AdaCore
The GNAT Pro Company

Alfa – GNATtest – GNATprove

Yannick Moy, AdaCore

Hi-Lite annual meeting – May 10th, 2011

Alfa – A language for combining tests and proofs

- **Annotation Language For Ada**
  - Based on Ada 2012 features for specifying program behavior
  - Specification language is fully executable
  - Proofs adopt the executable semantics of assertions

- **Annotated Language of Functions in Ada**
  - Entity of interest is subprogram (function, procedure)
  - Tests and proofs applied to subprogram: unit test / unit proof
  - Sequential execution at the subprogram level (no tasking)
  - Emphasis on functional behavior described in contracts

- **Analyzable Language From Ada**
  - Exclude language features too difficult to analyze: exceptions, pointers, controlled types, interfaces
  - Tradeoffs to increase level of automatic proofs (formal containers)

Alfa – A few principles

- **Annotations are read-only**
  - No writes to global variables in contracts / assertions
  - Detect run-time errors in annotations
- **Code is unambiguous**
  - No behavior dependent on compiler choice (parameter passing, order of evaluation of expressions, etc.)
- **Global effects of subprograms are generated**
  - No manual annotations for variables read and written, contrary to SPARK, JML, ACSL, Spec#
  - Requires all subprograms called to be implemented
- **Subprograms in Alfa / not in Alfa can be mixed**
  - Fine-grain combination of provable / unprovable code

The following slides present the <u>current</u> definition of Alfa.

Alfa is an evolving language.

A <u>future</u> definition could be more permissive as long as it respects the principles just defined.

Alfa – Subprogram specifications

```
procedure In_Alfa (X : Type_In_Alfa) with
   Pre  => Expression_In_Alfa,
   Post => Expression_In_Alfa;

procedure Not_In_Alfa (X : Type_Not_In_Alfa) with
   Pre  => Expression_Not_In_Alfa,
   Post => Expression_Not_In_Alfa;

procedure In_Alfa (X : in out Type_In_Alfa);

procedure In_Alfa (X : in out Type_In_Alfa)
   Pre  => True,
   Post => True;
```

Alfa – Subprogram bodies

```
procedure Body_In_Alfa (X : Type_In_Alfa) with
  Pre  => Expression_In_Alfa,
  Post => Expression_In_Alfa;

procedure Body_In_Alfa (X : Type_In_Alfa) is
   Var1 : Type_Is_In_Alfa;
   type T is Type_Def_Is_In_Alfa;
   procedure Local is Procedure_Spec_Is_In_Alfa;
begin
   Spec_Is_In_Alfa (...);
   Code_Is_In_Alfa;
end Body_In_Alfa;
```

Alfa – Function calls in code

Problem

- Functions can write global variables
- Compiler decides order of evaluation of expressions
- Together → ambiguity

Solution

- In Alfa, functions cannot write global variables
- Note that procedures can write global variables

Alternatives

- Fixed order of evaluation for expressions
- Syntactic restriction for problematic calls `(X := F (Args))`
- Analysis of effects to detect potential ambiguity

Alfa – Function calls in annotations

Problem

- Contract of such functions is essential for proofs
- Translation of contract into axiom can introduce inconsistency
- Inlining contract at calling point adds complexity

Solution

- In Alfa, only expression functions can be called in specs
- Expression function (Ada 2012): function body is expression
- Additional restrictions: expr-fun has no contract itself; inside expr-fun's body, calls must be to expr-funs and not recursive

Alternatives

- Syntactic restrictions on contract of functions called in specs
  `(Pre => Expr_With_No_Call; Post => F'Result = Expr;)`
- Inlining of contract at calling point

Alfa – Not yet implemented

MAJOR

- OO programming: tagged types, dispatching
- Generics
- Axiomatized libraries (formal containers)
- Invariants on types (invariants and predicates)

MINOR

- Discriminant / variant records
- Array slices
- "declare" block statements
- Elaboration code
- Many corner cases in expressions

GNATtest – Unit test of Ada code

Benefit 1: Formalization of test case

Benefit 2: Automatic generation of test harness

Benefit 3: Automatic verification that test procedure matches test case

```
procedure In_Alfa (X : Type_In_Alfa) with
  Pre  => Expr_In_Alfa,
  Post => Expr_In_Alfa,
  Test_Case => (Name      => "RU sleepy?",
                Mode      => Normal,
                Requires => Expr_In_Alfa,
                Ensures  => Expr_In_Alfa);
```

