Rockwell Collins' Evolving FM Methodology

Konrad Slind
Trusted Systems Group
Rockwell Collins

January 25, 2014

Collaborators

- Rockwell Collins: Andrew Gacek, David Hardin, Darren Cofer, John Backes, Luas Wagner
- U. Minnesota: Mike Whalen, Tuan-Hung Pham

Rockwell Collins

Rockwell Collins is a company (NYSE: COL) that makes stuff that gets put into airplanes (commercial and military).

- flight deck, on-board networks
- communication
- information security
- etc.

Has had an FM group since the early 1990s.

Currently, approx. 10 full-time FM people in the group.

FM at Rockwell Collins

We work on internal projects and also compete for external funding from places like NASA and DoD.

We apply a wide range of FM technologies: SMT, model-checking, ITP.

Some are the usual suspects: e.g., PVS, ACL2, HOL4, numSMV, Prover, yices, cvc4, z3, ...

Some are homebrew.

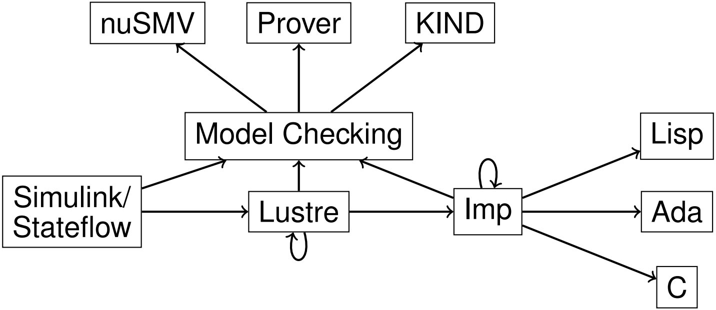
Old Stalwarts: PVS and ACL2

In the early days we used PVS and ACL2 a lot.

- Applications of PVS
  - AAMP5 microcode
  - JEM Java microprocessor
- Applications of ACL2
  - AAMP7 EAL7 MILS Certification by NSA
  - Greenhills Integrity 178B RTOS (EAL6+)

And we still use them a lot.

Old Stalwart: Gryphon



- Written in SML/NJ by Mike Whalen
- Still used a lot

Newcomer verification systems

The following tools have been recently developed by our group:

- OMS (Onboard Maintenance System)
- CAS (Crew Alerting System).
- SPEAR
- AGREE
- Resolute
- Guardol

OMS and CAS

Test case generation via model-checking.

- Test suites often needed to satisfy certification process requirements (DO178B)
- Developing them and running them can be expensive: need to book time on simulator, plus personnel have to be present to check correct physical actions taken
- Basic idea: a counterexample to a negated property is automatically translated into a test for that property
- Properties come from a database of requirements (equations between variables and boolean combinations of primitive tests on wire values)
- Automatically generates high-quality test-suites for free.

SPEAR

There's a need for easier ways to write and check high level requirements.

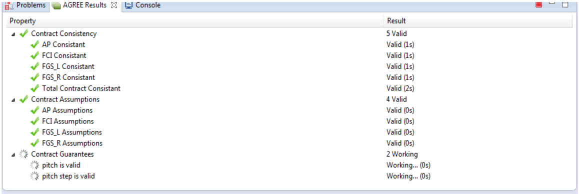
- Natural language is typically used, but formal requirements are useful for analysis
- Idea: provide a set of high level specification patterns in Temporal Logic
- Automatically check that low-level rqts meet high level rqts
- Uses Lustre as the target

AGREE

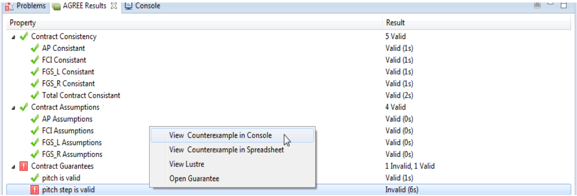
Assume-guarantee reasoning on contracts attached to AADL models.

- Tackling the problem of reasoning about requirements
- No implementations, just contracts
- Built as an extension of the OSATE Eclipse plug-in, which supports AADL.

