Computer Science Journal of Moldova, vol.**13**, no.2(38), 2005

# Interfaces to symbolic computation systems: reconsidering experience of Bergman

Svetlana Cojocaru Ludmila Malahova
Alexandru Colesnicov

**Abstract**

The article is based on experience of implementation of computer algebra system Bergman and on analysis of other symbolic computation systems. It is noted that many symbolic computation systems meet similar interface problems. Necessity of strict separation of calculation engine and interface shell, functions of these components, and requirements to them are motivated. The described approach will be used at future development of Bergman.

## 1 Introduction

Bergman [8] is a computer algebra system for symbolic calculations in non-commutative and commutative algebra. It calculates Gröbner basis and related information. Bergman is written in Lisp, and the users were to be communicate with the system through underlying Lisp console. This was found unsuitable for most users, and a graphical shell was developed in Java. The system and its interface shell were described elsewhere [1, 4–7].

We present below some inferences from our experience with Bergman and its shell. The article is organized as follows:

- General problems of interfaces to symbolic calculation systems are discussed in Sec. 2. We refer to several existing systems to illustrate our observations.

- Some aspects of Bergman shell implementation and used techniques are discussed in Sec. 3. We permitted here some technical details.
- We try to describe and motivate desired features of interface to a symbolic calculation system in Sec. 4.

## 2 General problem of interfaces to symbolic calculation systems

We suppose the existence of a program executing symbolic computations (a symbolic computation engine, or, simply, an engine). Symbolic computations are widely used in many areas, including pure and applied mathematics, theoretical physics, etc. Multitude of solved problems makes investigators to create specialized engines for symbolic computations in the cases when use of general purpose systems is inefficient, or the necessary functionality is not implemented even in commercial systems. As a rule, the creator of such system has not enough time, resources, and qualification to develop the interface for it. It is not unusual that rich mathematical ideas implemented in an engine are enveloped in poorly designed interface. The absence of the user-friendly standard interface do not permit the extensive usage of such system because of requiring special knowledge and skills, e.g., in programming, to use it. Another problem of symbolic computation engines is multitude of data formats and the implied difficulty in communication between different engines.

We will divide all functionality of a symbolic computation system (SCS) in two parts: the engine features (the computations that it can execute) and the interface features. It is obvious that the latter are external relative to the former and are almost independent of them.

Development of interfaces for SCS was and remains an object of long-time investigations [2, 3].

The problem of SCS interface has the following aspects:

1. Interaction with text editors;
2. Graphics;

3. Interaction with numerical calculation systems;
4. Interaction between different symbolic computation systems (including interaction through computer networks);
5. Testing support;
6. Interaction with end users.

Investigations show that SCS interface development should solve the following problems:

- 2-D presentation of mathematical expressions,
- Editing of mathematical expressions that includes sub-expression manipulation,
- Windows that model sheets of paper and combine texts, formulas, and graphics,
- Processing and presentation of long expressions,
- Simultaneous use of several SCS, which implies the necessity to solve problems of data conversion, configuration management, and communication protocols,
- Satisfaction of special needs for teaching systems, (in particular, the possibility to show intermediate results and explications of processes applied to obtain them; elaboration of electronic manuals, and especially interactive ones),
- Interface extensibility providing additions of new menus, new fragments of on-line documentation, etc.,
- Guiding of the user during the whole period of his/her problem solving,
- The system should be self-explanatory; its operational mode should be understandable directly from the experience of interaction with the system,
- Control over problem formulation correctness and over information necessary to solve it.

The primary scope of an interface is creation of a comfortable environment for a mathematician or another specialist that uses mathematical apparatus. It would be preferable for these users to input data and to obtain mathematical results in their natural 2-dimensional form. The linear form of input can be used also as the input is faster but it imposes additional conventions to enter powers, indices, fractions, etc., or uses additional characters. It is necessary also to provide possibilities to edit expressions, integrate them with a usual text, and obtain results in a form suitable for publication of an article (e.g., LaTeX) or in Internet (e.g., MathML).

The syntactic check of entered mathematical expressions and the spelling check of accompanying text would be also desired features.

The following three categories of SCS can be found analyzing SCS interfaces:

1. Systems or packages that do not have a special interface,
2. Interfaces based on a (specialized) programming language,
3. Graphical interfaces.

This division is not strict: e.g., most systems with graphical interface possess their own programming language also.

