Automatic Guidance for the Formal Verification of High Integrity Ada

Andrew Ireland

Dependable Systems Group
Computing and Electrical Engineering
Heriot-Watt University
Edinburgh

Motivation and Context

- The benefits of formal verification within the development of critical systems are recognized.
- Scalability and usability of formal verification techniques remain key challenges to researchers.
- Automation and high-level interaction will play an important role in meeting challenges.
- Focus here of theorem proving as an enabling technology for source code verification.

Overview

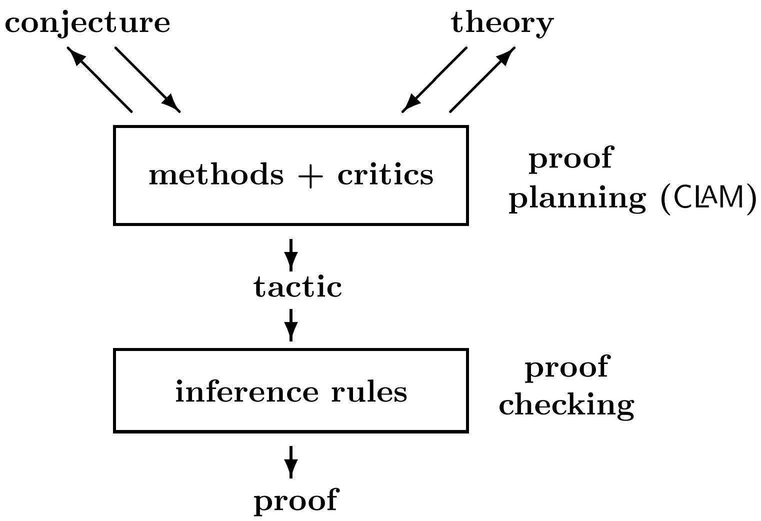
- Proof planning and proof discovery capabilities.
- Proof planning for the SPARK approach to high integrity Ada.
- Current research goals and potential proposals for the future.

Proof Plans

Proof plan = tactics + methods + critics

- **Proof tactics:** prescribe the structure of a proof at the level of the primitive rules of deduction.
- **Proof methods:** given a collection of general-purpose tactics, proof methods guide the construction of a special-purpose tactic for a given conjecture.
- **Proof critics:** when methods fail their associated critics suggest proof patches by analyzing partial success in terms of method preconditions and the overall proof plan instantiation.

Strategic-Deductive Separation



CLAM proof planner was developed within the Mathematical Reasoning Group (University of Edinburgh).

Why Have A Separate Planning Level?

- Method and critic applicability is more constrained than tactic applicability – supports greater **proof automation**.
- Method and critic applicability is informative – supports **high-level interaction**.
- Methods and critics support greater **flexibility** in the use of proof strategies than a conventional script or tactic-based environment.
- Methods and critics promote the **reusability** of proof strategies across different application domains and logics.

Proof by Mathematical Induction

- Proof by mathematical induction is required if you want to reason about recursion, iteration, parameterized designs, ...
- Structural induction on lists:

  Conjecture: $\forall t : list(\tau).\ P(t)$

  gives rise to a base case:

  Goal: $P(nil)$

  and a step case:

  Given: $P(t)$

  Goal: $P(h :: t)$

Note *nil* denotes the empty list and :: is the list constructor.

A List Induction Example

- Conjecture:

$$\forall t : list(\tau).\ rot(len(t), t) = t$$

- Generalized conjecture:

$$\forall t : list(\tau).\forall l : list(\tau).\ rot(len(t), t \mathbin{@} l) = l \mathbin{@} t$$

- Base case:

  Goal: $rot(len(nil), nil \mathbin{@} l) = l \mathbin{@} nil$

- Step case:

  Given: $\forall l' : list(\tau).\ rot(len(t), t \mathbin{@} l') = l' \mathbin{@} t$

  Goal: $rot(len(h :: t), (h :: t) \mathbin{@} l) = l \mathbin{@} (h :: t)$

Definitions and Properties

$$
\begin{aligned}
len(nil) &= 0 \\
len(X :: Y) &= 1 + len(Y) \\
nil \mathbin{@} Z &= Z \\
(X :: Y) \mathbin{@} Z &= X :: (Y \mathbin{@} Z) \\
rot(0, Z) &= Z \\
rot(1 + X, nil) &= nil \\
rot(1 + X, Y :: Z) &= rot(X, Z \mathbin{@} (Y :: nil)) \\
(X \mathbin{@} Y) \mathbin{@} Z &= X \mathbin{@} (Y \mathbin{@} Z) \\
X \mathbin{@} (Y :: Z) &= (X \mathbin{@} (Y :: nil)) \mathbin{@} Z
\end{aligned}
$$

