SymbolicData:SDEval - Benchmarking for Everyone
– Preprint –
(Submitted for Computer Algebra in Scientific Computing, CASC 2013)

Albert Heinle[1], Viktor Levandovskyy[2], Andreas Nareike[3]

[1] Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
[2] Lehrstuhl D für Mathematik, RWTH Aachen University, Aachen, Germany
[3] HTWK Leipzig, Leipzig, Germany

Abstract. In this paper we will present SDEval, a software project that contains tools for creating and running benchmarks with a focus on problems in computer algebra. It is built on top of the Symbolic Data project, able to translate problems in the database into executable code for various computer algebra systems. The included tools are designed to be very flexible to use and to extend, such that they can be utilized even in contexts of other communities. With the presentation of SDEval, we will also address particularities of benchmarking in the field of computer algebra.
Furthermore, with SDEval, we provide a feasible and automatizable way of reproducing benchmarks published in current research works, which appears to be a difficult task in general due to the customizability of the available programs.
We will simultaneously present the current developments in the Symbolic Data project.

1 Introduction

Benchmarking of software – i.e. measuring the quality of results and the time resp. memory consumption for a given, standardized set of examples as input – is a common way of evaluating implementations of algorithms in many areas of industry and academia. For example, common benchmarks for satisfiability modulo theorems (SMT) solvers are collected in the standard library SMT-LIB ([BST10]), and the advantages of various solvers like Z3 ([DMB08]) or CVC4 ([BCD$^+$11]) are revealed with the help of those benchmarks.

Considering the field of computer algebra, there could be various benchmarks for the different computation problems. Sometimes, one can find common problem instances throughout papers dealing with the same topics, but often there is no standard collection and authors use examples best to their knowledge. For the calculation of Gröbner bases for example, there is a collection of ideals that often

appear when a new or modified approach accompanied by an implementation is presented (e.g. in [Neu12], the author used the KATSURA-n, $n \in \{11, 12\}$ from [KFI+87] and CYCLIC-m, $m \in \{8, 9\}$ from [BH08]). Regarding the computation on that set, the new implementation is then compared to existing and available ones.

An outstanding systematic practice is realized by the computer algebra lab, lead by V. P. Gerdt, of the"Joint Institute For Nuclear Research", on their website about the progress in research of computing Janet and Gröbner bases of complicated polynomial systems (`http://invo.jinr.ru/`).

Nevertheless, in many areas there is rarely a standard test set, and often the calculated timings are hard to reconstruct due to different parameters in algorithms that can be set.

Another difficulty is the fair evaluation on how much time is consumed; we will discuss this topic detailed in section 3.

The SYMBOLIC DATA project ([Grä09]) started more than 10 years ago to collect various instances of problems, especially in computer algebra.

In this paper, we discuss particularities concerning code quality evaluation especially for the computational algebra community and present SDEVAL, a benchmarking toolbox written in PYTHON covering the following two main tasks:

(i) Creating benchmark sets with the help of the problem instances provided by the SYMBOLIC DATA database entries.
(ii) Running benchmarks, with a flexible (i.e. cross-community adaptable) interface that makes reproduction as simple as possible.

For item (i), we implemented for certain computational problems (e.g. calculation of a Gröbner basis) translators of respective problem instances in SYMBOLIC DATA into executable code for a set of computer algebra systems. It can be done using a graphical user interface, or alternatively via a program run in the terminal. This task addresses e.g. developers, who want to compare the running time of their implementations with those of available software without the necessity of becoming familiar with all of the available systems. Additionally, it addresses mathematicians who discovered a certain instance for a computational problem and want to examine what computer algebra systems are able to solve it and what solutions are provided, as they might differ – dependent on the uniqueness of the result – for the different systems.

Item (ii) has a broader range of possible uses. First of all, it provides a way to run arbitrary programs on different inputs. Optionally, it monitors the computations and terminates programs automatically if they exceed a user-given time or memory limit. Moreover, it provides the user with intermediate information on which tasks are currently run and which have already finished, and one can stop certain calculations manually without having to start the whole process all over again. The information is provided through a generated HTML document and – for potential automation purposes – through an XML file. We chose for this part a structure that is strongly connected to a folder, which can be shared with others and after an negligible amount of adjustment to another machine, the results can be reproduced. We envision for the future that `tar`-balls of those

folders would be published with computation-focused papers, so that it becomes easier to verify results of the authors.

