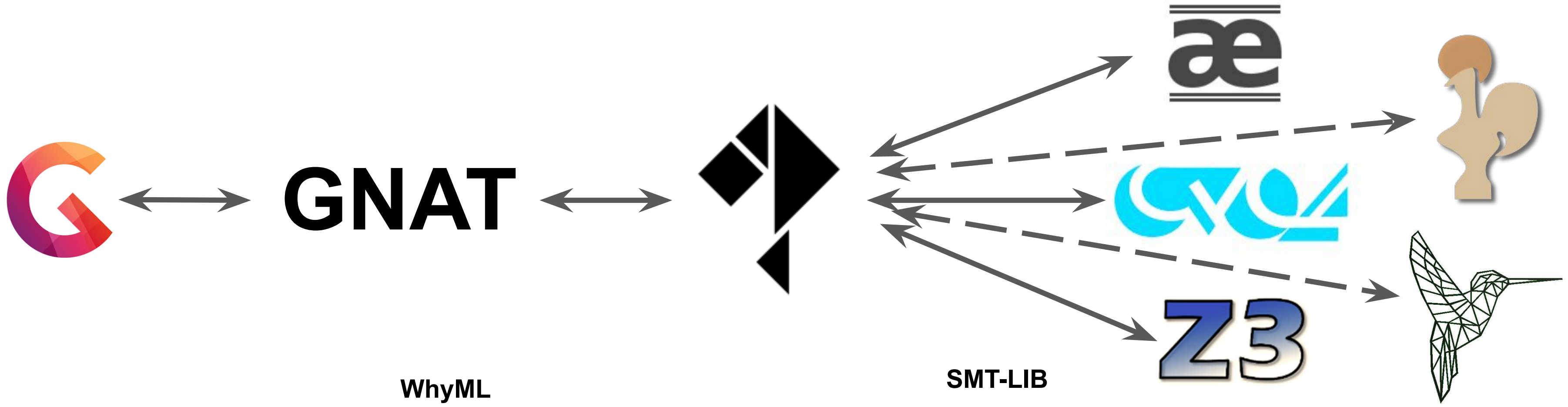
Interaction Features in SPARK

Yannick Moy - AdaCore

AdaCore altran
PARTNERSHIP

PROOF IN USE
SPARK-2014.ORG/PROOFINUSE

SPARK - Auto-Active Proof for Ada Programs



SPARK

```
A(1) := 42;
```

WhyML

```
a.map__content <-
  set
    (a.map__content)
    (let temp = 1 : int in
       assert { temp ... };
       temp)
    (42 : value)
```

SMT-LIB

```
(assert
  (not
  (=> (dynamic_property 0 1000000
         (to_rep a__first) (to_rep a__last))
  (=> (and (= (to_rep a__first) 1)
           (<= 0 (to_rep a__last)))
  (<= (to_rep a__first) 1)))))
(check-sat)
```

SPARK - User-Level View

SPARK

```
A(1) := 42;
```

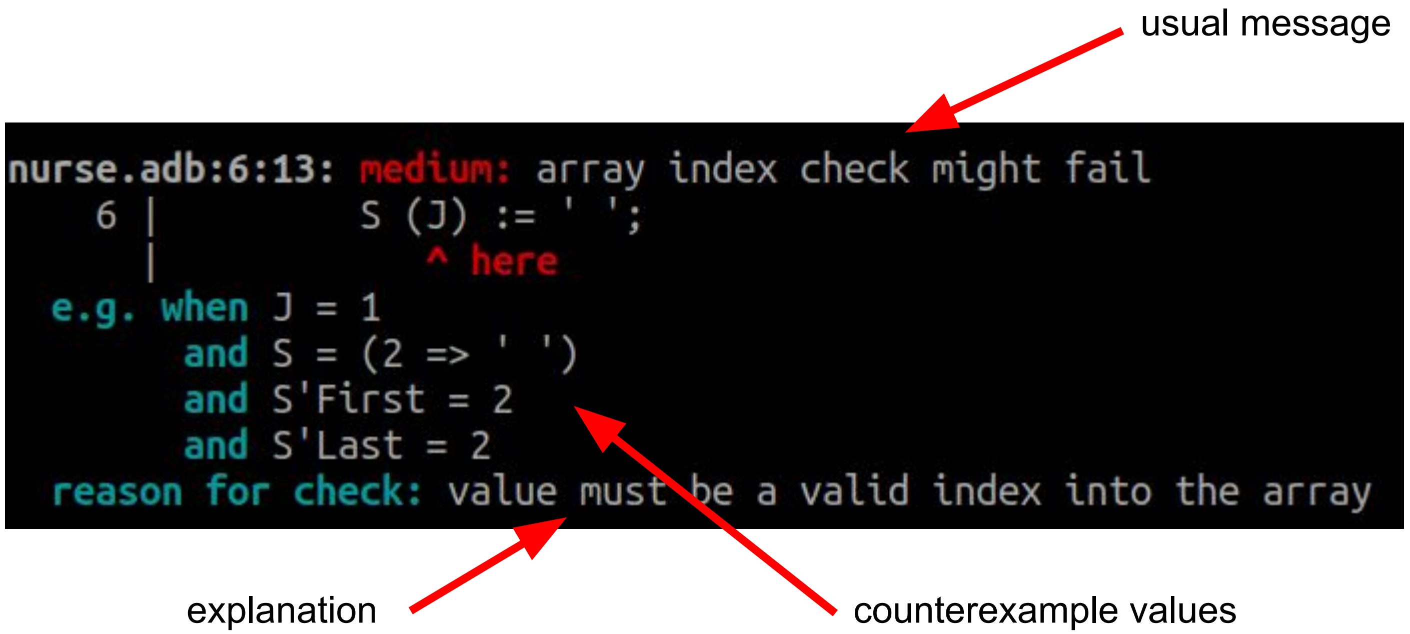
```
file.adb:6:13: medium: array index check might fail
```

