Alfa & GNATprove: progress and future work

Yannick Moy

Hi-Lite Joint Meeting
November 29, 2011
Achievements!
Progress in the last 6 months

- mode check is complete
  - absence of run-time errors in preconditions

- mode prove is complete:
  - verification of contracts
  - verification of absence of run-time errors

- preanalyzed standard library

- ACATS pass without errors

- transition from Why2 to Why3

- Alfa definition given by reference manual

- strategy for mixing proof+test:
  - Alfa friendly subset of Ada
  - special compiler mode for testing
procedure Shadow_Effect (P : in out Painting; D : Dot)  
with  
Pre => (for some C in Color => P.Plain(C) = D),  
Post => (for all C in Color =>  
  (if P.Plain'Old(C) = D then  
     P.Shadow(C) = D));

procedure Shadow_Effect (P : in out Painting; D : Dot) is  
begin  
  for C in Color loop  
    if P.Plain(C) = D then  
      P.Shadow(C) := D;  
    end if;  
  end loop;  
end Shadow_Effect;
for C in Color loop
  pragma Assert
  (P.Plain = P.Plain'Old and then
   (for all J in Color'First .. Color'Pred (C← ) ⇒
    (if P.Plain (J) = D then P.Shadow (J) ← D))));
if P.Plain(C) = D then
  P.Shadow(C) := D;
  end if;
end loop;
function Get_Plain (P : Painting; C : Color)

return Dot is (P.Plain(C));

function Plain_Is_Dot (..) return Boolean is
(Get_Plain(P,C) = D);

function Some_Plain_Is_Dot (..) return Boolean is
(for some C in Color => Plain_Is_Dot(P,C,D));

procedure Shadow_Effect (..) with

Pre => Some_Plain_Is_Dot(P,D),
Post => (for all C in Color =>
(if Plain_Is_Dot(P'Old,C,D) then
   P.Shadow(C) = D));
Alfa restrictions

- no tasking
- no exceptions
- no pointers
- no controlled types
- no Unchecked_Conversion
- no goto
- functions cannot write global variables

Restriction Alfa applies to units completely in Alfa.
Implicit contracts

- strong typing:
  - precondition: input parameters have in-type values
  - postcondition: output parameters have in-type values
  - → additional VCs to prove
  - what about global variables?

- effects:
  - global variables read and/or written
  - special Heap variable
  - computed by GNATprove

- non-aliasing:
  - similar to SPARK and Why3
  - detected by GNATprove
Implicit contracts

- **strong typing:**
  - precondition: input parameters have in-type values
  - postcondition: output parameters have in-type values
  - → additional VCs to prove
  - what about global variables?

- **effects:**
  - global variables read and/or written
  - special Heap variable
  - computed by GNATprove

- **non-aliasing:**
  - similar to SPARK and Why3
  - detected by GNATprove
Implicit contracts

- strong typing:
  - precondition: input parameters have in-type values
  - postcondition: output parameters have in-type values
  - → additional VCs to prove
  - what about global variables?

- effects:
  - global variables read and/or written
  - special Heap variable
  - computed by GNATprove

- non-aliasing:
  - similar to SPARK and Why3
  - detected by GNATprove
Mixing proof and test

formal verification of $P$ assumes:

- precondition of $P$
- postcondition of subprograms called
- both user-defined and implicit ones

Assumptions made for proof should be verified by testing.

2 cases:

- tested $T$ calls proved $P$
  $\rightarrow$ check precondition of $P$ at run-time
- proved $P$ calls tested $T$
  $\rightarrow$ check postcondition of $T$ at run-time
- ...during test of $T$!
Mixing proof and test

formal verification of $P$ assumes:

▶ precondition of $P$
▶ postcondition of subprograms called
▶ both user-defined and implicit ones

Assumptions made for proof should be verified by testing.

2 cases:
▶ tested $T$ calls proved $P$
  $\rightarrow$ check precondition of $P$ at run-time
▶ proved $P$ calls tested $T$
  $\rightarrow$ check postcondition of $T$ at run-time
▶ ...during test of $T$!
Mixing proof and test

formal verification of $P$ assumes:

- precondition of $P$
- postcondition of subprograms called
- both user-defined and implicit ones

Assumptions made for proof should be verified by testing.