This paper is structured as follows. Section 2 deals with current developments regarding Symbolic Data and presents a general overview of this project. The subject of Section 3 will be SDEval. We will show how one can use it and extend/adjust it to individual computation problems if needed. We will finish by addressing related work in Section 4 and future tasks to come in Section 5.

The current version of the presented toolkit SDEval can be found at `github.com/ioah86/symbolicdata`. The latest informations on Symbolic Data are available at `http://symbolicdata.org`.

2 Current Developments in the Symbolic Data Project

2.1 The General Idea

When Symbolic Data started, the main incentive was to revise a largely unstructured collection of polynomial systems from different areas and thus making them more accessible and in consequence easier to maintain. Additionally, the gained knowledge and experience should allow Symbolic Data to be extended to other computer algebra related data. We will begin by sketching the main problems very briefly and will then focus on the newer development which draws heavily from the rapid development of the Semantic Web technologies in the last years.

Right from the beginning of Symbolic Data, it was clear that the data produced by the computer algebra community is highly inhomogeneous. But rather than ignoring this and trying to fit all available data into a given arbitrary schema, we sought ways to incorporate this inhomogeneity.

There are two main questions which arise in the given scenario. The first one concerns the resources themselves: how is the data stored? Since there is a wide range of computer algebra systems, the file formats vary as well. The problem was here, to find a suitable file format that is versatile enough to serve as a lingua franca between different computer algebra systems and as well widespread enough to be readily processed by most programming languages. This question concerns the resources themselves and will be discussed in subsection 2.2.

The second question is where, how and which information *about* the resources is stored. This kind of information is often called **metadata**. Symbolic Data stores data and metadata separately. This has two advantages: Firstly, data and metadata do not have to be stored on the same server. Secondly, for resources with large file sizes it is often tedious to move around and edit those resource files.

The question of *how* to store metadata will be treated in subsections 2.3 and 2.4. The hardest part remains *which* metadata should be stored. For polynomial systems there are concepts like dimension, highest degree, number of variables, homogeneity properties etc. For other fields it will be other concepts and this can only be answered by the computer algebra community as a whole.

2.2 XML as Resource Format

Until around the year 2000 the XML file format was very popular to store any kind of data. It still is but there are other alternatives, most prominent JSON and YAML. One key property of all three is that the files are human-readable. From a programmer's point of view, an important feature is the availability of XML parsers and serializers for every major programming language.

It should be noted that XML is not really a language on its own but rather a set of rules that can be extended by an XML Schema (which usually consists of one or more XSD files) to define a custom language or file format. One of the various applications of this technique is for instance XHTML ([P+00]).

Another advantage is that an XML file can be easily checked against an XML Schema to determine whether it is well-formed or not. Symbolic Data uses XML based file formats to store resource data.[4] A recent discussion has been if one should adhere to this or if this question is not even relevant. With the separation of resource data and metadata this is however not a question that has to be answered right away.

2.3 Different ways to collect metadata

There are mainly three ways to create and store metadata, each of which with its own advantages and disadvantages.

A common way to store information about given resources is to set up a 'classical' relational database, e.g. MySQL or PostgreSQL. However, before setting up a database the structure of the data should be clear to the point that an Entity-relationship model could be drawn. Altering the abstract model after a large amount of data has already been stored in the database is often a tedious task, to say the least.

Another, somewhat different, method to create metadata is to add tags or keywords to each record. This has been made popular by the Web 2.0. The main advantage is, that additional tags can be introduced at any given time. One disadvantage concerning consistency is that an author of an entry is not forced to include these tags.

While XML was not originally aimed at being a format for databases, it can be used to do precisely this. And just as SQL is a languange to query SQL databases, there is also a query language for XML databases, namely XQuery.

The third way to store metadata is to use the RDF/OWL standards. While SQL databases can be visualized as collection of tables with cross references, RDF metadata can be seen as a directed graph with labelled edges. Since this is the way we adopted for Symbolic Data we will discuss this in greater detail in the next subsection.

