SPIRAL
www.spiral.net

Proving Hybrid Control Operator Language (HCOL)

Proof Assistant Approach Overview

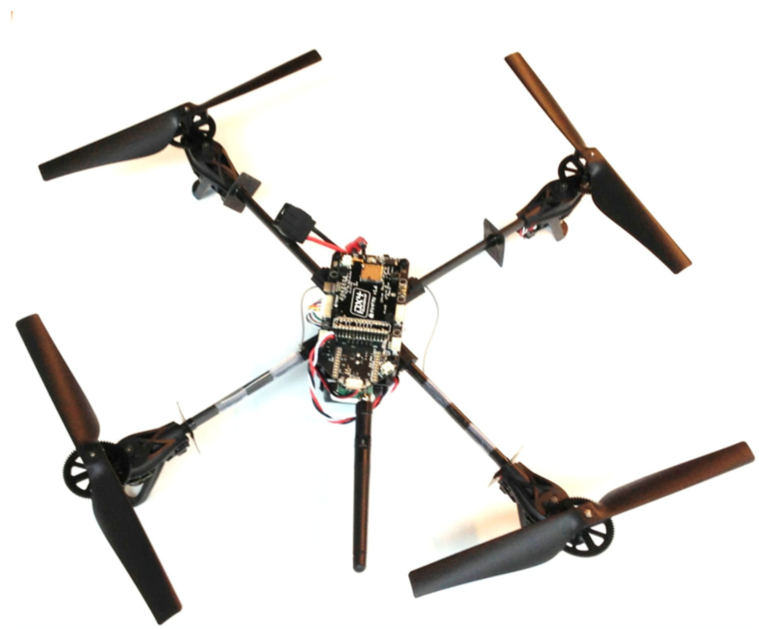
Vadim Zaliva, Franz Franchetti, Jeremy Johnson, David Padua
CMU, Drexel, UIUC

Funded by the DARPA I2O HACMS Program through award FA8750-12-2-0291

2014-11-05

Motivation and Funding ;)

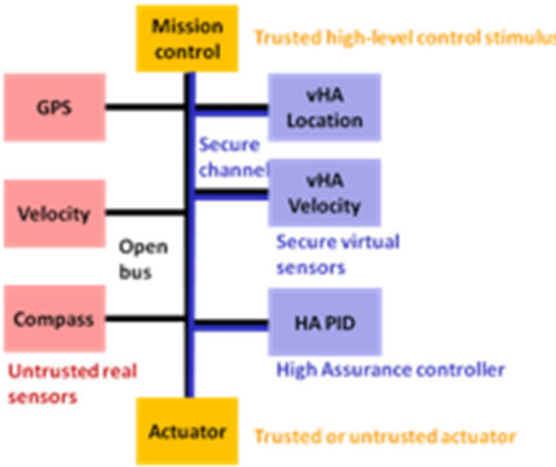
"The goal of the HACMS program is to create technology for the construction of high-assurance cyber-physical systems, where high assurance is defined to mean functionally correct and satisfying appropriate safety and security properties."



Spiral
www.spiral.net

Summary: High Assurance Spiral

High Assurance Abstraction



HA Spiral Architecture

Code Synthesis

$$\mathrm{y}^{t+h} = \left[\mathrm{I}_3 \,|h\, \mathrm{I}_3\right](\mathrm{x}^t \oplus \mathrm{v}^{t+h})$$

```
let(y:=var(TArray(TReal, 3)),
   xv:=var(TArray(TReal, 6)), h := TReal(1/100),
   func([inparam(xv), outparam(y)],
```

Verification and Proofs

$$\mathrm{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$= ?$

$$\sum_{i=0}^{2} \mathrm{e}_i^3 \mathrm{I}_1 (\mathrm{e}_i^3)^\top = [1\,0\,0][1][1\,0\,0]^\top + [0\,1\,0][1][0\,1\,0]^\top +$$

Spiral: Translating a Formula into Code

Input:

