasks (seems "simple")
- Proof of manipulation of protected objects (similar as "volatile")

Data Invariants (1/2)

- Subtype predicates
  - "strong" invariant
  - Add a predicate to a type that should always be true
  - Will support only limited form in SPARK:
    - Cannot mention global variables
  - Support in GNATprove seems straightforward
    - insert assumptions/assertions where needed

Data Invariants (2/2)

- Type invariants
  - "weak" invariant
  - Add a predicate to a type that can be temporarily broken by "primitive operations" (functions) of that type
  - In SPARK, invariant should not depend on global variables
  - Sufficient to enrich precondition/postcondition?
  - Do we need more restrictions?

Non-Linear Arithmetic (1/2)

Two lines of work:

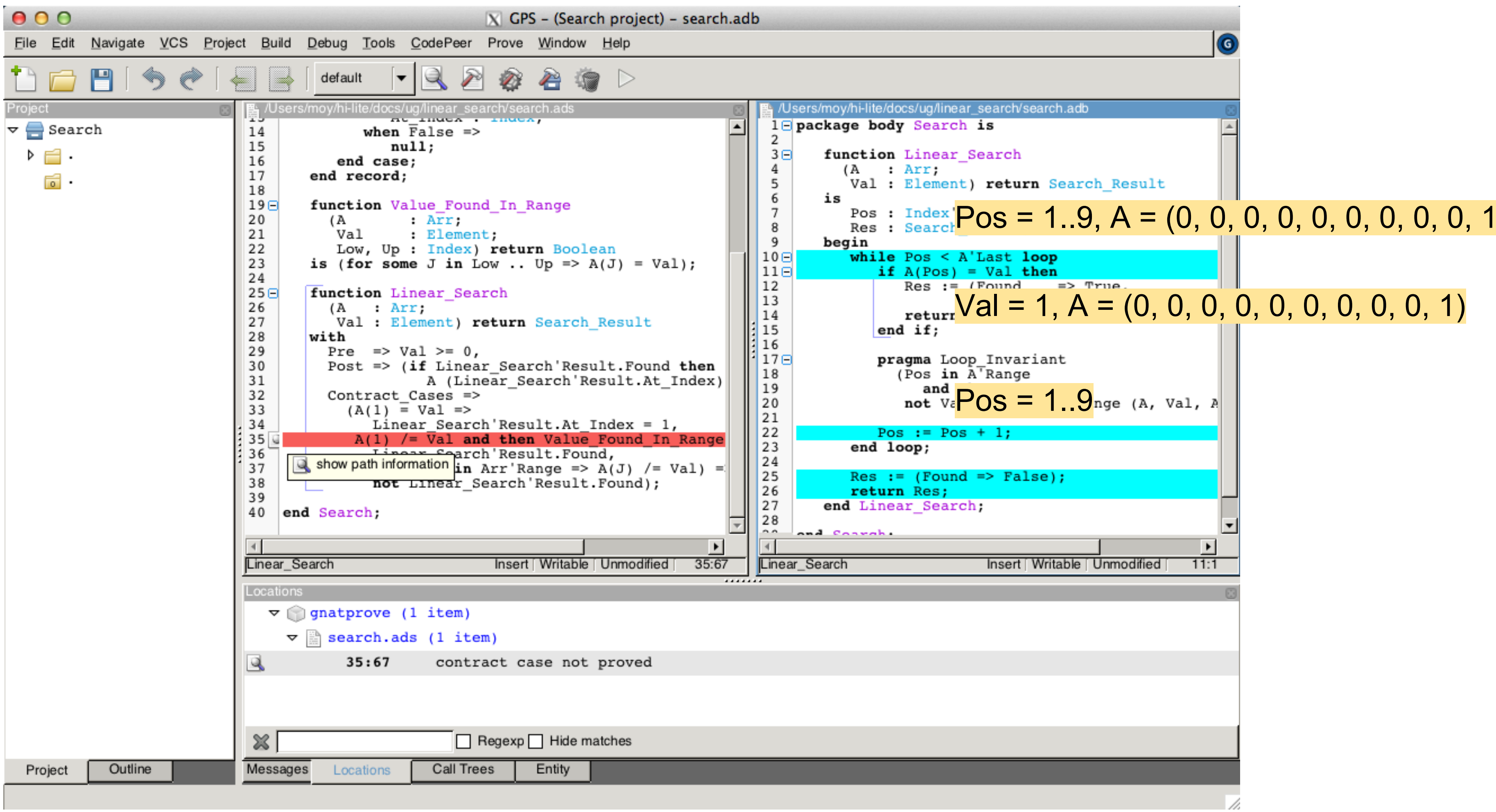
1. Axiomatisation + Heuristics
   - produce Why encoding/axiomatisation for non-linear operations
   - improve Alt-Ergo's provability and performance based on practical problems
2. Keep good interface with multiple provers
   - non-linear arithmetic is an active research area
   - decision procedures for SMT solvers using bit-vectors, computer algebra
   - implemented in Z3, CVC4, Boolector, Alt-Ergo?