There are many systems that have command line interface only, e.g., Singular [22], Bergman, and Yacas [24].

Absence of graphical interfaces is compensated partially by integration in an existing editor like Emacs or its derivatives (e.g., such are Macaulay [13] and Singular), in Scientific Workplace [21], Scientific Word, or Scientific Notebook (e.g., Maple [14] and MuPAD [17]). MuPAD has the interface to Java but their approach is opposite to ours: MuPAD itself is regarding as the shell, and Java programs are treated as applets or plug-ins expanding MuPAD itself. Services provided from these editors may be not too sophisticated but create much more comfortable environment than operating in ASCII from the command line.

We can mention also specialized editors developed to serve as interfaces to SCS. One such editor is TeXmacs by Joris van der Hoeven [23]. It combines elements of TeX and Emacs and was successfully applied for Macaulay 2, Reduce [20], MuPAD, Maxima [16]. Another front-end product with graphical interface is FrontMan [11]. It offers a small but useful set of possibilities (transparent SSH sessions, syntactic coloring of input information and results, export of sessions in HTML format, integrated document visualization through a Web browser, multiple simultaneous sessions). An important fact is that these editors permit creation of users own style. It is a kind of personalization created by a user.

Most of features mentioned above can be found in systems with graphical interfaces. Examples of such systems are Derive [10], Mathematica [15], etc. In general, most of these systems provide:

- Visualization of mathematical formulae in 2-dimensional format,
- Sub-expression manipulation,
- Separate windows for data input and results output,
- Separate windows for graphical operations,
- Export of results in a printable format (RTF, PDF, LaTeX, etc.)
- Comfortable navigation with on-line help,
- Integration with an existing or specially developed editor that facilitates editing of mathematical texts,
- Demonstration of intermediate steps to explain processes of expression transformation.

In addition, so-called "notebooks" support operations over text, mathematical formulae, and graphics. Most of them can be adapted to user's preferences individualizing menus and toolbars, and assigning hot keys to actions.

Communication between different systems is to be supported by use of specially developed unified formats for mathematical formulae, like OpenMath [18] or OpenXM [19].

## 3 Inferences from development of Bergman and its shell

Here we analyze and motivate several solutions taken by us in Bergman implementations. We selected only less trivial aspects; big part of design (e.g., main menu structure, toolbar, sliders, etc.) is usual for graphical shells of any kind.

### 3.1 Implementation language

Bergman was designed under commercial PSL (Portable Standard Lisp). It works also under Reduce [20] that is in its turn based on PSL. For better dissemination of Bergman, we ported it to free CLISP [9] implementation of ANSI Common Lisp. We needed therefore for Bergman shell a programming system that was free and as portable as CLISP.

We selected Java [12] by obvious reasons. We developed the Bergman shell in Intel PC under Linux and Windows, and in Sun Sparc under Solaris, and tested CLISP port of Bergman and the shell in all these machines.

The selection of Java should be classified as very successful. We found only several small problems in porting the shell:

1. Under Windows, the execution thread with Bergman do not terminates simultaneously with calculations as under Unix. It seems to happen due to implementation of program run in terminal window: at first, the terminal itself is started in the thread, and then CLISP Bergman is started under the terminal. The termination of Bergman does not mean termination of terminal. We catch the terminal output and stop the process after the corresponding message. This solution seems to be not very perfect, and we need some additional tuning of running Bergman from the shell under Windows.
2. The screen font sizes and proportions are different in all three systems: Linux, Solaris, and Windows. The design of screen forms is to be made taking this into account. We were to redesign several forms as we tested the shell under Solaris at the first time. The problem is solved by careful form design and obligatory testing under all available systems.

## 3.2 Adaptable user interface: sessions and environments

A session is a set of parameters that fully defines the problem to be solved. Session is implemented as a directory where all data are saved, Bergman input files are generated, and Bergman output files are produced. Sessions serve to return to previous Bergman calculations, modify them, and experiment with them.

Informally, sessions give to a mathematician the possibility to use the previous experience of Bergman's users (own or others) and to save the current setup for the future calculations.

With sessions, it is possible:

- select a session and load its data to panels, i.e., switch to the saved problem;
- create a new session;
- save data in the selected session;
- save data in a different session (save as...);
- delete a session.

One can see also the session directory, comment, and statistics of its usage.

The session mechanism is usual for dialog shells and IDEs (Integrated Desktop Environments). For Bergman, we found it necessary to generalize the notion of session. We called the new feature "environments".

