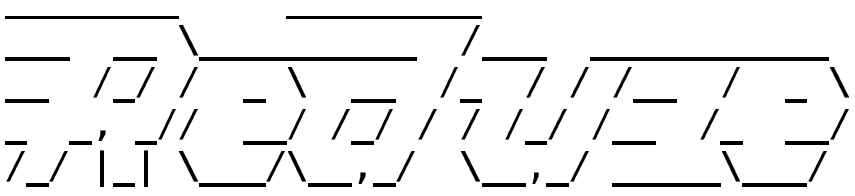

Reduze 2
Tutorial

A. von Manteuffel
C. Studerus

February 8, 2012

Contents

1 Introduction

`Reduze` is a computer program for reducing Feynman integrals to master integrals employing a variant of Laporta's reduction algorithm.

This tutorial describes version 2 of the program. New features include the distributed reduction of single topologies on multiple processor cores. The parallel reduction of different topologies is supported via a modular, load balancing job system. Fast graph and matroid based algorithms allow for the identification of equivalent topologies and integrals.

The user documentation of `Reduze` 2 is split in the following way:

concepts: The basic concepts and algorithms `Reduze` 2 relies on are described in our paper.

tutorial: This tutorial provides practical help with the installation of the program and explains the general usage of the program along specific examples.

reference: The actual reference user documentation is provided by the online help of the program executable (`reduze -h`).

The source package of `Reduze` and the latest version of this tutorial can be found at the web page `http://projects.hepforge.org/reduze`.

2 Installation

2.1 Basic installation guide

First, here is the quick guide for the impatient. To install a minimal build of Reduze 2 on a personal computer running Linux:

1. install CMake and GiNaC using the package manager of your distribution
2. download the Reduze source package and unpack it (replace 2.0.0 by the actual version number):

```
cd /tmp
tar -xzf reduze-2.0.0.tar.gz
```

3. configure, build, test and install (replace /home/mustermann/myprograms with the actual path you use for your program installations):

```
cd /tmp/reduze-2.0.0
cmake -DCMAKE_INSTALL_PREFIX=/home/mustermann/myprograms
make -j 2
make check # optional
make install
```

The rest of this chapter gives more details on these steps, describes how to install Reduze with its full feature set and addresses user defined locations of external libraries.

2.2 External dependencies

Reduze was developed on Linux with the GNU C++ compiler. We expect Reduze to work with minor modifications also on other Un*x flavors and reasonably up-to-date compilers. More specifically, we rely on ISO C++ 1998 including its template functionality and on POSIX functions, in particular for file and directory handling. Reduze depends on several external programs and libraries, which are described in the following.

In order to build and run Reduze, at least these two packages must be available:

CMake: This tool is used to build the program. Versions 2.6 and 2.8 are known to work. This package can be easily installed with a package manager on many Linux distributions.

GiNaC: The GiNaC library [1] is used by for algebraic manipulations. Versions 1.5.5 and 1.6.2 are known to work. This library and its header files can be easily installed with a package manager on many Linux distributions. Newer version might require the user to compile and install its own version.

The following packages are optional:

MPI: In order to profit from distributed computations, Reduze 2 must be run as an MPI program. We rely on the MPI-1 standard, a compatible implementation is available via the package manager of many Linux distributions. OpenMPI 1.4.1 and 1.4.3 are known to work. The build option USE_MPI controls whether to build an MPI program or a normal serial program.

Berkeley DB: The system of equations to be reduced can be stored in a Berkeley DB. This feature is useful to handle systems of equations which do not fit into main memory and to recover the full information of an aborted run using transactions. Note that the database can only used during the run, the final reduction results are always written to plain ASCII files. Versions 4.8.26 and 5.1.25 are known to work. Please note that in addition to the library itself the STL interface db-stl (see e.g. header file dbstl_common.h) is needed, which is not provided by all Linux distributions. If you compile the database yourself, please use the Berkeley DB configure switch --enable-stl. The Reduze build option USE_DATABASE controls whether to link against Berkeley DB.

Fermat: It is possible to use the program Fermat [2] (instead of GiNaC) for GCD calculations. Version 4.09 was tested in its 64-bit version. Fermat is available as an executable program only and not as library. The build option USE_FERMAT controls whether a small wrapper code should be included in Reduze to allow for the runtime option to use Fermat. It should be safe to leave this option on its default value ON.