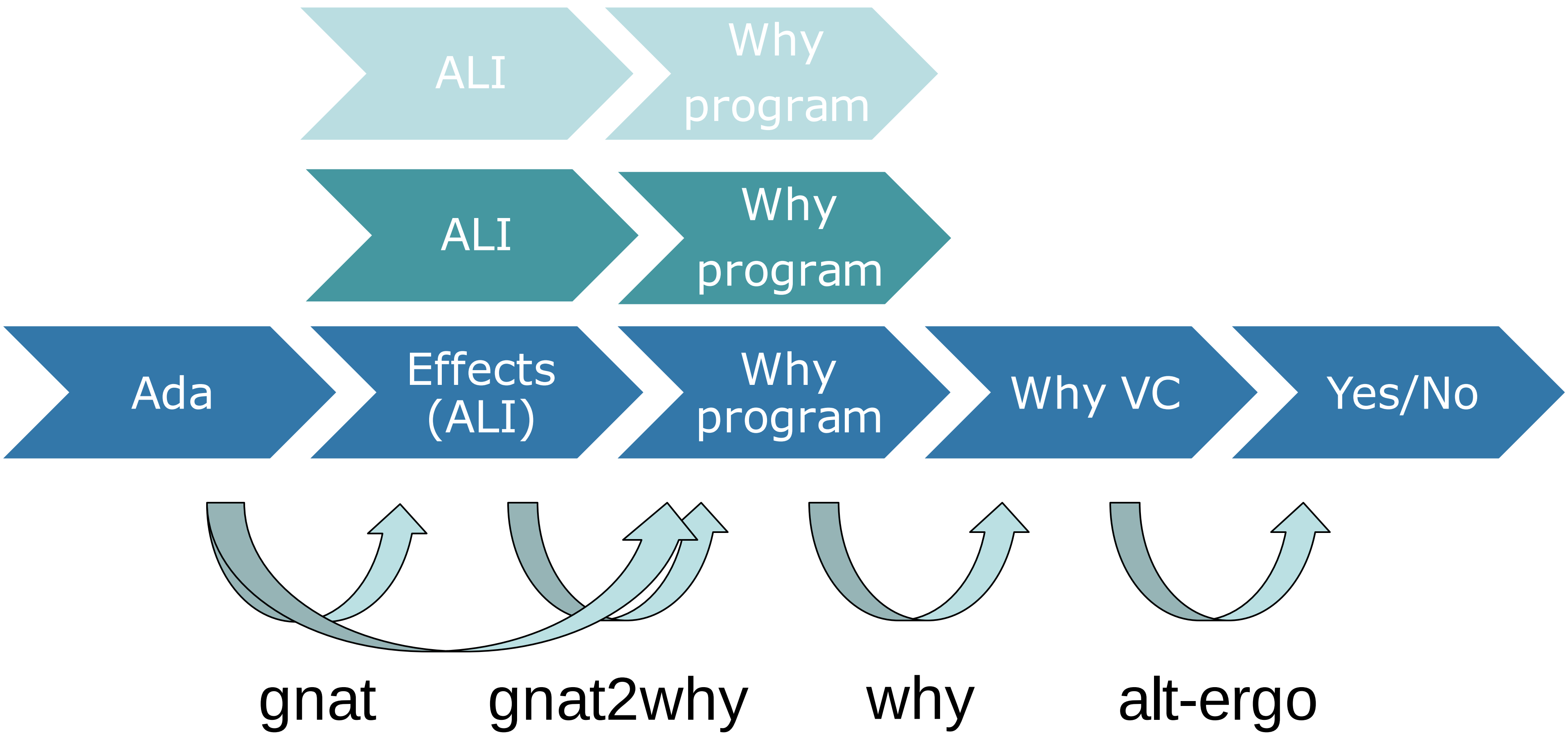
Formal contract and test case → GNATtest → Test harness → User writes test procedures → Automatic verification