The Nurse: Providing First Aid

Step 1: understand the immediate cause of the problem

The Nurse: Providing First Aid

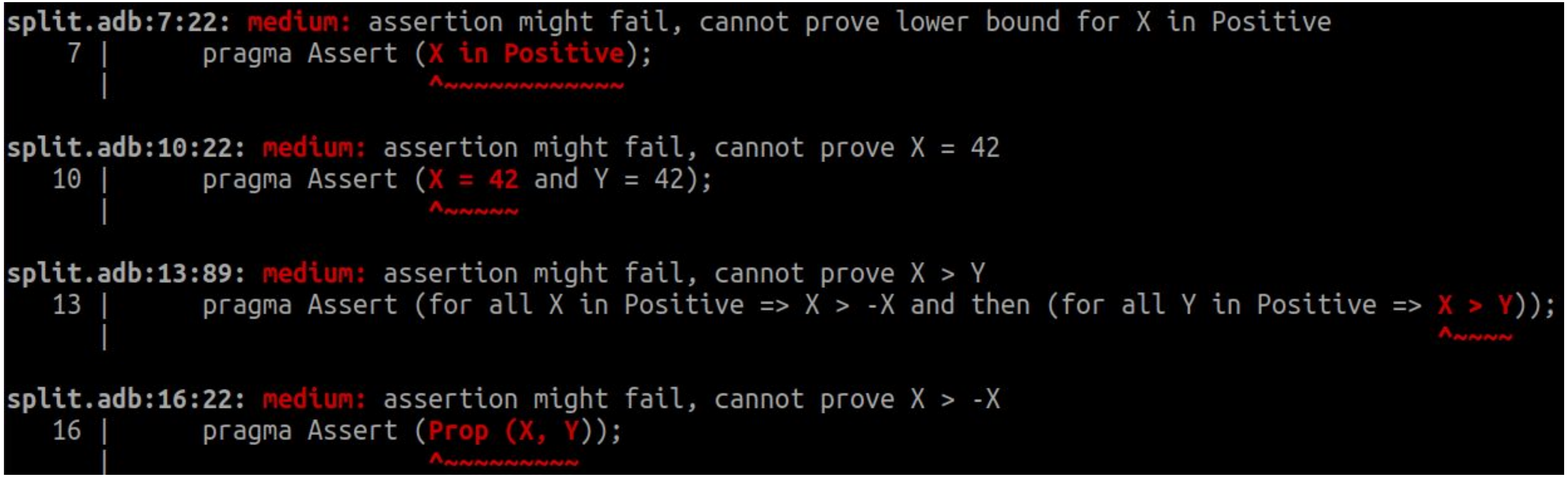
Step 1: understand the immediate cause of the problem



```
nurse.adb:6:13: medium: array index check might fail
    6 |          S (J) := ' ';
      |             ^ here
  e.g. when J = 1
        and S = (2 => ' ')
        and S'First = 2
        and S'Last = 2
  reason for check: value must be a valid index into the array
```