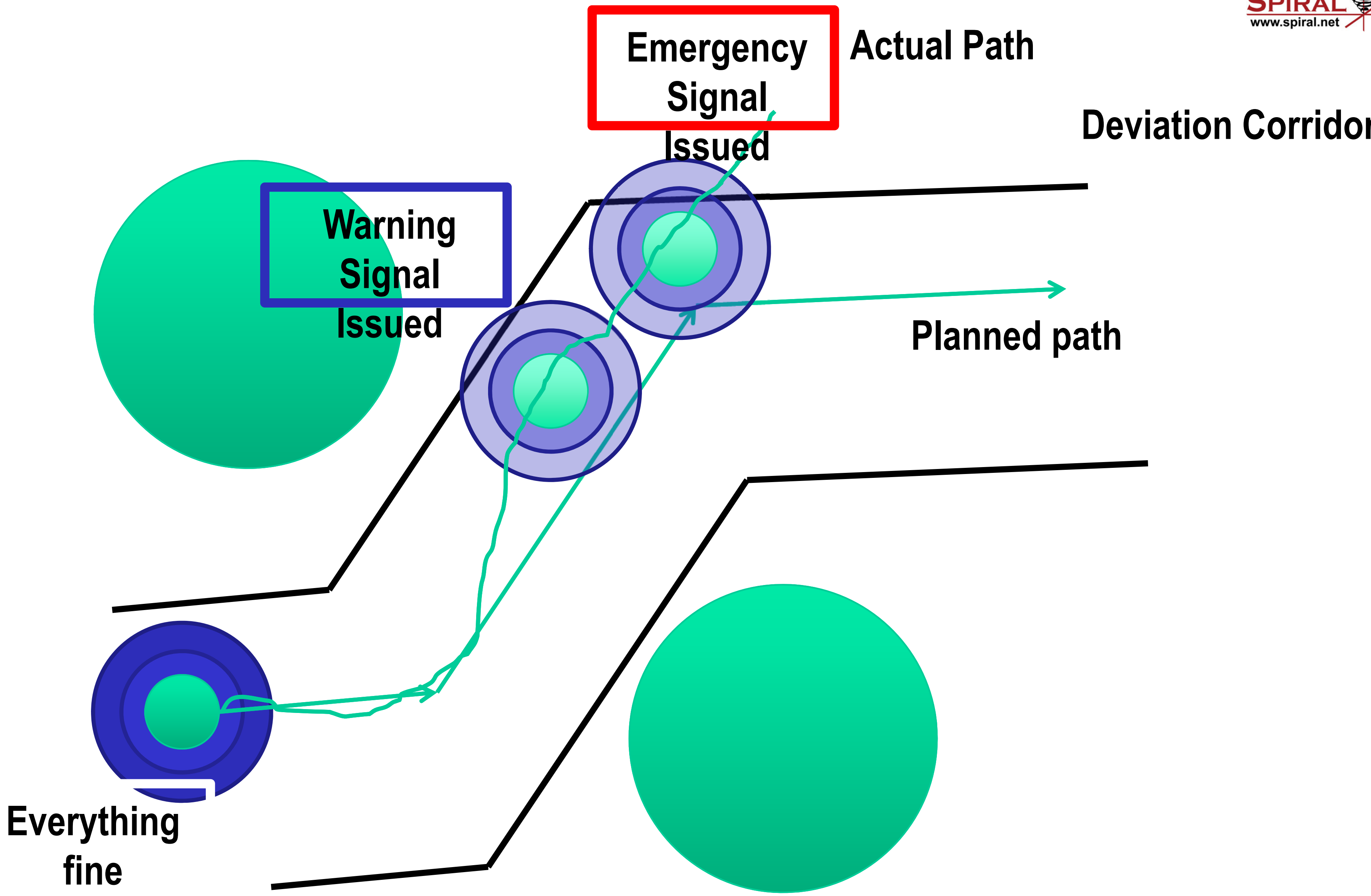
$$\underbrace{\mathrm{DFT}_8}_{\text{double}}$$

OL Formula: $$(\mathrm{DFT}_2 \otimes I_4)\, T_4^8 \left(I_2 \otimes \left((\mathrm{DFT}_2 \otimes I_2)\, T_2^4\, (I_2 \otimes \mathrm{DFT}_2)\, L_2^4\right)\right) L_2^8$$

∑-OL: $$\sum_{j=0}^{3} \left(S_j\, \mathrm{DFT}_2\, G_j\right) \sum_{k=0}^{1} \left(\sum_{l=0}^{1} \left(S_{k,l}\, \mathrm{diag}\left(\mathrm{t}_{k,l}\right) \mathrm{DFT}_2\, G_l\right) \sum_{m=0}^{1} \left(S_m\, \mathrm{diag}\left(\mathrm{t}_m\right) \mathrm{DFT}_2\, G_{k,m}\right)\right)$$