[4] We note that MathML is already an XML based file format for mathematical data. We decided not to use MathML for our resources. MathML describes formulas in a very detailed and almost scrupulous way that creates a big overload in terms of file size and also greatly decreases the human readability.

2.4 Collecting meta data with RDF

Even with the extended notion of tags that XML can establish, there is still something missing. Tags are often not independent but bear a structure themselves. For instance, a given tag can be a more special version of another tag. These relations between tags are often called 'semantics'.

RDF is an acronym of "Resource Description Framework" and it is above all a data model. There are different ways to serialize RDF, so RDF data can be written as XML or JSON, as well as Turtle ("Terse RDF Triple Language") which is a format specifically designed for RDF.

The basic idea is that each piece of information is stored in form of so-called **triples**. Each triple consists of a subject s, a predicate p and an object o. Those terms are used rather loosely instead of adhering to strict linguistic rules. Written down in Turtle, a triple can be expressed by simple juxtaposition and a final period:

$$s\ p\ o\ . \tag{1}$$

Subjects and predicates have to be a URI (Uniform Resource Identifier ([BL94])) while objects (or 'values') can be either be a URI or a literal in lexical form (a text string) which can either be plain or typed. There are some common types (e.g. 'integer') but custom types can be defined as well. The sets of subjects, objects and predicates are not necessarily (mutually) disjoint. Most notable, a predicate of one triple can also become the subject or object of another triple. It will shortly be clear how this makes sense.

A valid question is of course how one would start working with RDF. Suppose one has no database at all and just a resource at
`http://symbolicdata.org/XMLResources/IntPS/Caprasse.xml`.
One could jot down some properties that help to find this resource. Some of these properties could be extracted directly from the resource, others would have to be calculated. It could look like this:

```
<http://symbolicdata.org/Data/PolynomialSystems/Caprasse>
    a sd:IntPS ;
    sd:hasDegree "56" ;
    sd:hasDegreeList "3,3,4,4" ;
    sd:hasLengthsList "4,4,9,9" ;
    sd:hasVariables "x,y,z,t" ;
    sd:relatedXMLResource
      <http://symbolicdata.org/XMLResources/IntPS/Caprasse.xml> .
```

Some explanations: These are six triples which could also be written in the form (1). Since all triples share the same subject (the first line), the code can be compacted by ending lines with a semicolon instead of a period. This signifies that the subject in the next triple is the same and thus can be omitted. The `sd:` is just an abbreviation for `http://symbolicdata.org/Data/Model/`.

One great thing about RDF is that one does not have to worry too much about the used predicates. Refactoring data can easily be done by 'search and

replace'. In many cases it is not entirely clear from the beginning what the 'right' predicates are (i.e. the predicates that are best suited). Contrary to an SQL database, RDF supports and even encourages a bottom-up approach to building a database.

RDFS and OWL. While RDF mostly defines how the data is structured on the lowest level, there is also RDF Schema, in short RDFS, which extends RDF by defining a number of basic classes and properties. The most basic extension is probably `rdfs:label`, which can be used to add a 'display name' to the URIs.

RDFS also provides the formal notion of an RDF class `rdfs:Class`, so it is possible to talk about the RDF classes in RDF itself. Building on this, RDFS defines `rdfs:subClassOf` as well as `rdfs:domain` and `rdfs:range`. The latter two are used to define the domain and range of a predicate p, in Turtle it looks like this:

$$p \ \ \texttt{rdfs:domain} \ \ c \ .$$

where c is an `rdfs:class`.

The next step is to use OWL (Web Ontology Language) and OWL2 to define a highly elaborated structure of the classes and predicates used. Again, like with RDFS, OWL is not really a new language but it defines new keywords that extend RDF.[5]

The collection of classes, predicates, their relation to each other, and the rules of their application is called **Ontology**. There are already various ontologies available[6], most notably the Dublic Core ontology, which defines how to talk about different kinds of publications, and the FOAF ontology, which defines how to talk about persons, their contact information and their connections to other people. We will not go into further detail about the vast range of possibilities that OWL offers.