Please note the CMake build flags USE_* described above only determine the *capabilities* of the Reduze executable. Additional options given at runtime control whether and how a specific feature should actually be employed for a specific computation. These runtime options will be listed in the online help only as far as the specific build supports them.

2.3 Advanced build options

We assume you already installed all libraries you want to use in Reduze and downloaded the source code of Reduze. First unpack the sources:

```
cd /tmp
tar -xzf reduze-2.0.0.tar.gz
```

An out-of-source build with MPI support may be configured with

```
mkdir /tmp/reduze-build
cd /tmp/reduze-build
cmake -DCMAKE_INSTALL_PREFIX=/home/mustermann/myprograms \
      -DUSE_MPI=ON /tmp/reduze-2.0.0
```

Build options specific to `Reduze` can be configured by adding the following `CMake` options:

```
-DUSE_MPI=ON
-DUSE_DATABASE=ON
-DUSE_FERMAT=ON
```

You can tell `CMake` to look for external libraries and headers not only automagically in the "usual directories" but also below the directory `/usr/local/extras` using the flags

```
-DCMAKE_INCLUDE_PATH=/usr/local/extras/include
-DCMAKE_LIBRARY_PATH=/usr/local/extras/lib
```

More ways to control `CMake` can be found in the `CMake` documentation, see in particular `http://cmake.org/cmake/help/cmake-2-8-docs.html#section_Variables`. It might avoid confusion to

```
rm -rf /tmp/reduze-build/*
```

in between different attempts to e.g. tell `CMake` to use the right one out of different versions of the same libraries or when changing the values of the `USE_*` flags. As the next-to-final resort you might want to have a look at the `CMakeLists.txt` files in the directories of the `Reduze` source directories.

After successful configuration you may want to compile the project with e.g. up to 2 compilations in parallel using

```
make -j 2
```

The program may be tested using

```
make check
```

or, for distributed `MPI` execution, using

```
make check_mpi
```

These tests might take a few minutes to be performed.

Finally, you can install the program executable and some additional files like application examples:

```
make install
```

Please feel free to contact us if you encounter problems building `Reduze`.

3 Usage

3.1 Getting started

To work with `Reduze` the user must provide configuration files and a job file. `Reduze` comes with a set of application examples which are copied to `'share/reduze/examples'` upon installation of `Reduze`. Probably the easiest way to get started is to copy one of the example directories

```
cd ~
cp -r myprograms/share/reduze/examples/example_1 reduze_try_1
```

and run

```
cd reduze_try_1
reduze jobs_reduction.yaml
```

Here we assumed a non-`MPI` build of `Reduze` was installed to `/myprograms`. In case of a `MPI` run you would replace the last command by something like

```
mpirun -np 5 reduze jobs_reduction.yaml
```

Details will be given in the following.

3.2 Configuration files

Configuration files define common input data such as the kinematics and integral families to use. `Reduze` treats the current working directory as the project directory. Configuration files must be located in a subdirectory `'config'` of the project directory. The following configuration files are mandatory:

```
config/kinematics.yaml
config/integralfamilies.yaml
```

while these configuration files are optional:

```
config/feynmanrules.yaml
config/global.yaml
```

All configuration files use the `YAML` format.

3.2.1 Kinematics

The file `kinematics.yaml` defines the kinematics and should contain a `YAML` document with a map. The map should contain only one entry, with key `kinematics` (to be taken as literal string) and a value of type `kinematics` (documented via the on-line help with `reduze -h kinematics`). Usually, this file should not be altered after `Reduze` used it for the first time to ensure consistency with existing results. The variable names of the symbols must start with an alphabetic letter. If `Fermat` is used they even must start with a lower case letter. An example for a $2 \to 2$ process is:

```
kinematics:
  incoming_momenta: [p1, p2]
  outgoing_momenta: [p3, p4]
  momentum_conservation: [p4, p1 + p2 - p3]
  kinematic_invariants:
    - [mt, 1]
    - [s,  2]
    - [t,  2]
  scalarproduct_rules:
    - [[p1,p1],  0]
    - [[p2,p2],  0]
    - [[p3,p3],  mt^2]
    - [[p1+p2, p1+p2], s]
    - [[p1-p3, p1-p3], t]
    - [[p2-p3, p2-p3], -s-t+2*mt^2] # == u
  symbol_to_replace_by_one: mt
```