C Code:

```
void sub(double *y, double *x) {
  double f0, f1, f2, f3, f4, f7, f8, f10, f11;
  f0 = x[0] - x[3];
  f1 = x[0] + x[3];
  f2 = x[1] - x[2];
  f3 = x[1] + x[2];
  f4 = f1 - f3;
  y[0] = f1 + f3;
  y[2] = 0.7071067811865476 * f4;
  f7 = 0.9238795325112867 * f0;
  f8 = 0.3826834323650898 * f2;
  y[1] = f7 + f8;
  f10 = 0.3826834323650898 * f0;
  f11 = (-0.9238795325112867) * f2;
  y[3] = f10 + f11;
}
```


Emergency Signal Issued
Actual Path
Deviation Corridor
Warning Signal Issued
Planned path
Everything fine

Minimum Safety Distance

- **Safety constraint from KeYmaera**

  p_o: Position of closest obstacle
  p_r: Position of robot
  v_r: Longitudinal velocity of robot
  A, b, V, ": constants

$$\|p_r - p_o\|_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right)$$

- **Definition as operator**

$$\mathrm{D}_{V,A,b,\varepsilon} : \quad \mathbb{R} \times \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{Z}_2$$

$$(v_r, p_r, p_o) \mapsto \left(p(v_r) < d_\infty(p_r, p_o)\right) \quad \text{with} \quad d_\infty(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|_\infty$$

$$p(x) = \alpha x^2 + \beta x + \gamma$$

$$\alpha = \frac{1}{2b}$$

$$\beta = \frac{V}{b} + \varepsilon\left(\frac{A}{b} + 1\right)$$

$$\gamma = \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon V\right)$$

Modeling Mathematical Objects in HCOL

- **Infinity norm**

$$\|.\|_\infty^n : \mathbb{R}^n \to \mathbb{R}$$
$$(x_i)_{i=0,\ldots,n-1} \mapsto \max_{i=0,\ldots,n-1} |x_i|$$

- **Chebyshev distance**

$$d_\infty^n(.,.) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$
$$(x, y) \mapsto \|x - y\|_\infty^n$$

- **Vector subtraction**

$$(-)_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$$
$$(x, y) \mapsto x - y$$

- **Pointwise comparison**

$$(<)_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{Z}_2^n$$
$$\big((x_i)_{i=0,\ldots,n-1}, (y_i)_{i=0,\ldots,n-1}\big) \mapsto (x_i < y_i)_{i=0,\ldots,n-1}$$

Modeling Mathematical Objects in HCOL (2)

- **Scalar product**

$$< .,. >_n\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

$$\big((x_i)_{i=0,\dots,n-1}, (y_i)_{i=0,\dots,n-1}\big) \mapsto \sum_{i=0}^{n-1} x_i y_i$$

- **Monomial enumerator**

$$(x^i)_n : \mathbb{R} \to \mathbb{R}^{n+1}$$

$$x \mapsto (x^i)_{i=0,\dots,n}$$

- **Polynomial evaluation**

$$P[x, (a_0, \dots, a_n)] : \mathbb{R} \to \mathbb{R}$$

$$x \mapsto a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

HCOL Basic Operators

$$\mathsf{Pointwise}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}^n$$
$$(x_i)_i \mapsto f_0(x_0) \oplus \cdots \oplus f_{n-1}(x_{n-1})$$

$$\mathsf{Pointwise}_{n\times n,f_i} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$$
$$\big((x_i)_i, (y_i)_i\big) \mapsto f_0(x_0, y_0) \oplus \cdots \oplus f_{n-1}(x_{n-1}, y_{n-1})$$

$$\mathsf{Reduction}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_i)_i \mapsto f_{n-1}(x_{n-1}, f_{n-2}(x_{n-2}, f_{n-3}(\ldots f_0(x_0, \mathsf{id}()) \ldots)$$

$$\mathsf{Atomic}_{f(.)} : \mathbb{R} \to \mathbb{R}$$
$$x \mapsto f(x)$$

$$\mathsf{Atomic}_{f(.,.)} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$
$$(x, y) \mapsto f(x, y)$$

$$\mathsf{Scale}_{n,(a,b)\mapsto a\diamond b} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$$
$$\big(\alpha, (x_i)_{i=0,\ldots,n-1}\big) \mapsto (\alpha \diamond x_i)_{i=0,\ldots,n-1}$$

$$\mathsf{Concat}_{m,n} : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^{m+n}$$
$$\big((x_i)_i, (y_i)_i\big) \mapsto \big(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}\big)$$

