# Formal Verification of HCOL Rewriting

Vadim Zaliva and Franz Franchetti, Carnegie Mellon University

Carnegie Mellon | SPIRAL www.spiral.net

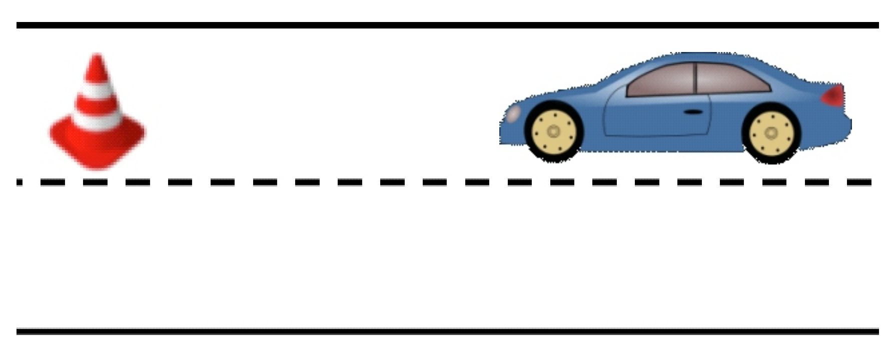
## Project's Goal and High Level Approach

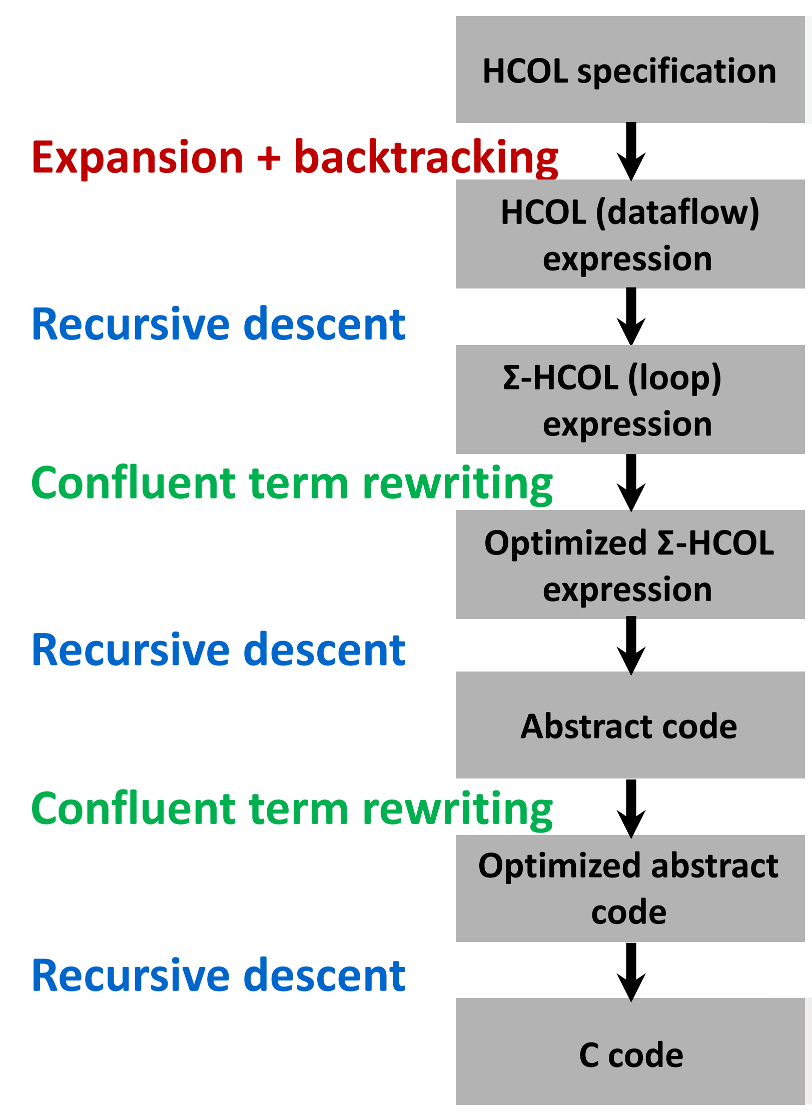
### Approach:

- Vehicular control system is specified in HCOL language
- HCOL specification is transformed to the code via a series of steps
- Transformation steps are formally verified in Coq proof assistant
- SPIRAL-Synthesized and formally verified code deployed on a robot.

