Formally Specified Computer Algebra Software DK10

Muhammad Taimoor Khan

Doktoratskolleg Computational Mathematics
Johannes Kepler University
Linz, Austria

March 11, 2010

Outline

Introduction

- Project goals
  - Formal specification of programs written in untyped computer algebra languages
  - Especially to find errors/inconsistencies
    - for example violation of method preconditions
- Computer algebra software at RISC as examples
  - DK11: rational parametric algebraic curves (Maple)
  - DK6: computer algebra tools for special functions in numerical analysis (Mathematica)
  - DK1: automated theorem proving (Mathematica)

Past Activities (October 2009 to February 2010)

- Course work
  - Computer Algebra
  - Automated Theorem Proving
  - Formal Methods in Software Development
  - Thinking, Speaking, Writing
  - Formal Methods Seminar
  - Programming Project
- Literature study
  - Type systems
    - Polymorphism
    - Abstract data types
  - Denotational semantics
  - Functional programming languages
    - Pattern matching
    - Type checking and inference

Software Study - Computer Algebra

- Bivariate difference-differential dimension polynomials and their computation
- Relative Gröbner bases computation (using M. Zhou and F. Winkler's algorithm)
- Maple implementation of the algorithms
- Software
  - Maple package *DifferenceDifferential*
  - Christian Dönch
- Literature reference
  - Christian Dönch. *Bivariate difference-differential dimension polynomials and their computation in Maple*. Technical report no. 09-19 in RISC Report Series, University of Linz, Austria, 2009.

Software Study - Computer Algebra

```
ddsub := proc(c,b)
        local f, g, i, m, n, a1;
        f := c; g:=b;
for i to nops(g) do
      g [i][1] := -g [i][1];
      f := [op(f ),g [i]];
end do;
for m from nops(f) by -1 to 1 do
       for n from m-1 by -1 to 1 do
             if f [m][2]= f [n][2] and f [m][3]=f [n][3] and f [m][4]=f [n][4] then
                   a1 := f [m][1]+f [n][1]; f [n] := subsop(1=a1,f [n]);
                   f := subsop(m=NULL,f ); n := m;
             end if;
       end do;
end do;
....
return f;
end proc;
```

Software Study - Computer Algebra

- Potential considerations
  - Limited types used i.e. integer and list
  - Not much use of Maple libraries - mostly standalone
  - No destructive update of data structures
  - Imperative style of development

Procedural/functional Maple package

Software Study - Algorithmic Combinatorics

- Advanced applications of holonomic systems approach
- Computations in Ore algebras
- Non-commutative Gröbner bases
- Solving linear system of differential equations
- Software
  - Symbolic summation and integration for holonomic functions
  - Mathematica package - *HolonomicFunctions*
  - Christoph Koutschan
- Literature reference
  - Christoph Koutschan. *HolonomicFunctions (User's Guide)*. Technical report no. 10-01 in RISC Report Series, JKU, Austria, January 2010.
  - Christoph Koutschan. *Advanced Applications of the Holonomic Systems Approach*. RISC-Linz, JKU. PhD Thesis, September 2009.

Software Study - Algorithmic Combinatorics

```
OrePlus [p1:OrePolynomial[data1_List, algebra_OreAlgebraObject, order_],
         p2:OrePolynomial[data2_List, algebra_OreAlgebraObject, order_]] :=
Module[{i1, i2, l1, l2, c, c1, c2, m1, m2, sum, coeffPlus},
l1 = Length[data1];
If[l1 === 0, Return[p2]];
l2 = Length[data2];
If[l2 === 0, Return[p1]];
coeffPlus = algebra[[3]];
i1 = 1; i2 = 1;
sum = {};
While[i1 <= l1 && i2 <= l2,
      {c1, m1} = data1[[i1]];
      {c2, m2} = data2[[i2]];
      If [m1 === m2, c = coeffPlus[c1, c2];
      If[Not[MatchQ[c, 0|0.]], AppendTo[sum, {c, m1}]];
      i1++; i2++; ,
      If[OreOrderedQ[m1, m2, order], AppendTo[sum, {c1, m1}]; i1++;
      .... ];];
```

Software Study - Algorithmic Combinatorics

- Potential considerations
  - Based on pattern matching
  - Imperative style of programming
  - Use of abstract data types
  - Use of customized Mathematica functionality
  - Not much use of Mathematica libraries

Procedural/functional Mathematica program with abstract data types

Software Study - Automated Theorem Proving

- Theorema set theory prover (STP)
- Automated prover for theorems
- Works with Prove-Compute-Solve (PCS) strategy
- Integrated with Theorma infrastructure (not standalone)
- Software
  - Mathematica package *SetTheory'Prover'*
  - Wolfgang Windsteiger
- Literature reference
  - W. Windsteiger. *An Automated Prover for Zermelo-Fraenkel Set Theory in Theorema*. JSC 41(3-4), pp. 435-470, 2006, Elsevier, ISSN 0747-7171.
  - W. Windsteiger. *A Set Theory Prover in Theorema: Implementation and Practical Applications*. RISC. PhD Thesis, May 2001.

Software Study - Automated Theorem Proving

```
STP[•If[l_,T_∈ Intersection[A_, B__],i_],a_•asml,af_]:=
    Module[
    {goalList=MapIndexed[•If[NewLabel[ćMembershipAlternatives,l,#2],T∈ #1,i]&,
    {A,B}],proofSits},
    proofSits=Map[Psit[#,a, af]&,goalList];
    ProofStep[Prinfo[ćMembershipFiniteIntersection,l,goalList],
    Sequence @@ proofSits]]
```

Software Study - Automated Theorem Proving

- Potential considerations
  - Pattern matching rules used
    - Are the rules overlaping?
    - Are the rules exhaustive?
  - Implicit type definitions
  - Declarative style of programming

Functional Mathematica program, essentially based on pattern matching

Current Activities

- Definition of simplified/typed versions of Mathematica and Maple (say *MiniMma* and *MiniMaple*)
  - Syntax
  - Type system
  - Semantics and soundness of typing
- Implementation of a type checker prototype
  - Static typing as a prerequisite to logic specification
- Experiments with software fragments available at RISC
- Next - Formal Specification language

Thanks for your attention!