HCOL Rules

- **Breakdown rules**

$$d_\infty^n(.,.) \rightarrow \|.\|_\infty^n \circ (-)_n$$

$$(\diamond)_n \rightarrow \mathsf{Pointwise}_{n\times n,(a,b)\mapsto a\diamond b}$$

$$\|.\|_\infty^n \rightarrow \mathsf{Reduction}_{n,(a,b)\mapsto \max(|a|,|b|)}$$

$$< .,. >_n \rightarrow \mathsf{Reduction}_{n,(a,b)\mapsto a+b} \circ \mathsf{Pointwise}_{n\times n,(a,b)\mapsto ab}$$

$$P[x,(a_0,\dots,a_n)] \rightarrow < (a_0,\dots,a_n), . > \circ (x^i)_n$$

$$\begin{aligned}(x^i)_n \rightarrow\ & \mathsf{Concat}_{1,n}((1),.) \circ \mathsf{Scale}_{n,(a,b)\mapsto ab} \\ & \circ \left(\mathrm{e}_1^{1\times n}(.)[-]\,\mathrm{I}_n(.)\right) \circ (x^i)_{n-1} \quad \text{for} \quad n>1\end{aligned}$$

$$(x^i)_1 \rightarrow \mathsf{Concat}_{1,1}((1),.)$$

$$\mathrm{I}_n(.) \rightarrow \mathsf{Pointwise}_{n,a\mapsto a}$$

HCOL Expansion

- **HCOL Operator Definition**

$$\mathsf{SafeDist}_{V,A,b,\varepsilon} : \mathbb{R} \times \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{Z}_2$$

$$(v_r, p_r, p_o) \mapsto \left(\|p_r - p_o\|_\infty > \frac{v_r^2}{2b} + V\frac{v_r}{b} + \left(\frac{A}{b} + 1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon(v_r + V)\right)\right)$$

- **HCOL Breakdown Rule**

$$\mathsf{SafeDist}_{V,A,b,\varepsilon}(.,.,.) \to \Big(P[x,(a_0,a_1,a_2)](.) < d_\infty^2(.,.)\Big)(.,.,.)$$

with $a_0 = \frac{1}{2b},\ a_1 = \frac{V}{b} + \varepsilon\left(\frac{A}{b}+1\right),\ a_2 = \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon V\right)$

- **Fully Expanded HCOL Expression**

$$\begin{aligned}\mathsf{SafeDist}_{V,A,b,\varepsilon} \to \mathsf{Atomic}_{(x,y)\mapsto x<y} \\ \circ \Big(\Big(\mathsf{Reduction}_{3,(x,y)\mapsto x+y} \circ \mathsf{Pointwise}_{3,x\mapsto a_i x} \\ \circ\, \mathsf{Concat}_{1,2}((1),.) \circ \mathsf{Scale}_{2,(x,y)\mapsto xy} \\ \circ \left(\mathsf{e}_1^{1\times 2}(.)[-]\ \mathsf{Pointwise}_{2,x\mapsto x}\right) \circ \mathsf{Concat}_{1,1}((1),.)\Big) \\ \times \left(\mathsf{Reduction}_{n,(x,y)\mapsto \max(|x|,|y|)} \circ \mathsf{Pointwise}_{n\times n,(x,y)\mapsto x-y}\right)\Big)\end{aligned}$$

A case for formal verification

- Paper: *"Finding and Understanding Bugs in C Compilers"* [Yang et al. PLDI 2011]
- Approach: Test C compilers using generated random test cases.
- 8 C compilers where tested. 325(!) bugs where found in total. GCC: 79 bugs/25 critical, LLVM 202 bugs.
- CompCert: 10 bug in **unverified** front-end.

"What sets CompCert C apart from any other production compiler, is that it is formally verified, using machine-assisted mathematical proofs, to be exempt from miscompilation issues. In other words, the executable code it produces is proved to behave exactly as specified by the semantics of the source C program."