Step-case: Deductive Perspective

- **Given:**

$$\forall l' : list(\tau).\ rot(len(t), t \mathbin{@} l') = l' \mathbin{@} t \tag{1}$$

- **Goal:**

$$
\begin{aligned}
rot(\underline{len(h :: t)}, \underline{(h :: t) \mathbin{@} l}) &= \underline{l \mathbin{@} (h :: t)} \\
\underline{rot(1 + len(t), h :: (t \mathbin{@} l))} &= (l \mathbin{@} (h :: nil)) \mathbin{@} t \\
rot(len(t), \underline{(t \mathbin{@} l) \mathbin{@} (h :: nil)}) &= (l \mathbin{@} (h :: nil)) \mathbin{@} t \\
rot(len(t), t \mathbin{@} (l \mathbin{@} (h :: nil))) &= (l \mathbin{@} (h :: nil)) \mathbin{@} t
\end{aligned}
$$

Match with (1), l' becomes $(l \mathbin{@} (h :: nil))$

Step-case: Strategic Perspective

- **Given:**

$$\forall l' : list(\tau).\ rot(len(t), t \mathbin{@} l') = l' \mathbin{@} t$$

- **Goal:**

$$rot(len(\boxed{h :: \underline{t}}^{\uparrow}), \boxed{h :: \underline{t}}^{\uparrow} \mathbin{@} \lfloor l \rfloor) \;=\; \lfloor l \rfloor \mathbin{@} \boxed{h :: \underline{t}}^{\uparrow} \qquad \Big\} \text{ identify differences}$$

$$\left.\begin{array}{rcl}
rot(\boxed{1 + \underline{len(t)}}^{\uparrow}, \boxed{h :: \underline{t \mathbin{@} \lfloor l \rfloor}}^{\uparrow}) & = & \left\lfloor \boxed{\underline{l} \mathbin{@} h :: nil}^{\downarrow} \right\rfloor \mathbin{@} t \\
rot(len(t), \boxed{\underline{t \mathbin{@} \lfloor l \rfloor} \mathbin{@} h :: nil}^{\downarrow}) & = & \left\lfloor \boxed{\underline{l} \mathbin{@} h :: nil}^{\downarrow} \right\rfloor \mathbin{@} t \\
rot(len(t), t \mathbin{@} \left\lfloor \boxed{\underline{l} \mathbin{@} h :: nil}^{\downarrow} \right\rfloor) & = & \left\lfloor \boxed{\underline{l} \mathbin{@} h :: nil}^{\downarrow} \right\rfloor \mathbin{@} t
\end{array}\right\} \text{ reduce differences}$$

$$rot(len(t), t \mathbin{@} (l \mathbin{@} h :: nil)) \;=\; (l \mathbin{@} h :: nil) \mathbin{@} t \qquad \Big\} \text{ exploit match}$$

Note: $\lfloor \ldots \rfloor$ delimit position of match variables.

Strategic vs Deductive

- **Strategic:** proof search guided by annotated rewrite rules:

$$\begin{array}{rcl}
len(\boxed{X :: \underline{Y}}^{\uparrow}) & \Rightarrow & \boxed{1 + \underline{len(Y)}}^{\uparrow} \\
rot(\boxed{1 + \underline{X}}^{\uparrow}, \boxed{Y :: \underline{Z}}^{\uparrow}) & \Rightarrow & rot(X, \boxed{\underline{Z} \mathbin{@} (Y :: nil)}^{\downarrow}) \\
\boxed{(\underline{X \mathbin{@} Y}) \mathbin{@} Z}^{\downarrow} & \Rightarrow & X \mathbin{@} \boxed{(\underline{Y} \mathbin{@} Z)}^{\downarrow}
\end{array}$$

- **Deductive:** proof checking involves rewriting (no search):

$$\begin{array}{rcl}
len(X :: Y) & \Rightarrow & 1 + len(Y) \\
rot(1 + X, Y :: Z) & \Rightarrow & rot(X, Z \mathbin{@} (Y :: nil)) \\
(X \mathbin{@} Y) \mathbin{@} Z & \Rightarrow & X \mathbin{@} (Y \mathbin{@} Z)
\end{array}$$