AGREE at Work



Property	Result
Contract Consistency	5 Valid
AP Consistant	Valid (1s)
FCI Consistant	Valid (1s)
FGS_L Consistant	Valid (1s)
FGS_R Consistant	Valid (1s)
Total Contract Consistant	Valid (2s)
Contract Assumptions	4 Valid
AP Assumptions	Valid (0s)
FCI Assumptions	Valid (0s)
FGS_L Assumptions	Valid (0s)
FGS_R Assumptions	Valid (0s)
Contract Guarantees	2 Working
pitch is valid	Working... (0s)
pitch step is valid	Working... (0s)



Property	Result
Contract Consistency	5 Valid
AP Consistant	Valid (1s)
FCI Consistant	Valid (1s)
FGS_L Consistant	Valid (1s)
FGS_R Consistant	Valid (1s)
Total Contract Consistant	Valid (2s)
Contract Assumptions	4 Valid
AP Assumptions	Valid (0s)
FCI Assumptions	Valid (0s)
FGS_L Assumptions	Valid (0s)
FGS_R Assumptions	Valid (0s)
Contract Guarantees	1 Invalid, 1 Valid
pitch is valid	Valid (1s)
pitch step is valid	Invalid (6s)

AGREE Counterexample

Problems AGREE Results Console

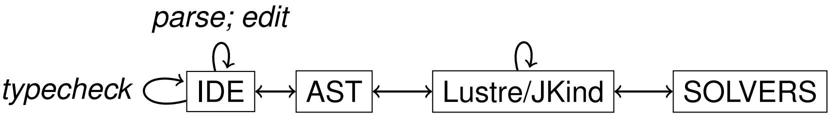
Counterexample

Variables for

Variable Name	0	1	2	3	4	5
{AD L.Pitch.Val}	10/7	16/7	22/7	4	34/7	34
{AD L.Pitch.Valid}	true	true	false	true	true	tr
{AD L Pitch Step Delta Valid}	true	true	true	true	true	tr
{AD R.Pitch.Val}	-3/7	3/7	9/7	15/7	3	3
{AD R.Pitch.Valid}	true	false	true	false	false	fa
{AD R Pitch Step Delta Valid}	true	true	true	true	true	tr
{AP.CSA.CSA Pitch Delta}	0	0	0	0	-2/7	33
{FGS L.GC.cmds.Pitch Delta}	34/7	34/7	34/7	34/7	34/7	34
{FGS L.GC.mds.active}	false	false	false	false	true	tr
{FGS L.LSO.Leader}	2	2	2	2	1	1
{FGS L.LSO.Valid}	true	true	false	true	true	tr
{FGS R.GC.cmds.Pitch Delta}	-3/7	-3/7	-3/7	-3/7	-3/7	-3

Common Aspects

These systems can be seen as instantiations of the following framework:



- IDE generated by **xtext.**
- **JKind** is our Java implementation of a parallel k-induction model checker for Lustre.
- Invokes SMT solvers (yices, z3, cvc4)
- Publically available:

  `https://github/agacek/jkind`

Resolute

Analysis of architectural properties.

- **Resolute** is a theorem prover for structural properties of AADL.
- The system is based on the notion of a *safety case*.
- We are currently applying it to security properties in then context of the HACMS project.
- All the properties are computable, so Resolute is just a way to conveniently write and check properties about, e.g., how components are connected together, etc.
- The safety case is generated in a goal-directed style and the resulting proof tree *is* the assurance case.

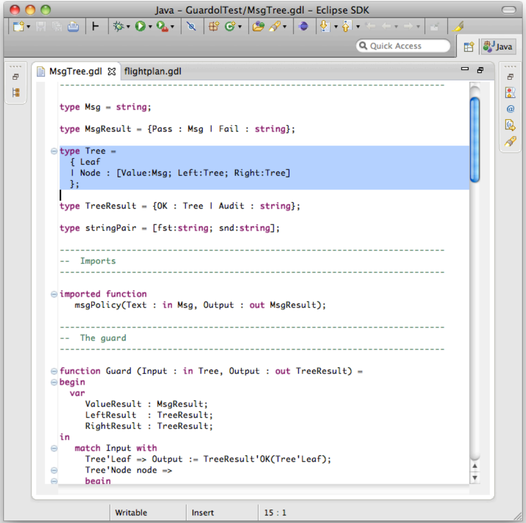
Guardol

A **guard** mediates information sharing between security domains according to a specified policy.