Non-Linear Arithmetic (2/2)

- Axioms + Heuristics
  - Advantages: tailored for industrial problems, short-term bang for buck
  - Disadvantages: possibly fragile, prover specific
- Decision procedures
  - Advantages: more predictable, based on fundamental knowledge, long-term solution
  - Disadvantages: possibly too time consuming, may not work on industrial problems
- Compare Simplifier vs Victor

A bit of both?

Counterexamples (1/3)



Counterexamples (2/3)

1. generate VCs with labels

   `goal Incorrect: (forall x: int. ("model:0":x) <> 0)`

2. call alt-ergo with switch -model

   `alt-ergo -model file.why`

3. extract equalities with literals from model

   `x = X1(arith):[0 [int]]`

4. display extracted values in GPS

Concrete model instead of propositional one?
Partial model when timeout reached?

Counterexamples (3/3)

SMT v2

```
(set-logic AUFLIA)
(declare-fun x () Int)
(assert (= x 0))
(check-sat)
(get-value (x))
(exit)
```

Why

```
goal Incorrect:
  (forall x: int.
    ("model:0":x) <> 0)
```

∀ x, y : int.	x <> 0	x = 0	x < 0	x < y
Z3	x = 0	x = 1	x = 0	x = 0, y = 0
CVC4	x = 0	x = 1	x = 0	x = 0, y = 0
Alt-Ergo	x = 0	x ∈]-∞;-1] ∪ [1; +∞[	x ∈ [0;+∞[	x > (y - 1)
Riposte	x = 0	x = -1	x = 0	x = 0, y = 0
Sireum Kiasan	x = 0	x = 1	x = 0	x = 0, y = 0

Floating-points (1/2)

- mathematical reals are used to model floating points in proof
- difference between executable semantics and proof semantics
- false positives and negatives
- Way out: use floating point semantics and proof tools with floating point support (Gappa, Alt-Ergo + Gappa)

Boldo, Clément, Filliâtre, Meyero, Melquiond, Weis: Wave equation numerical resolution: a comprehensive mechanized proof of a C program. *Journal of Automated Reasoning, 2013*

Floating-points (2/2)

- Floating point semantics also for assertions, is it a limitation?
- Is NaN allowed?
- Is +/-Inf allowed?
- Can we have a type "float" in Why3 (programs)?