Note: rewriting + meta-data = ripple method

Patterns of Success

Rippling-out:

$$f_1(\ldots, f_n(\boxed{c_1(\underline{\ldots})}^{\uparrow}), \ldots) \qquad \boxed{c_n(\underline{f_1(\ldots, f_n(\ldots), \ldots)})}^{\uparrow}$$

before **after**

Rippling-sideways:

$$f_1(\boxed{c_1(\underline{\ldots})}^{\uparrow}, f_n(\ldots)) \qquad f_1(\ldots, \boxed{c_n(\underline{f_n(\ldots)})}^{\downarrow})$$

before **after**

Rippling-in:

$$\boxed{c_n(\underline{f_1(\ldots, f_n(\ldots), \ldots)})}^{\downarrow} \qquad f_1(\ldots, f_n(\boxed{c_1(\underline{\ldots})}^{\downarrow}), \ldots)$$

before **after**

Note: ripple method is automatic and terminating

Ripple Method Applicability

$$rot(\boxed{1 + \underline{len(t)}}^{\uparrow}, \boxed{h :: (\underline{t \mathbin{@} \lfloor l \rfloor})}^{\uparrow}) \quad = \quad \ldots \quad \textbf{before}$$

$$rot(len(t), \boxed{(\underline{t \mathbin{@} \lfloor l \rfloor}) \mathbin{@} (h :: nil)}^{\downarrow}) \quad = \quad \ldots \quad \textbf{after}$$

Method preconditions:

1. *There exists differences between goal and given,* e.g.

$$rot(\boxed{1 + \underline{len(t)}}^{\uparrow}, \boxed{h :: (\underline{t \mathbin{@} \lfloor l \rfloor})}^{\uparrow})$$

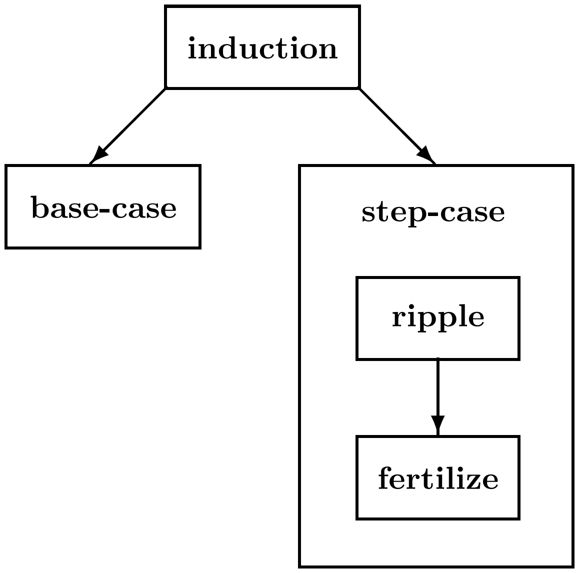
2. *and a matching wave-rule,* e.g.

$$rot(\boxed{1 + \underline{X}}^{\uparrow}, \boxed{Y :: \underline{Z}}^{\uparrow}) \Rightarrow rot(X, \boxed{\underline{Z} \mathbin{@} (Y :: nil)}^{\downarrow})$$

3. *and any conditions attached to the rewrite are provable,*
4. *and any inward differences can potentially be eliminated,* e.g.

$$\ldots \boxed{(\underline{t \mathbin{@} \lfloor l \rfloor}) \mathbin{@} (h :: nil)}^{\downarrow} \ldots$$

A Proof Plan for Induction



General-purpose Tactics for Induction

- `base_case(Base):`

  `Base` ≡
  `(...eval_def(Pos, Rule)...) then elementary`

- `step_case(Step):`

  `Step` ≡ `ripple(Waves) then fertilize(IndHyp)`

  `Waves` ≡ `...wave(Pos, [Rule, Dir])...`

- `induction(IndRule, Var:Typ)`

A Special-Purpose Tactic for Induction

- **Conjecture:**

$$\forall t : list(nat).\ rot(len(t), t) = t$$

- **Composite tactic** (generated automatically):

```
generalize(...) then
   induction([v0::t],[t:nat list]) then
      [base_case(... then
             induction([v1::l],[l:nat list]) then
                [base_case(... then
                    elementary(...)),
                 step_case(ripple(...) then
                              fertilize(...) then
                                 elementary(...))]),
       step_case(ripple(...) then
                    fertilize(...))]
```


Automatic Conjecture Generalization

- **Given:**

$$rot(len(t), t) = t$$

- **Goal:**

$$rot(len(\boxed{h :: \underline{t}}^{\uparrow}), \boxed{h :: \underline{t}}^{\uparrow}) = \boxed{h :: \underline{t}}^{\uparrow}$$

$$\underbrace{rot(\boxed{1 + \underline{len(t)}}^{\uparrow}, \boxed{h :: \underline{t}}^{\uparrow})}_{\textbf{blocked}} = \boxed{h :: \underline{t}}^{\uparrow}$$

Failure explained in terms of precondition 4 of rippling, *i.e.*

"and any inward differences can potentially be eliminated"

$$rot(len(t), \boxed{\underline{\ t\ }\ @\ (h :: nil)}^{\downarrow}) = \ldots$$

Note: no match variable to eliminate inward directed difference.