AdaCore
The GNAT Pro Company

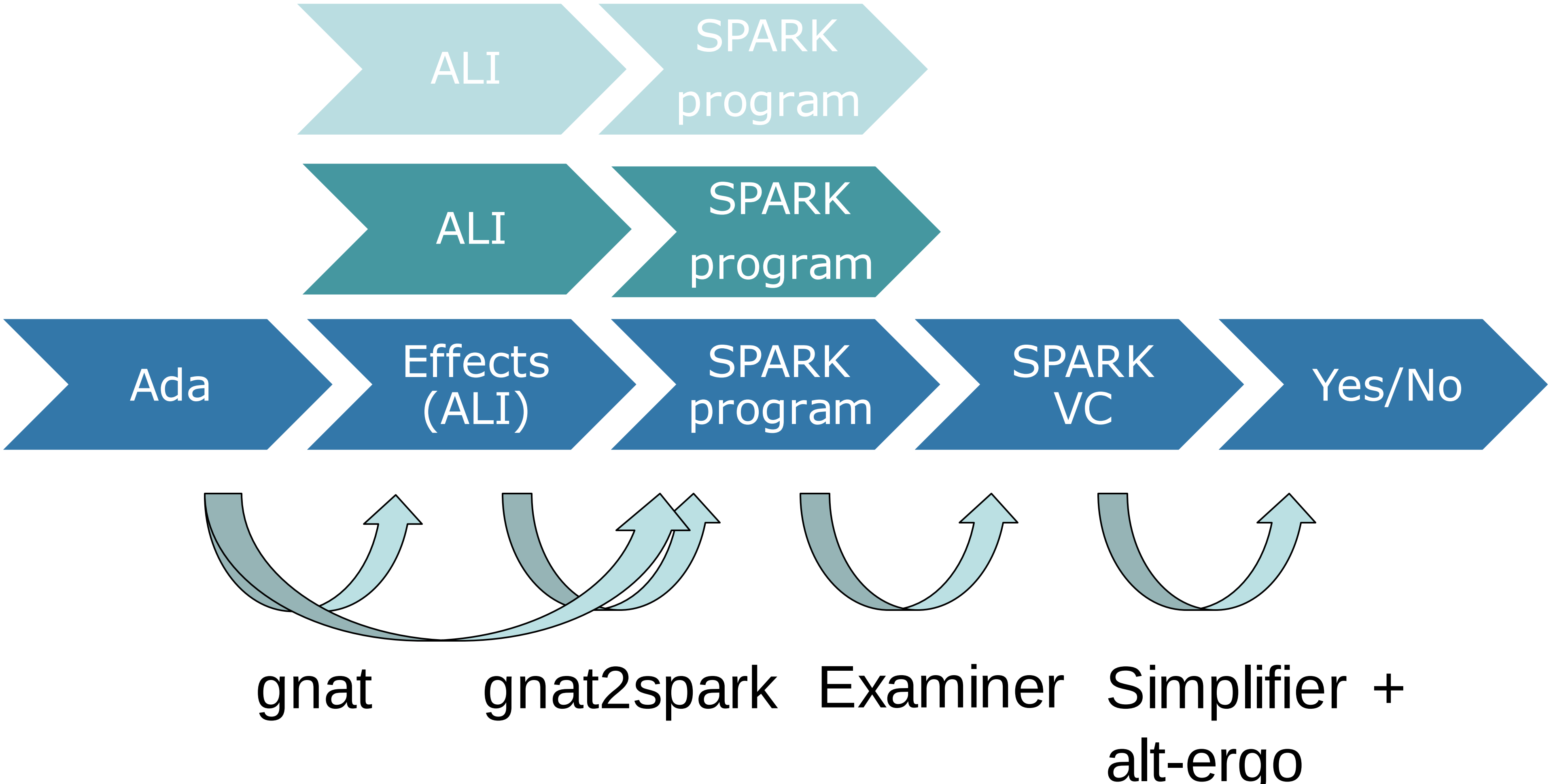
GNATprove – Unit proof of Ada code

1. Automatic detection of subprograms in Alfa
2. For subprograms in Alfa, but not yet supported, output reason (dispatching, slices, etc.)
3. Check that preconditions are auto-guarded (cannot raise run-time error whatever the context)
4. In progress: check subprogram contracts
5. In progress: check absence of run-time errors
6. TO DO: check no reads of uninitialized data (SPARK?)
7. TO DO: check absence of logically dead code (code which does not contribute to establish postcondition)
8. TO DO: check absence of redundant specifications

GNATprove – A compilation chain for proofs



GNATprove – Alternative compilation chain



Conclusion

❖ It is real!

- Ada 2012 implemented in GNAT (95%)
- Working prototypes GNATtest & GNATprove

❖ You can influence it

- Alfa is an evolving language
- Support for features will be prioritized according to user needs

❖ Most important work is yours

- User interaction with GNATtest & GNATprove
- Combining tests and proofs
- Real adoption in your context