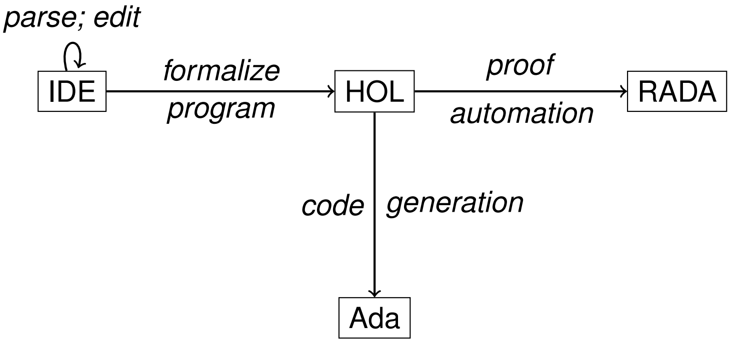
Literally a box on a wire, in many cases.

Guardol is a DSL for guards.

Guardol Example



Guardol Architecture

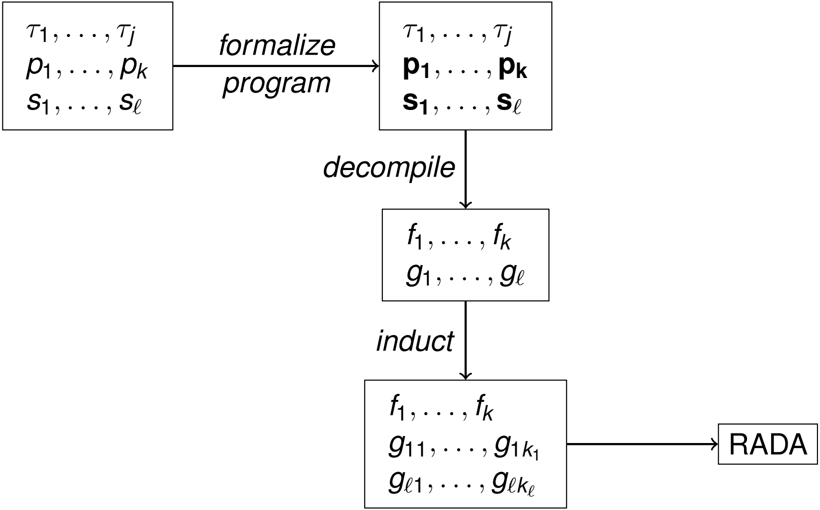


Verification

HOL4 is used as a **semantical conduit** to RADA

- **RADA** is a SMT-based system for reasoning about catamorphisms
- **HOL4** is an implementation of higher order logic.
- We use it to give a semantics to Guardol evaluation
- Decompilation into logic transforms specs about Guardol evaluation to properties of HOL functions
- Induction schemes from the definition of the functions are used to drive the skeleton of the inductive proof

Verification path



Things to Ponder

In the OLD days, the idea was that one worked in a particular proof system and translated problems into it. The semantics stared you in the face.

The NEW view is that one does an ad hoc connection of a UI and some FM tools. So translations are important.

We have found that providing an IDE for the language under consideration is very important. Hence auto-generation of IDEs for the concrete syntax of the object language.

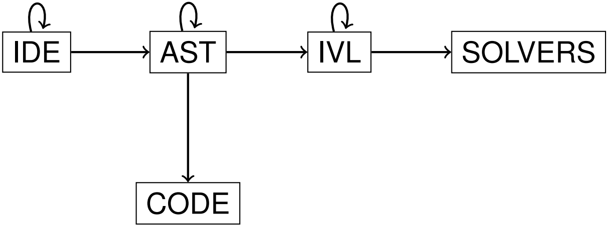
ANP$^{\mathbf{3}}$: **A** **N**ew **P**rover **P**er **P**roject