Csmith conclusions

"The striking thing about our CompCert results is that the middle end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task. The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users."

[Yang et al. PLDI 2011]

Proof Assistants

Automated proof assistants help to automate proof by performing series of user-defined steps (applications of proof tactics) which either discharge the proof goal or split it into several sub-goals. Each step said to modify the proof state and each tactic must be based on trusted inference methods. Examples of PAs:

- Isabelle
- Coq
- Agda
- ACL2
- PVS
- Lego
- Nuprl

Currently we will consider the two most popular ones: Isabelle and Coq.

Isabelle

- Based on Higher-Order Logic
- Written in *Standard ML* and *Scala*
- New tactics could be written in ML
- Rich library of various mathematical proofs

Coq

- Based on Calculus of Inductive Constructions
- Research dates back to 1985.
- Fist release of Coq published in 1989. In active development since.
- Written in *OCaml*
- Supports Dependent Types
- Includes DSL (called *Ltac* to build new tactics).
- Popular in Certified Programs development community (e.g. *CompCert,* Princeton, MIT).
- Meets the "*de Bruijn criterion*"

de Bruijn criterion

Proof assistants satisfy the “de Bruijn criterion” when they produce proof terms in small kernel languages, even when they use complicated and extensible procedures to seek out proofs in the first place. . . . To believe a proof, we can ignore the possibility of bugs during search and just rely on a (relatively small) proof-checking kernel that we apply to the result of the search.[2].

Curry-Howard correspondence

- Observed around 1958 by Haskell Curry and William Howard
- Shows the direct relationship between computer programs and mathematical proofs.

Types ←→Propositions

Programs ←→ Proofs

- Proving a proposition (a Type) is building an inhabitant of this type. Checking a proof is type-checking the proof term
- In Coq, the user types in tactics, guiding the proof development system to construct a proof-term. At the end, this term is type checked and the type is compared with the original goal.

Coq - Vectors

A vector is a list of size *n* whose elements belong to a set *A*. The recursive parameterized inductive type definition is as follows:

$$\texttt{Inductive}\ t\ A : nat \rightarrow \texttt{Type} :=$$
$$|nil : t\ A\ 0$$
$$|cons : \forall\ (h{:}A)\ (n{:}nat),\ t\ A\ n \rightarrow t\ A\ (S\ n).$$

Informally here we specify that a list could be constructed using one of two constructors: nil constructs an empty list (with zero length), while cons pre-pends an element to existing list increasing its length

Coq - Semirings

A *semiring* is defined as a data structure:

```
Structure semiring :=
    Semiring {
        Asring : Type;
        asr_0 : Asring;
        asr_1 : Asring;
        asr_add : Asring → Asring → Asring;
        asr_mul : Asring → Asring → Asring
    }.
```

Coq – Semiring properties

Declaring an instance of this structure to comply to semi ring theory the following properties must be provided (proven):

```
Record semi_ring_theory : Prop := mk_srt {
    SRadd_0_l : ∀ n, 0 + n == n;
    SRadd_comm : ∀ n m, n + m == m + n ;
    SRadd_assoc : ∀ n m p, n + (m + p) == (n + m) + p;
    SRmul_1_l : ∀ n, 1*n == n;
    SRmul_0_l : ∀ n, 0*n == 0;
    SRmul_comm : ∀ n m, n×m == m×n;
    SRmul_assoc : ∀ n m p, n*(m×p) == (n×m)*p;
    SRdistr_l : ∀ n m p, (n + m)*p == n×p + m×p
  }.
```

Defining operators in Coq

```
Parameter sr : semiring.

Definition SimpleReduction {A B: Type} (f: A->B->B) {n} (id:B) (a: t
A n) : B :=  fold_right f a id.

Definition SimplePointWise2 {A B C: Type} (f: A->B->C) {n} (a: t A n)
(b: t B n) : t C n := map2 f a b.

Definition ScalarProd {n}  (a b: t (Atype sr) n) : Atype sr :=
  fold_right (asr_add sr) (map2 (asr_mul sr) a b) (asr_0 sr).

Fixpoint EvalPolynomial {n} (a: t (Atype sr) n) (x:Atype sr) : Atype
sr  :=
  match a with
      nil => (asr_0 sr)
    | cons a0 p a' => (asr_add sr) a0 ((asr_mul sr) x (EvalPolynomial
a' x))
  end.
```