One key idea of the Semantic Web is, that one does not necessarily invent new ontologies but use and possibly extend existing ones. But even if one has two similar databases that each use their own ontology, OWL offers predicates to map the vocabulary of one ontology to that of another.

Putting all into practice with SPARQL. When working with RDF, the equivalent to a database is a collection of triples that are loaded into a so-called **triplestore**. When talking about SQL and XML databases we also mentioned their respective query languages. With SPARQL there is also a query language for RDF[7]. We will not explain SPARQL in greater detail here, but many examples can be found and tested live at `http://symbolicdata.org/wiki/QuickStart`.

[5] The reason for the new names is partly a historical one. The names mark different stages of the development of the Semantic Web and refer to different specifications.

[6] see also `http://semanticweb.org/wiki/Ontology`

[7] Actually there is more than one query language for RDF, but we focus on SPARQL here. For details see `http://en.wikipedia.org/wiki/RDF_query_language`

For the moment, let's just note that with SPARQL one can formulate queries like

- find all resources have a certain property (e.g. a degree of at most 36)
- find all resources that are missing a certain predicate p
- find all resources that are in class c_1, but not in class c_2
- find all resources of class c_1 together with their properties defined by predicates $p_1, \ldots, p_n$

SPARQL returns a subset of the triples from the triplestore. The format can be JSON, XML, Turtle and some others. What format is suited best depends on how the results will be further processed.

Linked Data. With RDF, OWL and SPARQL it is possible to advance beyond single databases towards a linked data network of computer algebra resources. This is of course a work in progress which can neither be done overnight nor by a single person. RDF is no magic tool which does all this automatically, but what it does is to provide the means to seriously tackle this project.

2.5 Ways of Using and Contributing to Symbolic Data

After having outlined the principles and benefits, we will now present the possibilities of using Symbolic Data and contributing to the project. Depending on the background and the time one is able to invest, there are different methods.

One of our goals is to incorporate new metadata into Symbolic Data, but even more we are interested in getting more people involved with it. The first place to start to get more information is our Wiki at `www.symbolicdata.org`.

For people who are already familiar with RDF or want to become acquainted with it, there is our official repository at GitHub: `github.com/symbolicdata/symbolicdata`. It contains some resource data and of course metadata. One can fork this repository, include own metadata and send a pull request.

Even without generating RDF metadata, it is possible to use the existing metadata and set up a local triplestore with (a subset of) Symbolic Data. Custom SPARQL queries can provide exactly the presentation of the data a user requests.

One might also already have a relational database. In this case, there are tools that can easily convert an SQL database into RDF and merge it with the collection of Symbolic Data. Sparqlify[8] even does this 'live' by translating SPARQL queries to SQL queries.

On the other hand, one might also have a large set of resources in a well-defined format for which it is (at least implicitly) clear how to calculate or extract metadata. We can help to do this (semi)automatically and store the result in RDF which can then be converted into an HTML presentation while additionally providing a SPARQL interface to others.

[8] `http://aksw.org/Projects/Sparqlify.html`

3 SDEval

3.1 Particularities about Benchmarking in Computer Algebra

Challenges. Writing benchmarks in the field of computer algebra differs from other benchmarking tasks. A collection of challenges that appear is the following.

- Sometimes, the results of computations are not unique; that is, several nonequal outputs can be equivalently correct. It is not always possible to find a canonical form for an output. Even if this is the case, the transformation of output into the canonical form can be quite costly. Moreover, the latter transformation is not necessarily provided by every single computer algebra system.
- Related to the previous item: If an answer is not unique, then the evaluation of the correctness of the output is often far from trivial.
- The field of computer algebra deals with a large variety of topics, even though it can be divided into classes of areas where certain common computational problems do appear. Thus, there need to be collections of benchmarks, optimally one as a standard for each class. The benchmark creation process should be flexible to be applicable in a wide range of areas.

We tried to address those challenges as much as possible when designing our toolkit.

In particular, the first item is something that differs the creation of benchmarks for computer algebra problems from most other fields of studies.