An environment is a partial set of data common for several sessions. It corresponds to the group of mathematical problems the user investigates during different sessions. E.g., after the installation the environments directory contains an environment called "commutative" that fixes a single parameter, the commutativity.

All new sessions are created using the current default environment. Inversely, when a new environment is created, it is based on the parameters of the current session. To save a session as an environment, the user selects parameters that are to be fixed, and drops other parameters.

## 3.3 Dialog data input

Main data for Gröbner basis calculations are list of variables and list of polynomials. They are entered in usual text areas. Other data may be represented as switches (flags) and selections. There are approx. 30 input fields the user should set.

It was a serious problem to design these fields in comfortable and reviewable manner. The first idea was to enter the session parameters step by step (in wizard mode) but it was rejected. One of reasons for it was the absence of suitable wizard implementation in Java, but the main reason was that such mode is quite tedious. Next, we tried to position input fields in five tabs but this was badly reviewable.

The current solution permitted us to concentrate all information in a relatively small panel. We use drop-down menus and labels that show the current selection. The drop-down menu is activated when the corresponding label is clicked. Let us revise an example to make the situation more clear.

There is a possibility to select the field for polynomial coefficients. The field characteristic can be 0, 2, or any odd prime number. In characteristic 0, we use integers instead of rational numbers by reducing all coefficients to a common denominator. The drop-down menu for coefficient field contains the following items:

- Machine integers (16-bit)
- Machine integers (32-bit)
- Machine integers (64-bit)
- Lisp integers (arbitrary precision)
- Characteristic 2
- Odd prime characteristic

Suppose the user selects odd prime characteristic. Then an additional small dialog window appears and the user is asked to enter an odd prime. The entered number is checked. Then the label shows: "Odd prime characteristic = $p$", where $p$ is the entered odd prime. Here we do not need the additional input field on the form for the odd prime number, and all information is always visible.

We use menus and not combo boxes because of pure technical reason. We can assign a program action to each menu item that is not the case with combo boxes.Using actions, we can treat mutually dependent parameters in a natural manner.

The equivalent set of radio buttons plus the input field (as in our previous implementation) occupies, obviously, the bigger form surface.

## 3.4 Expanded consoles

When we start a calculation under the shell, Bergman run in a console window we programmed for the case. Except of this, we have a possibility to start a console Bergman session. We found that the console features should be expanded. E.g., the calculation console has additional buttons “Stop” and “Continue” and its input and output are caught and come from/to files, but the session console has the feature of keyboard input imitation from an arbitrary file selected by the user.

### 3.5 Internal Bergman hooks

We inserted into Lisp coded Bergman modules so-called “hooks” where we call external programs supplied by the shell. “Stop” button can not stop calculations in an arbitrary place; the process continues up to some suitable point where the hook is inserted. The hook checks if the stop button was pressed, and stops calculation, with the possibility to continue it from the same point. There are many possible uses of such hooks, e.g., progress indicator that moves forward in some point of calculation.

## 4 Conclusion: desired features of interface to symbolic calculation system

We can conclude from all aforementioned that SCS interfaces have a lot of features and functions in common.

Discussion in Sec. 2 and 3 shows that all discussed problems can be reviewed as general problems of graphical shells to SCSs. We can look at Bergman as at the **calculation engine** which is called from the dialog shell. Inversely, we can look from the shell side and treat Bergman as the **calculation plug-in** for the shell.

The separation of calculation engine and interface shell means that a system can be developed for semi-automatic generation of interface shell for a SCS. Such system should contain ready made adaptable interface modules in Java and an interface generator.

In implementation of such system, the following objectives must be reached:

- To implement selected features of SCS interface at the generalized level permitting automated production of interface, taking into account specifics of this area. To implement cross-process interaction with a symbolic computation engine.

- To elaborate, inside the interface, tools for smart user personalization and intellectual adaptation to his/her preferences.
- To elaborate an interface generator that will produce interface for a target symbolic computation engine.
- To elaborate programs and filters for interaction with other existing SCS.
- To elaborate and/or adapt and integrate tools for auxiliary development tasks, e.g., help and documentation tools, extensibility support, testing support, etc.
- To elaborate and/or adapt and integrate tools for visual presentation of data and results.
- To produce, as the result, Java packages to be used in interface generation, and Java application(s) for interface development.