2 cases:

- tested $T$ calls proved $P$
  $\rightarrow$ check precondition of $P$ at run-time
- proved $P$ calls tested $T$
  $\rightarrow$ check postcondition of $T$ at run-time
- ...during test of $T$!
Testing and implicit contracts

- **strong typing:**
  - checks for in-type values added by compiler

- **effects:**
  - computed by GNATprove

- **non-aliasing:**
  - between arguments: checks added by compiler
  - between arguments and global variables in effects: detected by global static analysis
Alfa friendly restrictions

- no unchecked conversions between pointers
- no subprogram pointers
- no controlled types

Profile Alfa_Friendly applies to Alfa friendly units.
What next?
Language support

- generics: verification of instantiations
- formal containers
- discriminant records

1 type Image (Size:Natural; Color:Boolean) is
2 record
3   case Color is
4       when False => BnW : Pixels (1 .. Size);
5 ...

- tagged types, classwide types and dispatching
Extensions

- global contracts:
  - needed for subprograms not implemented in Ada
  - possibly useful for separate verification
- attribute Loop_Entry (instead of Old)
- LLR similar to Test_Case but part of contract
Faster proofs

- less VCs:
  - for example, N queens: 80 loc, 8609 VCs
  - need Why2 efficient WP in Why3
  - use memo feature of Why3 to avoid reproving
  - remove trivial VCs guaranteed by strong type

- smaller VCs:
  - skip hypotheses due to RE checks
  - fine-grain use of theories and modules

- easier VCs:
  - better model of arrays (length, first)
  - separate components of records in Why3

- faster prover:
  - use built-in theories (arrays)
  - selection of hypotheses (internship)
  - alternate provers (Z3?)
Faster proofs

- less VCs:
  - for example, N queens: 80 loc, 8609 VCs
  - need Why2 efficient WP in Why3
  - use memo feature of Why3 to avoid reproving
  - remove trivial VCs guaranteed by strong type

- smaller VCs:
  - skip hypotheses due to RE checks
  - fine-grain use of theories and modules

- easier VCs:
  - better model of arrays (length, first)
  - separate components of records in Why3

- faster prover:
  - use built-in theories (arrays)
  - selection of hypotheses (internship)
  - alternate provers (Z3?)
Faster proofs

- less VCs:
  - for example, N queens: 80 loc, 8609 VCs
  - need Why2 efficient WP in Why3
  - use memo feature of Why3 to avoid reproving
  - remove trivial VCs guaranteed by strong type

- smaller VCs:
  - skip hypotheses due to RE checks
  - fine-grain use of theories and modules

- easier VCs:
  - better model of arrays (length, first)
  - separate components of records in Why3

- faster prover:
  - use built-in theories (arrays)
  - selection of hypotheses (internship)
  - alternate provers (Z3?)
Faster proofs

- less VCs:
  - for example, N queens: 80 loc, 8609 VCs
  - need Why2 efficient WP in Why3
  - use memo feature of Why3 to avoid reproving
  - remove trivial VCs guaranteed by strong type

- smaller VCs:
  - skip hypotheses due to RE checks
  - fine-grain use of theories and modules

- easier VCs:
  - better model of arrays (length, first)
  - separate components of records in Why3

- faster prover:
  - use built-in theories (arrays)
  - selection of hypotheses (internship)
  - alternate provers (Z3?)
Implement analyses

- for proof alone:
  - non-aliasing between parameters and global variables
  - VCs for in-type parameter values (initialization)

- for test and proof:
  - special mode of GNAT compiler
  - detect violations of Alfa friendly subset
  - insert checks for in-type parameter values
  - insert checks for parameter non-aliasing
  - global static analysis for non-aliasing between parameters and global variables
User interaction

- currently: batch mode
- integration in project files
- integration in GPS/GNATbench
- ability to use Why3 IDE
- which kind of interaction?