Proving Scalar Product Breakdown Rule

Now we can try to prove the following HCOL breakdown rule:

$$< .,. >_n \rightarrow \mathrm{Reduction}_{n,(a,b)\mapsto a+b} \circ \mathrm{Pointwise}_{n\times n,(a,b)\mapsto ab}$$

Proof in Coq:

```
Theorem breakdown_ScalarProd: forall (n:nat)
(a v: t (Atype sr) n),
   ScalarProd a v = ((SimpleReduction (asr_add sr)
(asr_0 sr)) ∘ (SimplePointWise2 (asr_mul sr) a)) v.
Proof.
  intros.
  unfold compose, SimplePointWise2, SimpleReduction,
ScalarProd.
  reflexivity.
Qed.
```

The proof walkthrough (step 1/7)

Proof State:

```
1 subgoals, subgoal 1 (ID 13)

  ============================
   forall (n : nat) (a v : t (Atype sr) n),
   ScalarProd a v =
   (SimpleReduction (asr_add sr) (asr_0 sr) ∘ SimplePointWise2
(asr_mul sr) a) v
```

Proof progress:

```
intros.
unfold compose.
unfold SimplePointWise2.
unfold SimpleReduction.
unfold ScalarProd.
reflexivity.
```

The proof walkthrough (step 2/7)

Proof State:

```
1 subgoals, subgoal 1 (ID 16)

  n : nat
  a : t (Atype sr) n
  v : t (Atype sr) n
  ============================
   ScalarProd a v =
   (SimpleReduction (asr_add sr) (asr_0 sr) ∘ SimplePointWise2
(asr_mul sr) a)
     v
```

Proof progress:

```
intros.
unfold compose.
unfold SimplePointWise2.
unfold SimpleReduction.
unfold ScalarProd.
reflexivity.
```

The proof walkthrough (step 3/7)

Proof State:

```
1 subgoals, subgoal 1 (ID 17)

  n : nat
  a : t (Atype sr) n
  v : t (Atype sr) n
  ============================
   ScalarProd a v =
   SimpleReduction (asr_add sr) (asr_0 sr)
     (SimplePointWise2 (asr_mul sr) a v)
```

Proof progress:

~~intros.~~

~~unfold compose.~~

```
unfold SimplePointWise2.
unfold SimpleReduction.
unfold ScalarProd.
reflexivity.
```

The proof walkthrough (step 4/7)

Proof State:

```
1 subgoals, subgoal 1 (ID 18)

  n : nat
  a : t (Atype sr) n
  v : t (Atype sr) n
  ============================
   ScalarProd a v =
   SimpleReduction (asr_add sr) (asr_0 sr) (map2 (asr_mul sr) a v)
```

Proof progress:

~~intros.~~
~~unfold compose.~~
~~unfold SimplePointWise2.~~
unfold SimpleReduction.
unfold ScalarProd.
reflexivity.

The proof walkthrough (step 5/7)

Proof State:

```
1 subgoals, subgoal 1 (ID 19)

  n : nat
  a : t (Atype sr) n
  v : t (Atype sr) n
  ============================
   ScalarProd a v =
   fold_right (asr_add sr) (map2 (asr_mul sr) a v) (asr_0 sr)
```

Proof progress:

~~intros.~~
~~unfold compose.~~
~~unfold SimplePointWise2.~~
~~unfold SimpleReduction.~~
unfold ScalarProd.
reflexivity.

The proof walkthrough (step 6/7)

Proof State:

```
1 subgoals, subgoal 1 (ID 20)

  n : nat
  a : t (Atype sr) n
  v : t (Atype sr) n
  ============================
   fold_right (asr_add sr) (map2 (asr_mul sr) a v) (asr_0 sr) =
   fold_right (asr_add sr) (map2 (asr_mul sr) a v) (asr_0 sr)
```

Proof progress:

~~intros.~~

~~unfold compose.~~

~~unfold SimplePointWise2.~~

~~unfold SimpleReduction.~~

~~unfold ScalarProd.~~

reflexivity.