### Goal:

To synthesize executable code for the control system of a robot satisfying certain safety and security properties and to produce machine-checkable proofs assuring that this code implements functional specification.






**Mathematical specification**

$$\text{SafeDist}_{V,A,b,\varepsilon}(.,.,.) \to \left(P[x,(a_0,a_1,a_2)](.) < d^2_\infty(.,.)\right)(.,.,.)$$

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

**Final code**

```
int dwmonitor(float  *X, double  *D) {
    __m128d u1, u2, u3, u4, u5, u6, u7, u8 , x1, x10,
    int w1;
    unsigned _xm = _mm_getcsr();
    _mm_setcsr(_xm & 0xffff0000 | 0x0000dfc0);
    u5 = _mm_set1_pd(0.0);
    u2 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN
    u1 = _mm_set_pd(1.0, (-1.0));
    for(int i5 = 0; i5 <= 2; i5++) {
        x6 = _mm_addsub_pd(_mm_set1_pd((DBL_MIN + DBL_MIN
        x1 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
        x2 = _mm_mul_pd(x1, x6);
        x3 = _mm_mul_pd(_mm_shuffle_pd(x1, x1, _MM_SHUFF
        x4 = _mm_sub_pd(_mm_set1_pd(0.0), _mm_min_pd(x3,
        u3 = _mm_add_pd(_mm_max_pd(_mm_shuffle_pd(x4, x4,
        u5 = _mm_add_pd(u5, u3);
        x7 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
        x8 = _mm_mul_pd(x7, u2);
        x9 = _mm_mul_pd(_mm_shuffle_pd(x7, x7, _MM_SHUFFLE2(0
        x10 = _mm_sub_pd(_mm_set1_pd(0.0), _mm_min_pd(x9, x8)
        u1 = _mm_add_pd(_mm_max_pd(_mm_shuffle_pd(x10, x10,
```

## HCOL Basic Operators

$\text{Pointwise}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}^n$

$(x_i)_i \mapsto f_0(x_0) \oplus \cdots \oplus f_{n-1}(x_{n-1})$

$\text{Pointwise}_{n\times n,f_i} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$

$\left((x_i)_i,(y_i)_i\right) \mapsto f_0(x_0,y_0) \oplus \cdots \oplus f_{n-1}(x_{n-1},y_{n-1})$

$\text{Reduction}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}$

$(x_i)_i \mapsto f_{n-1}(x_{n-1}, f_{n-2}(x_{n-2}, f_{n-3}(\ldots f_0(x_0, \text{id}())\ldots)$

$\text{Atomic}_{f(.)} : \mathbb{R} \to \mathbb{R}$

$x \mapsto f(x)$

$\text{Atomic}_{f(.,.)} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

$(x,y) \mapsto f(x,y)$

$\text{Scale}_{n,(a,b)\mapsto a\circ b} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$

$\left(\alpha,(x_i)_{i=0,\ldots,n-1}\right) \mapsto (\alpha \circ x_i)_{i=0,\ldots,n-1}$

$\text{Concat}_{m,n} : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^{m+n}$

$\left((x_i)_i,(y_i)_i\right) \mapsto \left(x_0,\ldots,x_{n-1},y_0,\ldots,y_{n-1}\right)$

## HCOL Rewriting Rules

$d^n_\infty(.,.) \longrightarrow \|.\|^n_\infty \circ (-)_n$

$(\circ)_n \longrightarrow \text{Pointwise}_{n\times n,(a,b)\mapsto a\circ b}$

$\|.\|^n_\infty \longrightarrow \text{Reduction}_{n,(a,b)\mapsto \max(|a|,|b|)}$

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

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

$(x^i)_n \longrightarrow \text{Concat}_{1,n}((1),.) \circ \text{Scale}_{n,(a,b)\mapsto ab} \circ \left(e_1^{1\times n}(.)[-]I_n(.)\right) \circ (x^i)_{n-1} \quad \text{for} \quad n > 1$