The second item leads to one of the design decisions we made for SDEval, namely that we provide an interface for decision routines, and partially include some serving as an example how they could be added. Then, a particular community can deal with this question based on their problems, and provide SDEval with the information on what routine to call to obtain an answer.

Correct and Feasible Time Measurement. Another seemingly trivial, yet controversial question is the correct time measure of computations, as mentioned in the introduction. It is very common in computer algebra systems to provide a time measuring functionality, and many of the timings provided in papers were calculated using those commands, since it is easily available.

Nevertheless, this methodology is questionable. Often one cannot verify their validity due to e.g. their source not being open. Furthermore, sometimes some run-time-benefiting calculations are already done during the initialization phase; therefore one has to specify clearly where to start the provided time measurement. If one makes use of the implemented techniques, every program has to be analyzed in detail to find the correct spot to start the time counting in order to make the comparison fair. Hence, the use of system-provided time measuring is not practical for fair comparisons.

A widely-spread method in software development is to run programs with the `time` command provided with Unix based operating systems. Even though

the time for parsing input – which is in general not the complex part about the computations done in computer algebra – would then also being taken into account, we decided that this method is the best choice for SDEval.

It has also another benefit: We are interested in extracting the timing results from the output files in an automated way, and there is a standard for providing timings given by the IEEE standard `IEEE Std 1003.2-1992 (‘‘POSIX.2’’)`; the `time` command can be instrumented using a parameter to provide its output according to this standard. Arranging this format for the output with the help of the included time measurement mechanisms in computer algebra systems can be regarded as an infeasible requirement for a user.

3.2 The Creation of a Benchmark Suite

Basic Terminology. Let us start with defining some terminology we want to use throughout this section. This will serve the purpose of a better understanding of the design principles of SDEval.

Definition 1 (SD-Table). *An SD-Table denotes the folder structure of a chosen sub-folder in the `XMLRessources` folder in the* Symbolic Data *project. Those sub-folders represent the tables with computation problems in the* Symbolic Data *project.*

Example 1 (SD-Table). An example for an SD-Table is `IntPS`. It contains instances of ideals in a polynomial ring over $\mathbb{Q}$ using integer coefficients.

Definition 2 (Problem Instance). *A problem instance is in our context a representation of a concrete input – aligned to the* Symbolic Data *format – that can be used for one or more algorithms. The input values for the chosen algorithm are contained in this problem instance. A problem instance is always contained in an SD-Table.*

Example 2 (Problem Instance). A problem instance is for example the entry `Amrhein` (an integer polynomial system taken from [AGK96]) in the SD-Table `IntPS`. It contains variables and a basis of polynomials, and those can be used for Gröbner basis computations, for example.

Definition 3 (Computation Problem). *A computation problem is a concrete and completely specified member of a family of algorithms. In the context of* SDEval*, it specifies which computations we want to perform on certain problem instances.*

A selection of computation problems is already provided in the SD-Table `COMP`. The selection can be extended by the user.

Example 3 (Computation Problem). A computation problem is for example the computation of a Gröbner basis given an ideal over a polynomial ring over $\mathbb{Q}$ using the lexicographic ordering (abbr. `GB_Z_lp`, can be found in the SD-Table `COMP`).

Definition 4 (Task). *A task consists of a computation problem and a selection of problem instances that are suitable as inputs for it.*

Automated Creation of Benchmarks Now that we have defined some basic terminology, we will address how a benchmark suite can be generated using the problem instances given in the SD-Tables.

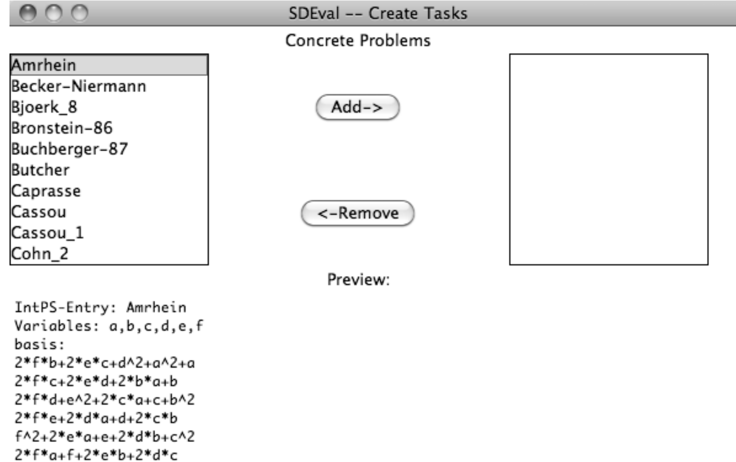
In the toolbox, one can find two Python programs that can do this job: `ctc.py` and `create_tasks_gui.py`.