The Nurse: Providing First Aid

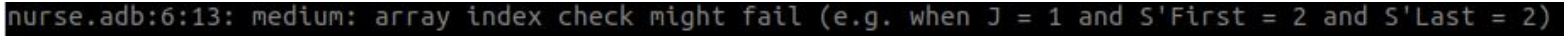
```ada
pragma Assert (X in Positive);
pragma Assert (X = 42 and Y = 42);
pragma Assert (for all X in Positive => X > -X and then (for all Y in Positive => X > Y));
function Prop (X, Y : Natural) return Boolean is (X > -X and X > Y);
pragma Assert (Prop (X, Y));
```



```
split.adb:7:22: medium: assertion might fail, cannot prove lower bound for X in Positive
    7 |      pragma Assert (X in Positive);
      |                     ^~~~~~~~~~~~~

split.adb:10:22: medium: assertion might fail, cannot prove X = 42
   10 |      pragma Assert (X = 42 and Y = 42);
      |                     ^~~~~~

split.adb:13:89: medium: assertion might fail, cannot prove X > Y
   13 |      pragma Assert (for all X in Positive => X > -X and then (for all Y in Positive => X > Y));
      |                                                                                        ^~~~~

split.adb:16:22: medium: assertion might fail, cannot prove X > -X
   16 |      pragma Assert (Prop (X, Y));
      |                     ^~~~~~~~~~~
```

The Nurse: Providing First Aid

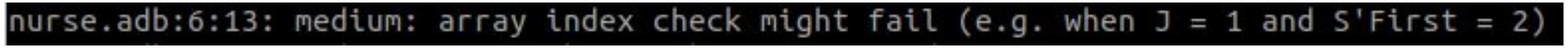
SPARK 16: we get a counterexample! :-)



```
nurse.adb:6:13: medium: array index check might fail (e.g. when J = 1 and S'First = 2 and S'Last = 2)
```

SPARK 17: we lost the counterexample :-\

SPARK 18: we regain a counterexample :-?



```
nurse.adb:6:13: medium: array index check might fail (e.g. when J = 1 and S'First = 2)
```

SPARK 21: we have a better counterexample \o/ (previous slide)

SPARK 22: we lost again the counterexample…

SPARK 23: … but we already recovered it in the next release!

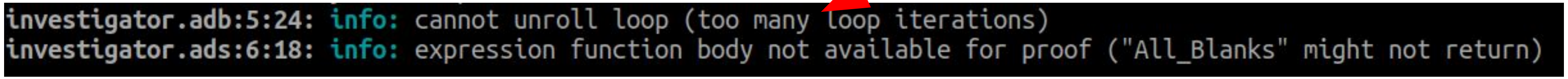
The Investigator: Looking for Probable Cause

Step 2: understand the root cause of the problem

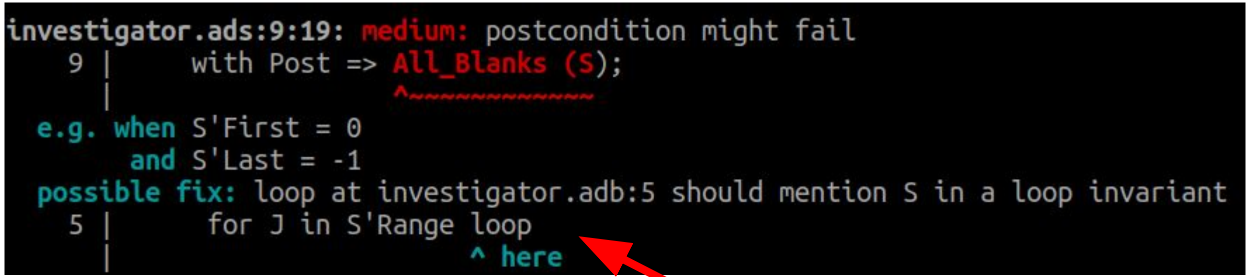
The Investigator: Looking for Probable Cause

Step 2: understand the root cause of the problem

internal information



```
investigator.adb:5:24: info: cannot unroll loop (too many loop iterations)
investigator.ads:6:18: info: expression function body not available for proof ("All_Blanks" might not return)
```



```
investigator.ads:9:19: medium: postcondition might fail
    9 |     with Post => All_Blanks (S);
      |                  ^~~~~~~~~~~~~~
  e.g. when S'First = 0
        and S'Last = -1
  possible fix: loop at investigator.adb:5 should mention S in a loop invariant
    5 |      for J in S'Range loop
      |                 ^ here
```

possible root cause

The Investigator: Looking for Probable Cause

No loop unrolling

info: cannot unroll loop (value of loop bounds / too many loop iterations)