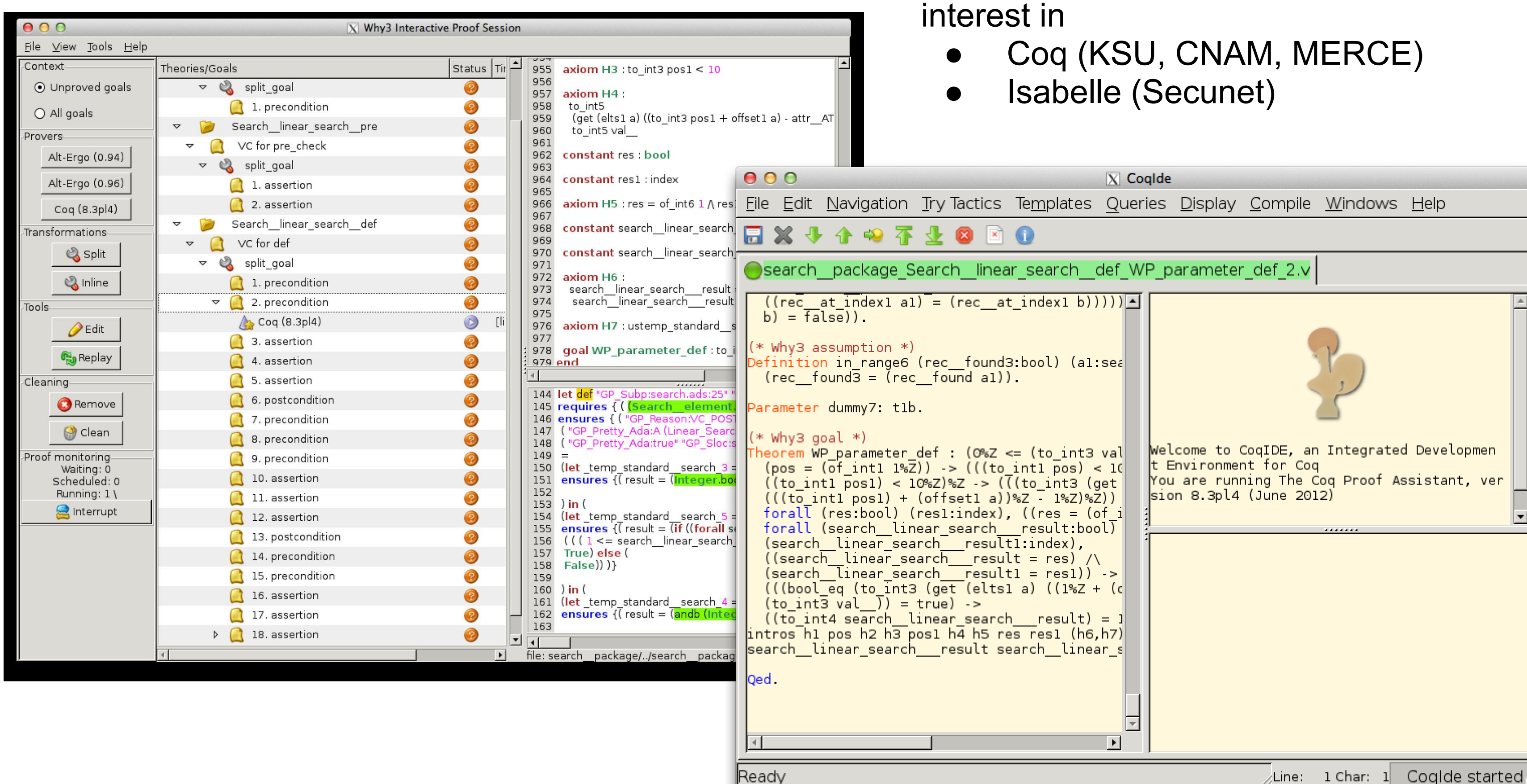
Multi-prover Approach

- Benefits
  - increase provability (portfolio approach)
  - help during debugging (detect prover shortcomings, generate counterexamples)
- SMT solvers
  - encoding is important, ongoing work
  - use of SMT built-in types as much as possible
  - careful use of triggers
- First-order provers
  - Why to Spass, E-prover, Vampire
  - possibly more: Paradox, Equinox..
  - need more investigation on practical problems

Axiomatized Units

- User can define Why3 theories for Ada units
  - To improve efficiency (containers)
  - To improve expressivity (sum_of, permutation...)
- Works for generic packages
  - Uses Why3 clone
- User can start from auto-generated stubs
  - Generate expected signature for Ada elements
  - Generate complete translation of Ada types
  - One namespace per Ada declaration
- Theories provided for SPARK Libraries

Bridge to Manual Provers



interest in

- Coq (KSU, CNAM, MERCE)
- Isabelle (Secunet)