The first one is a command-line program, the second one provides a graphical user interface.

Those scripts perform the following three steps

1. The user chooses from a set of computation problems.
2. After that, the script collects possible problem instances across the SD-Tables and presents them to the user. One can pick the desired problem instances that should be included in the benchmark. An illustration of this step is given in Figure 1.
3. In the last step, besides setting configuration parameters, the user selects from a set of computer algebra systems for which it is known that they contain implementations of the algorithms that solve the selected computation problem.

Fig. 1. The selection of the problem instance from integer polynomial systems



After these three steps, the user confirms his choices and a folder is generated containing executable codes for the selected computer algebra systems, a Python script to run all the calculations and some adjustable configuration files (e.g. if the user wants to change call parameters for a computer algebra system). The concrete structure is given as in Figure 2.

Fig. 2. Folder structure of a taskfolder

```
+ TaskFolder
| - runTasks.py         //For Running the task
| - taskInfo.xml        //Saving the Task in XML-Structure
| - machinesettings.xml //The Machine Settings in XML form
| + classes             //All classes of the SDEval project
| + casSources          //Folder containing all executable files
| | + SomeProblemInstance1
| | | + ComputerAlgebraSystem1
| | | | - executablefile.sdc
| | | + ComputerAlgebraSystem2
| | | | - executablefile.sdc
| | | + ...
| | + SomeProblemInstance2
| | | + ...
| | + ...
```

We will refer to this folder as taskfolder from now on. This folder can then be sent to the machine where the computations are intended to be run.

As outlined before, the creation tool is very flexible and easily extensible. This is due to the object oriented nature of the code written in Python. One can specify new computation problems, and declare which problem instances can be chosen as inputs. The respective code for the computer algebra systems can be added in a template-fashion and does not require familiarity with the particular concepts of Python.

3.3 Running a Benchmark Suite

General Assumption 1: Whereas the creation of the benchmark suite is possible on any machine where Python is installed, the running routine requires a machine running with a UNIX-like operating system (e.g. Linux or Mac OS X). We require the `time` command or some equivalent to be supported, which is in general always the case on UNIX systems.
General Assumption 2: Calculations are run within a terminal. This decision was made due to the fact that calculations are often sent to a compute server. The connection to that server is in general provided through a terminal interface.

The running of a benchmark is closely connected to the taskfolder as presented in the previous section. As one can see in Figure 2, it contains a Python script called `runTasks.py`. One can either generate an individual taskfolder using the design principles given in the documentation, or one can use a taskfolder generated by the task creation scripts.

If one executes `runTasks.py`, all the stored scripts for all the contained computer algebra systems will be run consequently. Using execution parameters, one can instruct the script to kill a process once a given time or memory limit is reached.

The script will create – if not yet existent – a sub-folder within the taskfolder named `results`. Within `results`, there will be a folder named by the time

stamp when `runTasks.py` was executed, where it will store the results of the computations and some monitoring information about the executed scripts in form of HTML and XML files.

During the execution process, the user can feel free to terminate manually a running process without having to restart `runTasks.py`. It will simply continue with the next waiting program on the next script in the queue.

This design of the benchmark execution part has the following benefit. Future authors that execute their scripts on certain files could provide their taskfolder with the paper they submitted. Then everyone can see the results (i.e. the outputs of the programs), and verify the timings using the calculated table. Furthermore, they can run the calculation using `runTasks.py` after adjusting the configuration to their machine (i.e. replacing the call commands for the computer algebra systems to those used on one's machine).

There are further uses of the running routines. As one can see, the execution of the benchmarks can be seen completely detached from the creation part. This means, that one can create one's own taskfolder, defining programs one wants to run and provide the inputs inside the `casSources` folder.

