Software Specialization as Applied to Computational Algebra

Pouya Larjani

McMaster University

April 3, 2013

Introduction

- Background: Gaussian Elimination
- Generating optimized solvers for algebraic problems
- Gröbner basis
- Generating a solver
- Buchberger's Algorithm
- Applications

Specializations of Gröbner Bases

- Linear polynomials: Gaussian Elimination
- Univariate polynomials: Euclidean Algorithm
- Special encoding in exponents: Integer Programming
- F_2: Binary Decision Diagrams
- Other special encodings: Graph colouring, geometric theorem proving, ...

Solver Generator

$$
\begin{aligned}
&{}_{p}|\mathrm{GBSolver}|_{f} \quad \models \\
&\quad p = (\mathrm{DB}, \mathrm{BK}, \mathrm{PA}, \mathrm{IP}, \mathrm{PC}, \mathrm{SS}, \mathrm{ES}, \mathrm{SP}, \mathrm{NR}, \mathrm{RS}, \mathrm{CF}, \mathrm{OP}) \ \wedge \\
&\quad \mathrm{DB} \ |\!\models \ \mathrm{Trace} \ \wedge \\
&\quad \mathrm{BK} \in \mathrm{BasisKind} \ \wedge \\
&\quad \mathrm{PA} \ |\!\models \ \mathrm{PolynomialAlgebra} \ \ C, \ \ M, \ \ T, \ \ \mathsf{Poly} \ \wedge \\
&\quad \mathrm{IP} \ |\!\models \ \mathrm{Input} \ \ \mathsf{Poly}, \ \ \mathsf{Inp} \ \wedge \\
&\quad \mathrm{PC} \ |\!\models \ \mathrm{Container} \ \ \mathsf{Poly}, \ \ \mathsf{Idx}, \ \ \mathsf{Cnt} \ \wedge \\
&\quad \mathrm{SS} \ |\!\models \ \mathrm{SelectionStrategy} \ \ \mathsf{Idx}, \ \ \mathsf{Cnt}, \ \ \mathsf{Sel} \ \wedge \\
&\quad \mathrm{ES} \ |\!\models \ \mathrm{ExpansionStrategy} \ \ \mathsf{Idx}, \ \ \mathsf{Cnt}, \ \ \mathsf{Sel}, \ \ \mathsf{Exp} \ \wedge \\
&\quad \mathrm{SP} \ |\!\models \ \mathrm{SPoly} \ \ \mathsf{Poly}, \ \ \mathsf{Idx}, \ \ \mathsf{Cnt}, \ \ \mathsf{Sp} \ \wedge \\
&\quad \mathrm{NR} \ |\!\models \ \mathrm{NormalRemainder} \ \ \mathsf{Poly}, \ \ \mathsf{Cnt}, \ \ \mathsf{Nr} \ \wedge \\
&\quad \mathrm{RS} \ |\!\models \ \mathrm{ReductionStrategy} \ \ \mathsf{Idx}, \ \ \mathsf{Cnt}, \mathsf{Rs} \ \wedge \\
&\quad \mathrm{CF} \ |\!\models \ \mathrm{CanonicalForm} \ \ \mathsf{Poly}, \ \ \mathsf{Cnt}, \ \ \mathsf{Cf} \ \wedge \\
&\quad \mathrm{OP} \ |\!\models \ \mathrm{Output} \ \ \mathsf{Poly}, \ \ \mathsf{Out} \ \ \rightarrow \\
&\quad \bar{f} \in \ulcorner \mathsf{Inp} \ \ \rightarrow \ \ \mathsf{Out} \urcorner \ \wedge \\
&\quad \forall_{\sigma \in \Sigma} \ \ {}_{i}|[\![f]\!]\sigma|_{o} \quad \models \\
&\qquad i \subset C[\vec{x}] \ \ \rightarrow \\
&\qquad \bar{o} \subset C[\vec{x}] \ \ \wedge \ \ \langle i \rangle = \langle \bar{o} \rangle \ \ \wedge \ \ \mathrm{Gröbner}(\bar{o}) \ \wedge \\
&\qquad \mathrm{BK} \in \{\mathrm{MinimalBasis}, \mathrm{ReducedBasis}\} \ \ \Rightarrow \ \ \mathrm{Minimal}(\bar{o}) \ \wedge \\
&\qquad \mathrm{BK} = \mathrm{ReducedBasis} \ \ \Rightarrow \ \ \mathrm{Reduced}(\bar{o})
\end{aligned}
$$