The proof walkthrough (step 7/7)

Proof State:

Proof progress:

```
~~intros.~~
~~unfold compose.~~
~~unfold SimplePointWise2.~~
~~unfold SimpleReduction.~~
~~unfold ScalarProd.~~
~~reflexivity.~~
```

Coq – Code Extraction (Haskell example)

```
map2 :: (a1 -> a2 -> a3) -> Nat -> (T a1) -> (T a2) -> T a3
map2 g n v1 v2 = case v1 of {
   Nil -> case v2 of {
     Nil -> Nil;
     Cons h n0 t -> unsafeCoerce (\_ -> Prelude.error "absurd case")};
   Cons h1 n0 t1 -> case v2 of {
     Nil -> Prelude.error "absurd case";
     Cons h2 n1 t2 -> Cons (g h1 h2) n1 (map2 g n1 t1 t2)}}

fold_right :: (a1 -> a2 -> a2) -> Nat -> (T a1) -> a2 -> a2
fold_right f n v b = case v of {
   Nil -> b;
   Cons a n0 w -> f a (fold_right f n0 w b)}

scalarProd :: Nat -> (T Asring) -> (T Asring) -> Asring
scalarProd n a b =
  fold_right (asr_add realring) n (map2 (asr_mul realring) n a b)
    (asr_0 realring)
```

Scalar Product Breakdown Rule in Isabelle

```
lemma rule_ScalarProd :
  fixes v::"'val::{comm_ring_1} list"
  fixes a::"'val::{comm_ring_1} list"
  assumes
  aa:"a≠[]" and  vv:"v≠[]" and lav: "length a = length v"
  shows "ScalarProd a v = (Reduction (op +) 0 ∘ PointWise (length a) (λ x i . x*(nth a i)))
v"
proof -
  have l: "ScalarProd a v = [foldl (op +) 0 (map (%(x,y). x*y) (zip a v))]" by (simp add:
iprod_def foldl_listsum)
  have r1:"hd ((Reduction (op +) 0 ∘ PointWise (length a) (λ x i . x*(nth a i))) v) =
    foldl op + 0 (zipwith0 (λx i. x * a ! i) v (natrange (length a)))" by simp
  have "(zipwith0 (λx i. x * a ! i) v (natrange (length a))) = map (%(x,y).  (λx i. x * a !
i) x y) (zip v (natrange (length a)))"
    using lav zipwith0_map by (metis natrange_len)
  also have "... =  map (%(x,y).  x*y) (zip v (map (nth a) (natrange (length a))))"
    by (metis dnr lav natrange_map_all)
  finally have "(zipwith0 (λx i. x * a ! i) v (natrange (length a))) =  map (%(x,y).  x*y)
(zip v a)" by (metis natrange_map_all)
  hence "(Reduction (op +) 0 ∘ PointWise (length a) (λ x i . x*(nth a i))) v =
    [foldl op + 0 (map (%(x,y).  x*y) (zip v a))]" using single_element_list r1 by simp
  thus ?thesis using l lav map_mul_comm by fastforce
qed
```

Summary: Isabelle (vs. Coq)

Pros:

- Powerful automatic solving methods
- Based on widely understood mainstream HOL
- Rich library of mathematical theories

Cons:

- Less transparent decision strategies
- No dependent types
- Does not readily produce evidence trial (does not satisfy "*de Bruijn criterion*")

Summary: Coq (vs. Isabelle)