Even though the routines were designed to fit especially the needs of the computer algebra community, the principles can be used for almost any kind of program.

Another use would be to keep track of the development process of a software project over time. Executing the `runTasks.py` script after every version change would reveal profiling information on the different examples. The profiling can be automatized since the timing-data after every run is stored in an XML file.

3.4 Ways of Customizing and Contributing to SDEval

We have seen in the last section that the part of the execution of the respective programs on the problem instances is highly customizable. There are also ways for customization of the part where one creates benchmarks.

For the case that we have not considered a certain computer algebra system that is capable of solving a particular computation problem, a user familiar with this computer algebra system is able to provide a template without the need of a deep knowledge of Python.

For the case that there exists a not yet considered computation problem for which inputs can be derived from existing SD-tables, one can provide a representative Python class, a template for a computer algebra system code and link the respective SD-Tables to it.

More possibilities and details can be found in the documentation of SDEval.

4 Related Work

4.1 Related Work to SDEval

StarExec ([SST12]): This is an infrastructure especially for the logic solver communities. Its main focus is to provide a platform for them to manage their

benchmark libraries and run solver competitions. It is widely used in conferences based on logic solving to evaluate the benefits of new approaches. Moreover, it includes translators of problems between the different communities dealing with logic solving.

HOMALG ([BR08]): Focusing on constructive homological algebra, the HOMALG project provides an abstract structure for abelian categories and is distributed as a package of the computer algebra system GAP ([GAP13]). For time critical computations, it allows the usage of other computer algebra systems, i.e. the task is translated to the respective system and then executed.

SAGE ([S+08]): The popular computer algebra system SAGE provides as an optional package an interface to the database of integer polynomial systems (`IntPS`) of the SYMBOLIC DATA project. One can directly load those problem instances as objects in SAGE for further calculations.

4.2 Related Work to Symbolic Data

There are various sets of problem instances for different computation problems collected by different communities during the last couple of decades. Here is a selection of some interesting projects.

POCAB ([SEW12]): POCAB is a collection of models coming from the field of biology and chemistry. They concentrate on examining the different algebraic entities given in those models in order to apply algebraic methods in the process of their analysis. Their data of interest is coming from two renowned and publicly available databases, namely KEGG ([KG00]) and the BIOMODELS DATABASE ([LNBB+06]).

POLYTOPE DATABASE (`www.mathematik.tu-darmstadt.de/~paffenholz/data.html`): We established an interesting contact, namely to Dr. Andreas Paffenholz. He has a large amount of resource data as Polymake files for different kinds of polytopes. We will extract and possibly generate metadata and include this into SYMBOLIC DATA.

SWMATH (`http://swmath.org`): The focus of this project is to serve as an information service for mathematical software. It consists of an extensive database of software projects from the field of mathematics, and contributes a systematic linking of program packages with relevant publications.

QAOS (`http://qaos.math.tu-berlin.de`): This project at the TU Berlin is the successor of the "Kant Database of Number Fields". It provides an interface to various categories of algebraic objects.

5 Conclusion and Future Work

We have presented the latest developments of the SYMBOLIC DATA project, and a benchmarking tool named SDEVAL that is built on top of it. In this paper, we addressed the particularities of benchmarking in the field of computer algebra, and with SDEVAL, we have presented a flexible, extensible and easy-to-use tool that is designed to accept the challenge.

Moreover, we introduced a practice how the reproduction and the analysis of computations with their timings would become more feasible in the future. Our approach for that is the taskfolder containing the benchmark program and the respective input files.

A future task will be to extend the benchmark creation tool to contain both more computer algebra systems and computation problems. The program that runs the benchmarks will include output interpretation routines to determine reliably the correctness of results. For that, one has to consider every computation problem in a detailed way and we hope for support from the communities in the future to accomplish that.

The future work on the Symbolic Data project will consist of expanding the database with new entries provided by the community. We are constantly establishing contacts to interested researchers. Prof. Dr. Jürgen Klüners for example, who already has a huge Postgres database of number fields[9], pointed us to other resources for local fields and number fields, which are not yet conveniently accessible.