Patch: Plan Guides Goal Refinement

- **Given:**

$$\forall l' : list(\tau).\ rot(len(t), F_1(t, l')) = G_1(t, l')$$

- **Goal:**

$$rot(len(\boxed{h :: \underline{t}}^{\uparrow}), F_1(\boxed{h :: \underline{t}}^{\uparrow}, \lfloor l \rfloor)) = G_1(\boxed{h :: \underline{t}}^{\uparrow}, \lfloor l \rfloor) \quad \Big\} \text{ identify differences}$$

$$\left.\begin{array}{l}
rot(\boxed{1 + \underline{len(t)}}^{\uparrow}, \boxed{h :: \underline{t \mathbin{@} F_2(\boxed{h :: \underline{t}}^{\uparrow}, \lfloor l \rfloor)}}^{\uparrow}) \ = \ \ldots \\
rot(len(t), \boxed{\underline{t \mathbin{@} F_2(\boxed{h :: \underline{t}}^{\uparrow}, \lfloor l \rfloor)} \mathbin{@} h :: nil}^{\downarrow}) \ = \ \ldots \\
rot(len(t), t \mathbin{@} \boxed{\underline{F_2(\boxed{h :: \underline{t}}^{\uparrow}, \lfloor l \rfloor)} \mathbin{@} h :: nil}^{\downarrow}) \ = \ \ldots
\end{array}\right\} \text{ reduce differences}$$

Note: $F_1, G_1, \ldots$ denote second-order meta variables.

Patch: Plan Guides Goal Refinement

- **Given:**

$$\forall l' : list(\tau).\ rot(len(t), t \mathbin{@} F_2(t, l')) = G_1(t, l')$$

- **Goal:**

$$\left.\begin{array}{lcl}
\ldots & = & \ldots \\
\ldots & = & G_1(\boxed{h :: \underline{t}}^{\uparrow}, \lfloor l \rfloor) \\
\ldots & = & \boxed{\underline{G_2(\boxed{h :: \underline{t}}^{\uparrow}, \lfloor l \rfloor)} \mathbin{@} h :: nil}^{\downarrow} \mathbin{@} t
\end{array}\right\} \text{ reduce differences}$$

$$rot(len(t), t \mathbin{@} (l \mathbin{@} h :: nil)) \ = \ (l \mathbin{@} h :: nil) \mathbin{@} t \quad \Big\} \text{ coerce match}$$

$$\begin{array}{ll}
F_1 \equiv \lambda x.\lambda y.\ x \mathbin{@} F_2(x, y) & G_1 \equiv \lambda x.\lambda y.\ G_2(x, y) \mathbin{@} x \\
F_2 \equiv \lambda x.\lambda y.\ y & G_2 \equiv \lambda x.\lambda y.\ y
\end{array}$$

Dependable Systems Group

Automatic Conjecture Generalization

- In order to prove:

$$\forall t : list(\tau).\ rot(len(t), t) = t$$

- We need to synthesize and prove:

$$\forall t : list(\tau).\forall l : list(\tau).\ rot(len(t), t \mathbin{@} l) = l \mathbin{@} t$$

CLAM generates such generalizations via proof critics.

Automatic Lemma Discovery

- In order to prove:

$$\forall t : list(\tau).\forall l : list(\tau).\ rot(len(t), t \mathbin{@} l) = l \mathbin{@} t$$

- We need to synthesize and prove:

$$\begin{aligned} X \mathbin{@} (Y \mathbin{@} Z) &= (X \mathbin{@} Y) \mathbin{@} Z \\ X \mathbin{@} (Y :: Z) &= (X \mathbin{@} (Y :: nil)) \mathbin{@} Z \end{aligned}$$

CLAM generates such lemmata dynamically via proof critics.

Automatic Invariant Discovery

- In order to prove:

$$\{x = \mathcal{X} \wedge y = \mathcal{Y}\}$$
$$r := 1;$$
$$\textbf{while } (y > 0) \textbf{ do}$$
$$\textbf{begin}$$
$$\quad \textbf{if } odd(y) \textbf{ then } r := r * x;$$
$$\quad y := y \; div \; 2;$$
$$\quad x := x * x$$
$$\textbf{end}$$
$$\{r = exp(\mathcal{X}, \mathcal{Y}) \wedge \neg(y > 0)\}$$

- We need to synthesize and prove:

$$r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y})$$

CLAM generates such invariants dynamically via proof critics.