Code Generation

- Meta programming
- Semantics of code generation in F#: Quote, Eval, and Splice
- Extension to allow mixed-stage variables:

$$\ulcorner \textbf{let}\ t = \llcorner a \lrcorner\ \textbf{in}\ \llcorner f\ \ulcorner t \urcorner \lrcorner \urcorner$$

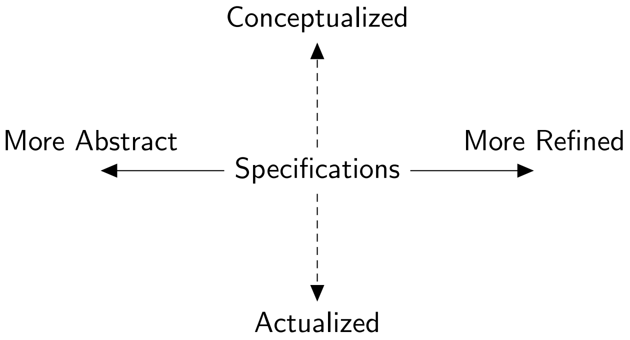
- Code generation as a Domain-Specific Language (DSL)
- Computation Expressions in F#

Codegen DSL

```
let reduceGen (G:Expr<seq<Polynomial>>) = codegen {
    use G' = List.Empty()
    for g in G do
        let n = scalar g (div one (LC g))
        let! m = Control.Let (Ref.Ref n)
        for p in G do
            yield! IfU (Bool.neq p g)
                (codegen {
                    let r = mod (Ref.Deref m) p
                    yield Ref.Assign m r
                })
        yield List.Add G' (Ref.Deref m)
    return G'
}
```

Specifications

- Defining behaviour of systems
- Language for describing specifications
- Adding or removing details: Refinement vs. Abstraction
- Conceptualization vs. Actualization



Formal Specifications

- Predicates and states
- Specification as a predicate: $[\![P]\!]\sigma$
- Software specification with pre and post conditions

$$\sigma\{\![P \rightharpoondown Q]\!\}\sigma' \equiv [\![P]\!]\sigma \wedge [\![Q]\!]\sigma;\sigma'$$

- Programs are state transformers, $m : \Sigma \rightarrow \Sigma$
- Program as an actualization (implementation) of a specification

$$m \Vdash P \rightharpoondown Q \;\equiv\; \forall_{\sigma\in\Sigma}\; [\![P]\!]\sigma \Rightarrow \sigma\{\![P \rightharpoondown Q]\!\}m\cdot\sigma$$

Refinement and Specialization

- Refinement restricts the models (programs) that satisfy a specification

$$S \sqsubseteq S' \equiv \forall_{\sigma,\sigma' \in \Sigma}\ \sigma\{[S']\}\sigma' \Rightarrow \sigma\{[S]\}\sigma'$$

- Adding conjuncts or removing disjuncts both refine a specification
- Specialization $P \rightharpoonup Q \trianglerighteq P' \rightharpoonup Q'$ means that $P \rightharpoonup Q \sqsubseteq P' \rightharpoonup Q'$ and

$$\forall_{\sigma,\sigma'}\ [\![P']\!]\sigma \wedge \sigma\{[P \rightharpoonup Q]\}\sigma' \Rightarrow \sigma\{[P' \rightharpoonup Q']\}\sigma'$$

Module Specifications

- Aspects and Features
- Module interface

$$\begin{array}{l} \text{moduleinterface:} \\ \quad \text{variable}_1 : \; spec_1 \in \mathbb{P} \\ \quad \vdots \\ \quad \text{program}_1 : \; softspec_1 \in \mathbb{S} \\ \quad \vdots \end{array}$$