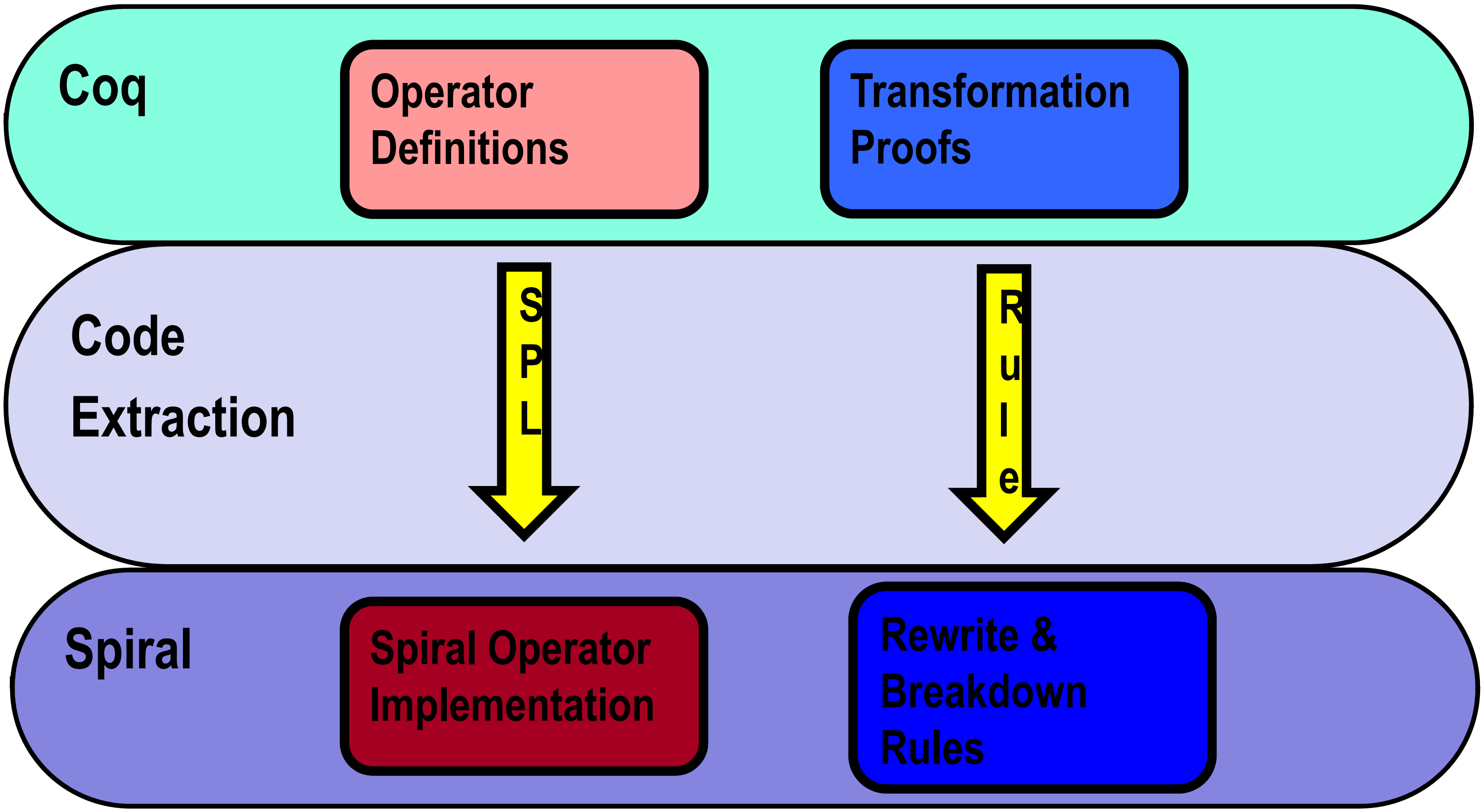
Pros:

- Supports dependent types
- Powerful type system
- Fine-grained, easy to understand tactics
- Produces evidence trial (does satisfy "*de Bruijn criterion*")

Cons:

- Based on lesser know *Calculus of Inductive Computations*
- The system perhaps better intuitively understood by Computer Scientists than by Mathematicians

Proving Transformations



Next Steps: Proving Code Generation

- Slides above were dealing with Axiomatic proofs of the HCOL operator language transformation.
- Next steps to prove:
  - *HCOL* ➢ *i-Code* transformation
  - i-Code ➢ i-Code
  - i-*Code* ➢ “C” code generation
  - “C” code ➢ machine code compilation
- Related:
  - Operational Semantics
  - *CompCert* “C” compiler
  - CLite language
  - The Semantics of x86 Multiprocessor Programs (*x86-TSO*, *x86-CC*)
  - A Formal Model of *IEEE* Floating Point Arithmetic

Further Reading

- Y. Bertot, P. Casteran. *"Coq'Art: Interactive theorem proving and program development*."
  Springer Verlag, 2004.
- T. Nipkow, L. Paulson, M. Wenzel. *"Isabelle/HOL: a proof assistant for higher-order logic."* Springer, 2002

- A. Chlipala. "*Certified programming with dependent types"*
  The MIT Press, 2013
- F. Baader and T. Nipkow, *"Term Rewriting and All That"*, Cambridge University Press, 1998.