3.2.2 Integral families

The file `integralfamilies.yaml` defines the integral families and should contain a `YAML` document with a map. The map should contain only one entry, where the key is `integralfamilies` (to be taken as literal string) and the value is a sequence of elements of type `integralfamily` (documented via the on-line help with `reduze -h integralfamily`). An ordering for the integral families and the corresponding integrals is inferred from the order in which the integral families are specified in the sequence (increasing from begin to end). For best performance, integral families with a lot of permutation symmetries should come first in the sequence (see section **??**). Usually, existing definitions in this file should not be altered after `Reduze` used them it for the first time to ensure consistency with existing results. However, it is possible to add new integral families *at the end* of the sequence without risking inconsistencies for existing reduction results. An example for a two-loop double box is:

```
integralfamilies:
  - name: planarbox
    loop_momenta: [k1, k2]
    propagators:
      - [ "k1", 0 ]
      - [ "k2", 0 ]
      - [ "k1-k2", 0 ]
      - [ "k1-p1", 0 ]
      - [ "k2-p1", 0 ]
      - [ "k1-p1-p2", 0 ]
      - [ "k2-p1-p2", 0 ]
```

```
    - [ "k1-p3", "mt^2" ]
    - [ "k2-p3", "mt^2" ]
  permutation_symmetries:
    - [ [ 1, 6 ], [ 2, 7 ] ]
    - [ [ 1, 2 ], [ 4, 5 ], [ 6, 7 ], [8, 9] ]
```

3.2.3 Crossed integral families

From a user-defined integral family several additional families are generated automatically by permutations of the external momenta. The *allowed crossings* are permutations which do not mix external momenta with different masses. Permutations between incoming and outgoing momenta involve a sign but are still allowed if they have the same mass. A crossing leads to a transformation of the kinematic invariants. If different crossings have the same change in the kinematic invariants they belong to the same class of *equivalent crossings*. Special crossings are the *identity crossing* and the crossings equivalent to the identity crossing. The former does not permute any external momenta or kinematic invariants and the latter does permute external momenta but leaves the kinematic invariants untouched.

The external momenta (joint list of incoming and outgoing momenta) of the underlying kinematics of an integral family are numbered in the given order (starting from 1). They are assumed to belong to external legs with the same leg numbers. The name of an integral family with crossed kinematics with respect to a user–defined "source integral family" `FAM` is given by the cycle notation of the permutation; e.g. the permutation $(1,2)(3,4,5)$ of the external legs leads to the family `FAMx1p2x3p4p5`, or simply `FAMx12x345` if there are no ambiguities.

Crossed integral families are used for target sectors in the sector mappings file (see section 3.10) to allow to identify sectors which only differ by a crossing.

`Reduze` never performs a reduction for crossed sectors but instead creates such reduction results from the corresponding uncrossed sector, if needed.

3.2.4 Feynman rules

The file `feynmanrules.yaml` defines Feynman rules and should contain a `YAML` document with a one-element map, where the key is `feynmanrules` and the value is of type `feynmanrules`. For more information see the on-line help with `reduze -h feynmanrules` and the Feynman rules file included in the source code package.

3.2.5 Global configurations

The file `global.yaml` collects other global configuration data. It should contain a `YAML` document with a map. At the moment, the only supported key is `paths` with a value of type `paths` (see the on-line help with `reduze -h paths`). It defines paths to external programs, e.g. to activate `Fermat` support.

3.3 Job files

`Reduze` executes jobs given in a job file. The job file name is given as argument when invoking `Reduze`. The file should contain a `YAML` document with a map. One element is mandatory for the map: it has the key `jobs` and its value is a sequence. Each element of this sequence should be a one-element map where the key is the name of a specific job and the value is of the corresponding job type. An example for a job file to perform reductions, select results and export them is:

```
jobs:
  - setup_sector_mappings: {}
  - reduce_sectors:
      sector_selection:
        select_recursively: [ [planarbox, 182] ]
      identities:
        ibp:
          - { r: [t, 5], s: [0, 1] }
  - select_reductions:
      input_file: "myintegrals"
      output_file: "myintegrals.sol"
  - export:
      input_file: "myintegrals.sol"
      output_file: "myintegrals.sol.inc"
      output_format: "form"
```

The job file can also contain advanced run options not be described in this section.