Contract not available

info: *(implicit)* function contract not available for proof ("F" might not return)

info: (*implicit)* function contract might not be available on recursive calls

info: call to "From_Universal_Image" is not handled precisely

Missing information

info: no contextual analysis of "F" (in assertion expression)

info: default initial condition on type "T" not available for proof in an assertion context

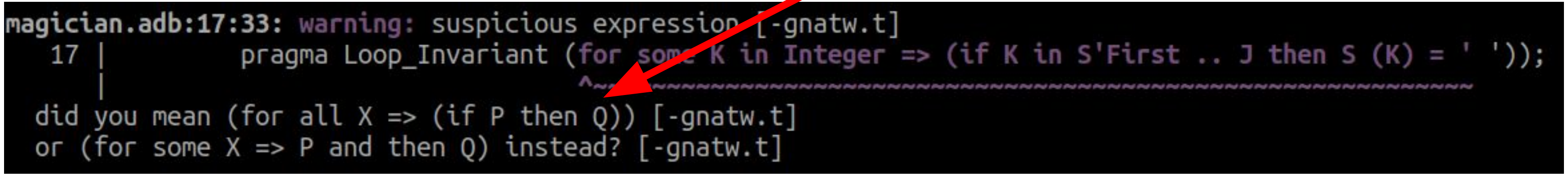
The Magician: Suggesting a Possible Fix

Step 3: fix the problem!

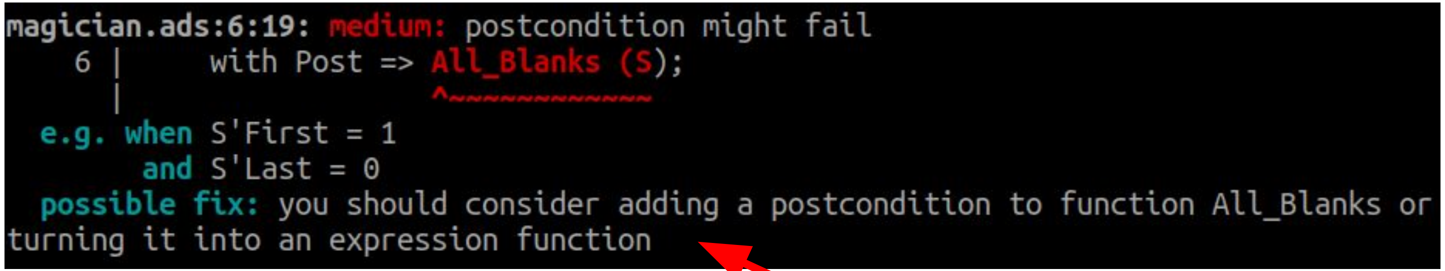
The Magician: Suggesting a Possible Fix

Step 3: fix the problem!

faulty pattern



```
magician.adb:17:33: warning: suspicious expression [-gnatw.t]
   17 |          pragma Loop_Invariant (for some K in Integer => (if K in S'First .. J then S (K) = ' '));
      |                                 ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  did you mean (for all X => (if P then Q)) [-gnatw.t]
  or (for some X => P and then Q) instead? [-gnatw.t]
```



```
magician.ads:6:19: medium: postcondition might fail
    6 |      with Post => All_Blanks (S);
      |                   ^~~~~~~~~~~~~~
  e.g. when S'First = 1
        and S'Last = 0
  possible fix: you should consider adding a postcondition to function All_Blanks or
turning it into an expression function
```

clearly missing information

The Magician: Suggesting a Possible Fix

warning: unused variable "N" in conjunct [-gnatw.t]
warning: consider extracting conjunct from quantified expression [-gnatw.t]

possible fix: use pragma Overflow_Mode or switch -gnato13 or unit Ada.Numerics.Big_Numerics.Big_Integers
possible fix: overlaying object should have an Alignment representation clause
possible fix: use "and then" instead of "and" in precondition

possible fix: add or complete related loop invariants or postconditions
possible fix: subprogram at p.ads:42 should mention V in a precondition
possible fix: add precondition (V in A'First .. A'Last) to subprogram at p.ads:42
possible fix: add precondition (if V >= 0 then W >= Integer'First + V else W <= Integer'Last + V) to subprogram at p.ads:42

Questions!

Can we provide correct counterexamples in more complex cases?

[F-IDE 2021, *"Explaining Counterexamples with Giant-Step Assertion Checking"*]

Can we adapt the tool feedback to the level of expertise of the user?

Can we improve the presentation of Verification Conditions?

[F-IDE 2018, *“Lightweight Interactive Proving inside an Automatic Program Verifier”*]

Can we develop true proof assistants? (closer to Office Clippy than to Coq)

Can the analyzer help the user help the analyzer help the user?