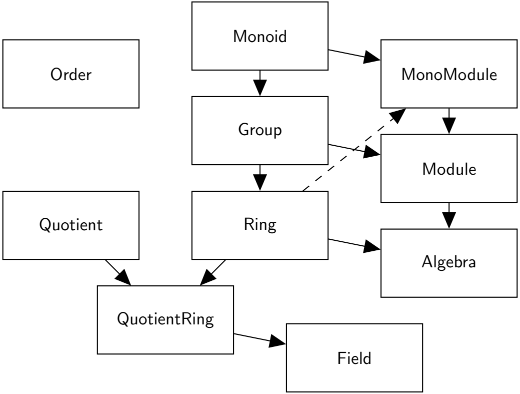
- Module implementation: $M \mid\models I$, if:

$$\forall_{\text{program}_i \in I} \; M.\text{program}_i \mid\models I.softspec_i \; \wedge$$
$$\forall_{\sigma \in \Sigma} \{[\text{Inv}(I)]\}\sigma \Rightarrow \{[\text{Inv}(I)]\} M.\text{program}_i \cdot \sigma$$

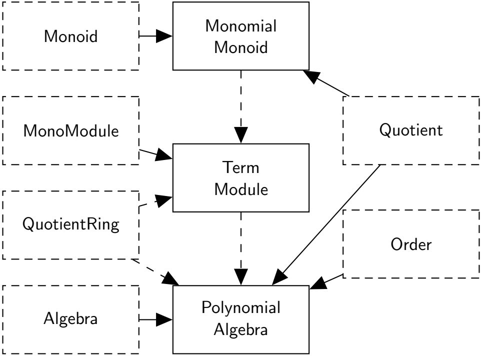
Algebraic Modules

- Specify the modules for computer algebra
- Methods of implementing algebra modules: Type refinements, type guarantees, type inclusions, value restrictions, grounded value conditions, universal equalities, and value profiles
- Module generators
- Polynomial algebra modules: Monomials, terms, orderings, and polynomials

Algebra Modules



Polynomial Modules

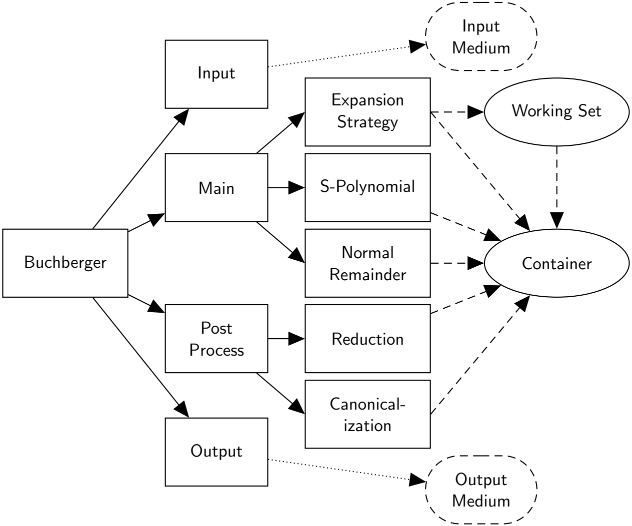


Modular Decomposition of Buchberger's Algorithm

Specified and implemented following seven categories of modules:

(1) Computational Algebra (feature) module: Polynomial Algebra
(2) Control (aspect) modules: BA generator, Main Algorithm, Post Process
(3) I/O (aspect) modules
(4) Storage (feature) modules: Working Set and Polynomial Container
(5) Processing (aspect) modules: S-Polynomial, Normal Remainder, Expansion Strategy
(6) Post-processing (aspect) modules: Reduction and Canonicalization
(7) Trace generation (aspect) module

Generator's Modules



- Code generator produces a final, specialized algorithm for finding Gröbner bases
- Inputs all of the modules defined above to generate the code:

```
val GBSolver :
 Trace * BasisKind * PolynomialAlgebra<'a,'b,'c,'d> *
 Input<'d,'e> * Container<'d,'f,'g> * WorkingSet<'h,'g,'i> *
 ExpansionStrategy<'f,'g,'i,'j> * SPoly<'d,'h,'g,'k> *
 NormalRemainder<'d,'g,'l> * ReductionStrategy<'f,'g,'m> *
 CanonicalForm<'d,'g,'n> * Output<'d,'o> ->
 Value<('e -> 'o)>
```