ANTLR4 grammar example

xtext uses **ANTLR** to generate parsers

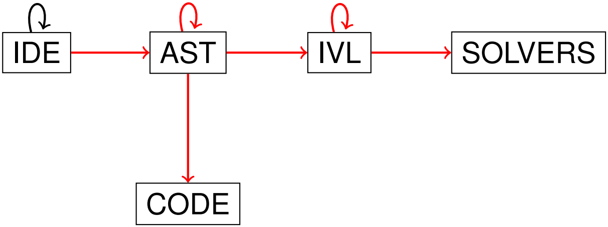
```
expr: ID
    | INT
    | REAL
    | BOOL
    | ID '(' (expr (',' expr)*)? ')'
    | 'not' expr
    | '-' expr
    | expr op=('*' | '/' | 'div') expr
    | expr op=('+' | '-') expr
    | expr op=('<' | '<=' | '>' | '>=' | '=' | '<>') expr
    | expr op='and' expr
    | expr op=('or' | 'xor') expr
    | expr op='=>'<assoc=right> expr
    | expr op='->'<assoc=right> expr
    | 'if' expr 'then' expr 'else' expr
    | '(' expr ')'
```

Things to Ponder

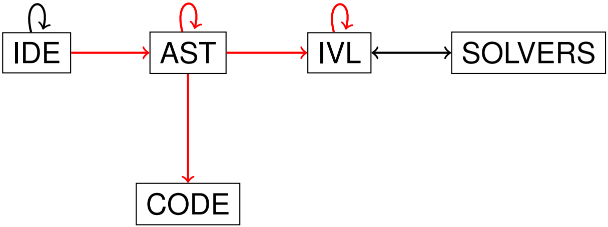


What is the soundness story? The TCB?

Things to Ponder



Translate SMT proofs

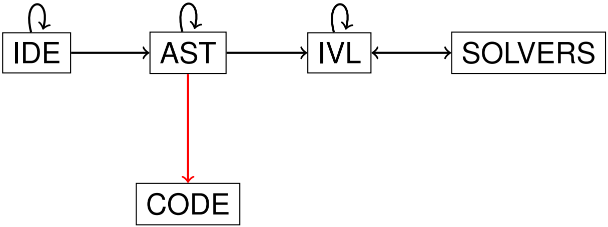


This has been done, but not for all theories.

Not widely adopted at the moment

Can also consider verified solvers (but solvers are usually in C)

Verified Parsing and Translation to IVL

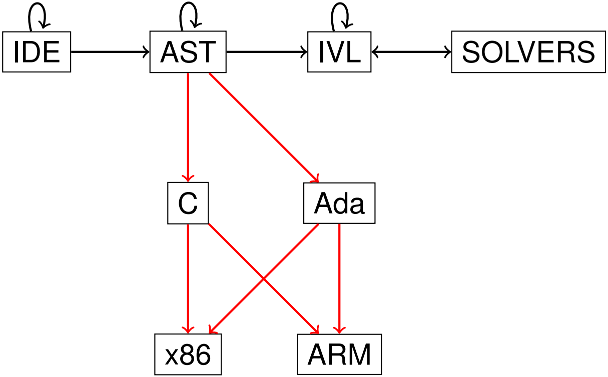


Frontend of cakeML shows this can be done.

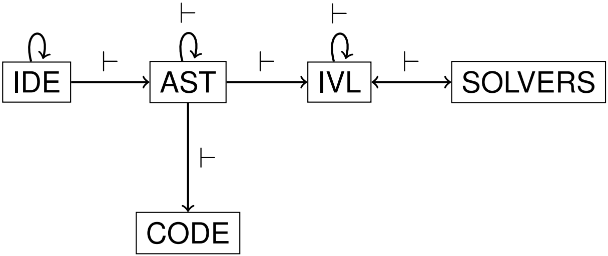
AST $\longrightarrow$ IVL verified translation also done in Guardol.

Verified Code Generation

This is commonly done by translation to source in some useful language like Ada or C. Then off-the-shelf compilers are applied.



An Ideal Setting Maybe



A lot of work to achieve.

Would it be materially better than something with less verification inside?

Related Systems

- Rustan Leino's Dafny/Boogie system
- J.C. Filliâtre's Why3
- Isabelle
- Lem

Summary

We apply a spectrum of FM methods at RC.

Things we think are good ideas:

- Providing IDEs for concrete syntax as a frontend to formal analyses. (With **xtext** this is quite easy to achieve.)
- Using high-level intermediate languages with nice semantics. (Provides a base from which to stage calls to proof procedures and solvers.)

Challenges: verification of the translations

THE END