Some preliminary research is necessary:

- Classification and technological description of features and functions specific to interfaces for SCS;
- Clarification and classification of cross-process interactions between interface modules and symbolic computation engines;
- Generalization of selected features to the level permitting automated interface generation;
- Description of adaptive behavior of interface elements, and techniques to implement it;
- As the result, the necessary scientific basis will be created to automated development of interfaces for SCS.

Interaction between different systems can be provided also.

SCS is a very useful tool that simplifies formula manipulation and handling of mathematical models for engineering applications, for mathematical research, for education and for many other areas. Our approach can be applied in all these cases and these areas will gain time and efforts for interface development. Universality of the proposed solution will be guaranteed if we will use Java as the technology for its implementation. The developed packages and applications will permit investigators in different areas to concentrate efforts on symbolic computation engines and to use ready-made interface solutions.

## References

[1] A. Colesnicov. *Implementation and usage of the Bergman package shell.* / Computer Science Journal of Moldova, **vol. 4,** nr. 2 (11), 1996, pp. 260–276.

[2] N. Kajler, N. Soiffer. *A Survey of User Interfaces for Computer Algebra Systems.* / Journal of Symbolic Computation, **vol. 25,** issue 2, February 1998, pp. 127-159.

[3] N. Kajler (ed.). *Computer-Human Interaction in Symbolic Computation.* - Springer-Verlag: Wien, 1998. - ISBN 3–211–82843–5.

[4] J. Backelin, S. Cojocaru, V. Ufnarovski. *BERGMAN.* In: Computer Algebra Handbook. J. Grabmeier, E. Kaltofen, V. Weispfenning (eds.). – Springer-Verlag: 2003, pp. 349–352.

[5] J. Backelin, S. Cojocaru, V. Ufnarovski. *The Computer Algebra Project Bergman: Current State.* In: "Commutative algebra, Singularities and Computer Algebra", eds. J. Herzog and V. Vuletescu, 2003, pp. 75–101. – Series II. Mathematics, Physics and Chemistry. **Vol. 115,** Kluwer Academic Publishers.

[6] J. Backelin, S. Cojocaru, A. Colesnicov, L. Malahova, V. Ufnarovski. *Problems in interaction with the Computer Algebra*

*System Bergman.* In: “Computational Commutative and Non-Commutative Algebraic Geometry”, **vol. 196,** NATO Science Series: Computer & Systems Sciences. S. Cojocaru et al. (eds.). – IOS Press, 2005, pp. 185–198.

[7] J. Backelin, S. Cojocaru, V. Ufnarovski. *Mathematical Computations Using Bergman.* - Lund University, Centre for Mathematical Science, ISBN 91–631–7203–8, 2005, 206 p.

## Web references

[8] Bergman: *http://www.math.su.se/bergman/*

[9] CLISP: *http://clisp.sourceforge.net/*

[10] Derive: *http://www.chartwellyorke.com/derive.html*

[11] FrontMan:
*http://rpmfind.net/linux/RPM/sourceforge/r/rp/rpmsforsuse/frontman-0.3.4-1.i386.html*
*http://www.eleceng.ohio-state.edu/ ravi/kde/frontman.html*

[12] Java: *http://java.sun.com/*

[13] Macaulay: *http://www.math.uiuc.edu/Macaulay2/*

[14] Maple: *http://www.maplesoft.com/*

[15] Mathematica:
*http://www.wolfram.com/products/mathematica/index.html*

[16] Maxima: *http://maxima.sourceforge.net/*

[17] MuPAD: *http://www.mupad.de/*

[18] OpenMath: *http://www.openmath.org/*

[19] OpenXM: *http://www.math.kobe-u.ac.jp/OpenXM/*

[20] Reduce: *http://www.reduce-algebra.com/*

[21] Scientific Workplace, Scientific Word, Scientific Notebook: *http://www.mackichan.com/*

[22] Singular: *http://www.singular.uni-kl.de/*

[23] TEXmacs: *http://www.math.upsud.fr/ anh/TeXmacs/TeXmacs.html*

[24] Yacas: *http://yacas.sourceforge.net/*

Received October 3, 2005

S. Cojocaru, L. Malahova, A. Colesnicov,

Institute of Mathematics and Computer Science,
5 Academiei str.
Chişinău, MD−2028, Moldova.
E–mail: *sveta@math.md*, *mal@math.md*, *kae@math.md*