Gröbner Bases Solver Generator Specification

$$
\begin{aligned}
&{}_p|\mathrm{GBSolver}|_f \quad \models \\
&\quad p = (\mathrm{DB}, \mathrm{BK}, \mathrm{PA}, \mathrm{IP}, \mathrm{PC}, \mathrm{SS}, \mathrm{ES}, \mathrm{SP}, \mathrm{NR}, \mathrm{RS}, \mathrm{CF}, \mathrm{OP}) \ \wedge \\
&\quad \mathrm{DB} \ |\!\models \ \mathrm{Trace} \ \wedge \\
&\quad \mathrm{BK} \in \mathrm{BasisKind} \ \wedge \\
&\quad \mathrm{PA} \ |\!\models \ \mathrm{PolynomialAlgebra} \ \ C, \ \ M, \ \ T, \ \ \mathsf{Poly} \ \wedge \\
&\quad \mathrm{IP} \ |\!\models \ \mathrm{Input} \ \ \mathsf{Poly}, \ \ \mathsf{Inp} \ \wedge \\
&\quad \mathrm{PC} \ |\!\models \ \mathrm{Container} \ \ \mathsf{Poly}, \ \ \mathsf{Idx}, \ \ \mathsf{Cnt} \ \wedge \\
&\quad \mathrm{SS} \ |\!\models \ \mathrm{SelectionStrategy} \ \ \mathsf{Idx}, \ \ \mathsf{Cnt}, \ \ \mathsf{Sel} \ \wedge \\
&\quad \mathrm{ES} \ |\!\models \ \mathrm{ExpansionStrategy} \ \ \mathsf{Idx}, \ \ \mathsf{Cnt}, \ \ \mathsf{Sel}, \ \ \mathsf{Exp} \ \wedge \\
&\quad \mathrm{SP} \ |\!\models \ \mathrm{SPoly} \ \ \mathsf{Poly}, \ \ \mathsf{Idx}, \ \ \mathsf{Cnt}, \ \ \mathsf{Sp} \ \wedge \\
&\quad \mathrm{NR} \ |\!\models \ \mathrm{NormalRemainder} \ \ \mathsf{Poly}, \ \ \mathsf{Cnt}, \ \ \mathsf{Nr} \ \wedge \\
&\quad \mathrm{RS} \ |\!\models \ \mathrm{ReductionStrategy} \ \ \mathsf{Idx}, \ \ \mathsf{Cnt}, \mathsf{Rs} \ \wedge \\
&\quad \mathrm{CF} \ |\!\models \ \mathrm{CanonicalForm} \ \ \mathsf{Poly}, \ \ \mathsf{Cnt}, \ \ \mathsf{Cf} \ \wedge \\
&\quad \mathrm{OP} \ |\!\models \ \mathrm{Output} \ \ \mathsf{Poly}, \ \ \mathsf{Out} \quad \multimap \\
&\quad \bar{f} \in \ulcorner \mathsf{Inp} \ \ \rightarrow \ \ \mathsf{Out} \urcorner \ \wedge \\
&\quad \forall_{\sigma \in \Sigma} \ \ {}_i|[\![f]\!]\sigma|_o \quad \models \\
&\qquad i \subset C[\vec{x}] \quad \multimap \\
&\qquad \bar{o} \subset C[\vec{x}] \ \ \wedge \ \ \langle i \rangle = \langle \bar{o} \rangle \ \ \wedge \ \ \mathrm{Gröbner}(\bar{o}) \ \wedge \\
&\qquad \mathrm{BK} \in \{\mathrm{MinimalBasis}, \mathrm{ReducedBasis}\} \ \ \Rightarrow \ \ \mathrm{Minimal}(\bar{o}) \ \wedge \\
&\qquad \mathrm{BK} = \mathrm{ReducedBasis} \ \ \Rightarrow \ \ \mathrm{Reduced}(\bar{o})
\end{aligned}
$$

Specialized Algorithm

- Generated over 200,000 instances of Buchberger's algorithm for testing
- Produced two more complex detailed specializations:
  (1) Gaussian Elimination generator by specializing the polynomial algebra of linear polynomials
  (2) Euclidean Algorithm generator by specializing the polynomial algebra of univariate polynomials

Conclusion

- The end