A reference list of available job types is obtained via the on-line help with `reduze -h jobs`. The help output will look similar to:

```
List of available job types:

apply_crossings:                  Generates reduction results for crossed sectors.
cat_files:                        Concatenates files.
collect_integrals:                Collects all integrals appearing in the input file.
compute_diagram_interferences:    Computes interferences of diagrams.
compute_differential_equations:   Computes derivatives of integrals wrt invariants.
export:                           Exports to FORM, Mathematica or Maple format.
extract_database_contents:        Extracts intermediate results from aborted reduction.
find_diagram_shifts:              Matches diagrams to sectors via graphs.
find_diagram_shifts_alt:          Matches diagrams to sectors via combinatorics.
generate_identities:              Generates identities like IBPs for given seeds.
generate_seeds:                   Generates integrals from a sector.
insert_reductions:                Inserts reductions in expressions.
normalize:                        Simplifies linear combinations and equations.
print_reduction_info_file:        Analyzes reductions in a file.
print_reduction_info_sectors:     Analyzes reductions available for sectors.
print_sector_info:                Prints diagrams and other information for sectors.
reduce_files:                     Reduces identities in given files.
reduce_sectors:                   Reduces integrals from a selection of sectors.
run_reduction:                    Low-level job to run a reduction.
select_reductions:                Selects reductions for integrals.
setup_sector_mappings:            Finds shifts between sectors via graphs.
setup_sector_mappings_alt:        Finds shifts between sectors via combinatorics.
sum_terms:                        Sums terms.
test:                             Performs some tests.
verify_same_terms:                Verifies two files contain the same terms.
```

Help for, say, the job `reduce_sectors` is obtained with `reduze -h reduce_sectors`.

Usually the first job to be executed should be the job `setup_sector_mappings`. This job determines various properties of the integral families, which are used to speed up reductions and eliminate ambiguities in the result. It is possible to list several jobs in one file or to distribute them among different job files which are executed in different runs for the same project directory.

3.4 Online help

`Reduze` comes with a built-in reference documentation for data types used in configuration files or job files. Invoking

```
reduze -h
```

shows how to access the on-line documentation. In particular, for a given keyword help is provided by

```
reduze -h KEYWORD
```

In case of an MPI it might be necessary to replace `reduze` with `mpirun -np 1 reduze`.

The help message will also print a template with default values which might be copy-and-pasted into the user's configuration or job file. Please note that additional modifications such as spaces in front of the lines might be necessary to use these templates at a specific position of the user's `YAML` file.

3.5 YAML format

Configuration and job files in `Reduze` 2 use the `YAML 1.2` format [7].

For a quick idea about the `YAML` language, let us give the following examples. A sequence of integers is given in standard notation as

```
- 2
- 5
- 10
```

or, alternatively, in flow notation as

```
[ 2, 5, 10 ]
```

A map of keys to strings is described by

```
name: "Max Mustermann"
city: "Irgendwostadt"         # this is a comment
```

where `name` is a key and `Max Mustermann` its value etc. The alternative flow notation reads

```
{ name: "Max Mustermann", city: "Irgendwostadt" }
```

YAML allows these structures to be nested, where different levels in the hierarchy are marked by indentation with *leading spaces*. An example for a map of keys to sequences is

```
bosons:
  - "gluon"
  - "photon"
fermions:
  - "electron"
  - "top quark"
```

We strongly recommend *not to use any tabulator characters* in the configuration or job files.

3.6 MPI build

In order to execute an MPI build of Reduze, you should start the executable via the mpirun (or mpiexec) script of your MPI implementation. A typical call to execute the jobfile jobs_reduction.yaml with 5 MPI processes would be

```
mpirun -np 5 reduze jobs_reduction.yaml
```

Please note that Reduze uses dedicated manager processes for the organization of the calculation:

- per run: one process is needed for the central job center
- per reduction of a sector: one process is needed for the job's manager, at least one process is needed for a worker

In a typical application, we therefore recommend to just start *more* MPI *processes* than the number of processor cores to be used ("overloading"). The central job center process will idle almost all of the time, it should be safe not to reserve a core for it. For small numbers of worker processes we also do not recommend to reserve a core for the manager. As an example, to reduce a selection of sectors on a computer with a double-core CPU, using 5 MPI processes via

```
mpirun -np 5 reduze jobs_reduction.yaml
```

would allow all load balancing mechanisms to be exploited: reduction of two sectors in parallel or reduction of a single sector on two cores. It might help to explicitly tell your MPI environment not to waste CPU while waiting for messages. Such settings are implementation specific, for OpenMPI you could add the command line option `--mca mpi_yield_when_idle 1`.