$(x^i)_1 \longrightarrow \text{Concat}_{1,1}((1),.)$

$I_n(.) \longrightarrow \text{Pointwise}_{n,a\mapsto a}$

## HCOL Formalization

### Syntax:

An HCOL expression can be represented by an Abstract Syntax Tree (AST). A subset of the language syntax could be defined in Coq using the following *inductive type*:

```
Inductive HOperator: nat→ nat → Type :=
|HOReduction : ∀ m (f: A → A →) 
  '{pF: !Proper ( (=) ==> (=) ==> (=) ) f } (id:A), HOperator m 1
|HOPointWise : ∀ n (f: A → A →A)
  '{pF: !Proper ( (=) ==> (=) ==> (=) ) f }, HOperator (n+n) n
|HOScalarProd: ∀ {k:nat}, HOperator (k+k) 1
|HOEvalPolynomial: ∀ {n} (a:vector A n), HOperator 1 1
|HOCompose: ∀ m {k} n, HOperator k n → HOperator m k →
  → HOperator m n.
```

### Semantics:

The semantics of is defined via an evaluation function, which takes an HOperator object and an input vector and returns the resulting vector:

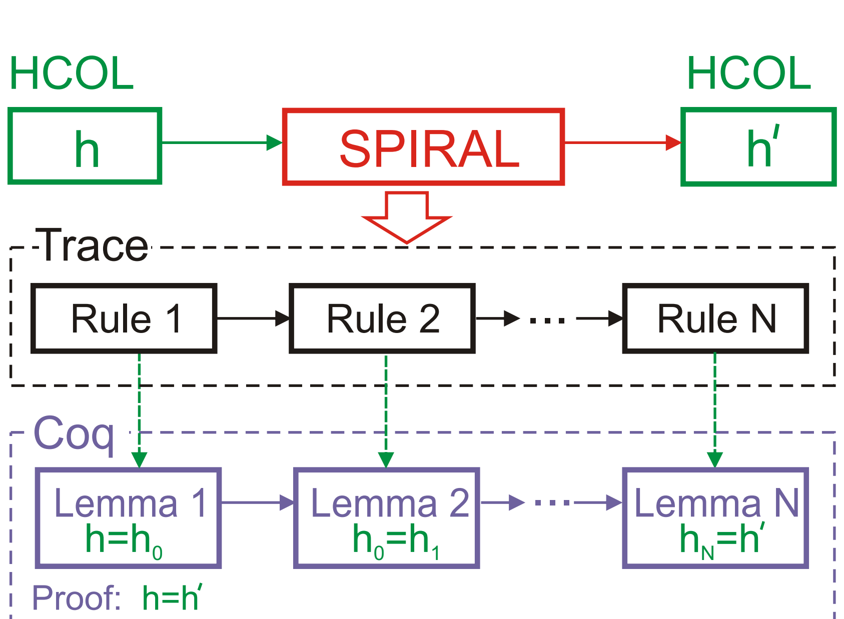
```
evalHCOOL: ∀ {m n}, HOperator m n → vector A m → vector A n.
```

### Operators' definition:

Example definitions of Polynomial operator:

```
Fixpoint EvalPolynomial {n} '{SemiRing A}
      (a: vector A n) (x:A) : A :=
   match a with
        nil ⇒ 0
      | cons a0 p a' ⇒ a0 + (x × (EvalPolynomial a x ))
   end
```

## Proving HCOL Rewriting




- Using Coq Proof Assistant
- Syntax: Inductive type for HCOL expressions
- Semantics: evaluation
- Equivalence: extensionality
- Rewriting rules as lemmas
- "Translation validation" – proving sequence of rule applications from SPIRAL trace.

## Rewriting Rules as Lemmas

We express each rule as a lemma stating equality of two operators.

For example:

```
Lemma breakdown_ScalarProd:
∀ {h:nat},
  HOScalarProd h =
  HOCompose _ _
      (HOReduction _ (+) 0)
      (HOPointWise _ (.*.) ).
```

We defined operator extensional equality:

```
Global Instance HCOL_equiv {i o: nat}: Equiv (HOperator i o) :=
fun a b ⇒ ∀ (x: vector A i), evalHCOL a x = evalHCOL b x.
```

Informally: two operators "a" and "b" are equal if for any input vector "x" the values of (evalHCOL a x) and (evalHCOL b x) are also equal.

## Results and Future Directions

We completed Axiomatic proofs of the HCOL operator language transformations:

- ☑ 7 breakdown rules
- ☑ 76 Lemmas
- ☑ 2,138 lines of Coq code

Next steps to prove:

- ❑ HCOL ➢ Σ-HCOL
- ❑ Σ-HCOL transformations
- ❑ Σ-HCOL ➢ i-Code
- ❑ i-Code ➢ "C" code generation
- ❑ "C" code ➢ machine code compilation

FMCAD 2015