Prof. Dr. Max Horn described briefly to us how a Symbolic Data library could be included into Gap. Since there are also resources available through Gap databases, it would be also interesting also make these searchable with Symbolic Data.

In personal talks with Pavel Metelitsyn it became clearer that a translation of resource data is an important task. Also it is interesting to not only store metadata but to generate it when needed. Sage ([S+08]) is probably best suited as an additional software service that can be combined with Symbolic Data. We will investigate this idea further.

As benchmarking is a very wide-ranged topic, we will figure out in the future if there are more challenges – maybe caused by computation problems we have not considered yet – that we are not aware of in the present state. It remains a practically relevant and interesting problem.

Acknowledgements

We thank the "Deutsche Forschungsgesellschaft" (DFG) for partial financial support for the development of SDEval ("DFG Priority Project SPP 1489").

We thank the "Europäische Sozialfonds für Deutschland" (ESF) for funding the work on Symbolic Data in the context of the project "eScience Forschungsnetzwerk Sachsen". Moreover we personally thank Dr. Toni Tontchev for his enthusiastic support.

We are grateful to Hans-Gert Gräbe for his encouragement and the fruitful discussions we had with him.

The authors thank Johannes Waldmann for his presentation on the benchmarking practice of the logic solver communities during the Symbolic Data workshop.

Special thanks to Mark Giesbrecht for his helpful suggestions and comments.

[9] http://galoisdb.math.uni-paderborn.de/home

References

[AGK96] Beatrice Amrhein, Oliver Gloor, and Wolfgang Küchlin. Walking faster. In *Design and Implementation of Symbolic Computation Systems*, pages 150–161. Springer, 1996.

[BCD+11] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *Computer Aided Verification*, pages 171–177. Springer, 2011.

[BH08] Goran Bjorck and Uffe Haagerup. All cyclic p-roots of index 3, found by symmetry-preserving calculations. *arXiv preprint arXiv:0803.2506*, 2008.

[BL94] Tim Berners-Lee. Universal resource identifiers in www. 1994.

[BR08] Mohamed Barakat and Daniel Robertz. HOMALG –a meta-package for homological algebra. *Journal of Algebra and its Applications*, 7(03):299–317, 2008.

[BST10] Clark Barrett, Aaron Stump, and Cesare Tinelli. The smt-lib standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*, volume 13, 2010.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[GAP13] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.6.3*, 2013.

[Grä09] Hans-Gert Gräbe. The SYMBOLICDATA project. Technical report, Technical report (2000-2009), 2009.

[KFI+87] S. Katsura, W. Fukuda, S. Inawashiro, N. M. Fujiki, and R. Gebauer. Distribution of effective field in the ising spin glass of the $\pm j$ model at $T = 0$. *Cell Biophysics*, 11(1):309–319, 1987.

[KG00] Minoru Kanehisa and Susumu Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30, 2000.

[LNBB+06] Nicolas Le Novere, Benjamin Bornstein, Alexander Broicher, Melanie Courtot, Marco Donizelli, Harish Dharuri, Lu Li, Herbert Sauro, Maria Schilstra, Bruce Shapiro, et al. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic acids research*, 34(suppl 1):D689–D691, 2006.

[Neu12] Severin Neumann. Parallel reduction of matrices in gröbner bases computations. In Vladimir P. Gerdt, Wolfram Koepf, Ernst W. Mayr, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, volume 7442 of *Lecture Notes in Computer Science*, pages 260–270. Springer Berlin Heidelberg, 2012.

[P+00] Steven Pemberton et al. Xhtml 1.0 the extensible hypertext markup language. *W3C Recommendations*, pages 1–11, 2000.

[S+08] William Stein et al. SAGE: Open source mathematical software, 2008.

[SEW12] Satya Swarup Samal, Hassan Errami, and Andreas Weber. PoCaB: a software infrastructure to explore algebraic methods for bio-chemical reaction networks. In *Computer Algebra in Scientific Computing*, pages 294–307. Springer, 2012.

[SST12] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. Introducing starexec: a cross-community infrastructure for logic solving. *Comparative Empirical Evaluation of Reasoning Systems*, page 2, 2012.