Productive Use of Failure

Precond	Generalization	Case analyze	Induction revision	Lemma discovery
1	$\star$	$\star$	$\star$	$\star$
2	$\star$	$\star$	$\circ$	$\bullet$
3	$\star$	$\bullet$		
4	$\bullet$			

$\star$ = **success** $\circ$ = **partial success** $\bullet$ = **failure**

Proof Critics: Flexibility and Reusability

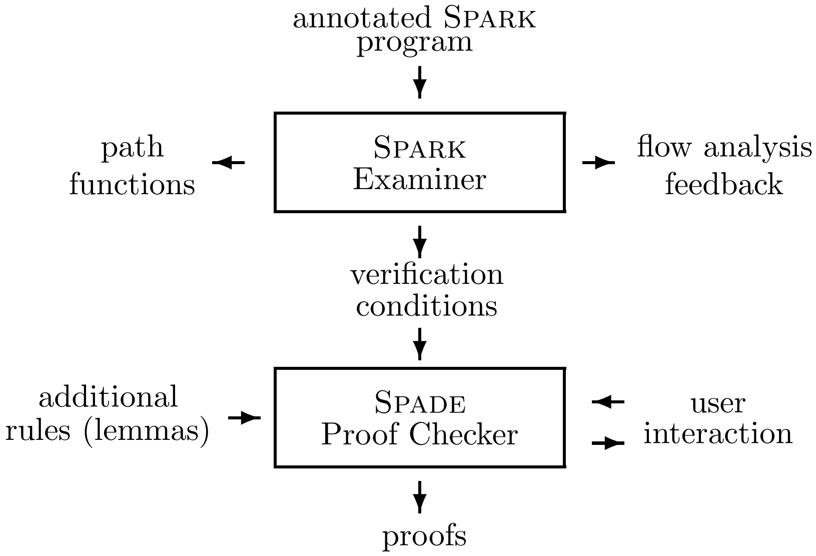
- Conjecture generalization
- Lemma discovery
- Induction revision
- Case analyze
- Replacing constants
- Adding new conjuncts
- Tail-invariants
- Range splitting

EPSRC Research Project

- Investigate the role of proof planning within the SPARK approach to high integrity Ada.
- In collaboration with Praxis Critical Systems and funded by the EPSRC Critical Systems programme – GR/R24081.
- Julian Richardson (Co-investigator),
  Bill Ellis (Research Associate).
- Project web pages:

  `http://www.cee.hw.ac.uk/~air/clamspark/`

Spark Examiner and Spade Proof Checker



Based upon Figure 3.1 from (Barnes, 1997)

Research Goals

- Construct a proof planning wrapper for Spade – will include an interactive proof patching capability.
- Transfer proof critics for conjecture generalization (intermediate assertions) and lemma discovery (rules) for use within Spade.
- Develop proof plans for reasoning about the absence of run-time errors.
- Undertake "industrial strength" case studies as supplied by Praxis.

Potential Research Proposals

Proof informed programming: programming support techniques with a hidden proof planning capability:

- SPARK approach advocates correctness by construction
- Program patterns that ease program proof are advocated
- But currently no mechanization

Proof engineering: proof planning as a basis for managing proof throughout the life-time of a software product:

- Z and SPARK proof perspectives
- Proof plans for reasoning about compliance arguments
- Change management for proof development & maintenance

Note: access to "industrial strength" proofs would be essential.

Summary

- Automation and high-level interaction have an important role to play in addressing the scalability & usability of formal verification techniques.
- Automatic proof discovery and high-level interaction capabilities are major benefits of proof planning approach.
- Time is ripe to investigate these capabilities on "industrial strength" problems – SPARK is an ideal vehicle.

Some References

- A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, A. Smaill
  Rippling: A Heuristic for Guiding Inductive Proofs,
  Journal of Artificial Intelligence, vol 62: 185–253, 1993.
- A. Ireland and A. Bundy.
  Productive Use of Failure in Inductive Proof,
  Journal of Automated Reasoning, 16(1-2):79–111, 1996.
- A. Ireland, M. Jackson, and G. Reid.
  Interactive Proof Critics,
  Formal Aspects of Computing, 11(3):302–325, 1999.
- A. Ireland and J. Stark.
  Proof Planning for Strategy Development,
  Annals of Mathematics and Artificial Intelligence 29(1-4):65-97, 2001.