The project directory should ideally be located on a fast disk, since potentially large amounts of data will be read from or written to it. In its current version, all Reduze processes need access to the project directory. While this should not be an issue on a single machine, for a cluster this requires using a distributed file system.

3.7 Fermat support

In a typical reduction a considerable amount of the computation time is spent for normalizing the coefficients of loop integrals within the identities using polynomial GCD computations. Per default, `GiNaC` is used for these computations. Optionally the algebra program `Fermat` [2] may be used as a drop-in replacement to speed-up the calculations.

The default build options of `Reduze` allow to directly use an external `Fermat` executable by specifying its path in the file `config/global.yaml` similar to:

```
paths:
  fermat: /home/mustermann/myprograms/apps/fermat/ferls
```

The `Fermat` executable should lie in the same directory as the directory `BACKWARDS` of the `Fermat` package. Please check the web-page of `Fermat` for an executable working for your specific system and remember that this program is not free software. Currently, `Fermat` support is limited to handling reduction equations and does not cover advanced linear combinations containing functions.

The `Fermat` binary is offered in two version: a dynamically linked version and a statically linked version. In our tests, the statically linked `Fermat` executable worked more reliably than the dynamically linked version.

In some cases `Fermat` terminates with an error message containing the string "number in trial_poly_divide". This is a failure of a certain `Fermat` routine, which can be turned off with the command `&(_t=0);` in the start up file in the `BACKWARDS` directory of `Fermat`.

Note: when using `Fermat`, *variable names* (e.g. for the dimension and for kinematic invariants) are required to begin with a *lower case* letter.

3.8 Directories

The following sub directories of the project directory are known to `Reduze`:

`config` contains the user input files.

`graphs` contains `DOT` files for graphs of sectors. With the `Graphviz` tools (e.g. `dot` or `neato`) it is possible to generate images in `PostScript` and other formats from them.

`log` contains log files.

`reductions` contains equations in a `Reduze` specific `ASCII` format. Each file contains reductions for a specific sector, the files are named according to `reduction_FAM_T_ID` for a sector of integral family `FAM` with `T` denominators and identification number `ID`.

`sectormappings` contains files with sector mappings for the different integral families in `YAML` format. These files store zero sectors and shift relations between different sectors. Usually they are generated by the job `setup_sector_mappings`, but an advanced user might also supply rules obtained from external sources.

`tmp` contains temporary files.

Typically, the user will only have to create the `config` directory with the input files, all other directories are output directories and will be created by `Reduze`.

3.9 Data file formats

`Reduze` uses a native `ASCII` based format for data files with

- equations
- linear combinations
- lists of integrals

In order to use results with another software, the job `export` may be used to convert data files into an appropriate output format. The following export formats are available:

- `FORM` format
- `Mathematica` format
- `Maple` format

The native `Reduze` formats should be self explaining for the most part. They are intended to be easily analyzable with the usual `GNU` command line tools such as `grep`, `sed`, `awk` and `wc`. Equations and linear combinations are separated by a line containing a semicolon. For lists of integrals each line describes an integral. An equation is written in form of a sum of terms which is implied to be zero (that is, all terms are on the same side of the equality sign). Usually an equation is normalized such that the coefficient of the leading integral is 1. A linear combination is a sum of terms which has a name but is not necessarily zero. A text line which describes an integral is (up to space characters) of the form

```
FAM  T ID  R S  E1 E2 ... En
```

where `FAM` denotes the integral family, `T` the number of different denominators, `ID` the identification number of its sector, `R` the number of denominator exponents in the integrand, `S` the number of numerator exponents in the integrand, `E1` the exponent of the first propagator in the integrand etc. Note that a positive $E1$ means a factor in the denominator of the integrand, please see our article for more details.

Various jobs require a file with a list of integrals as input. They can be given in the native `Reduze` format described above, or a more brief version where integrals are specified as

```
FAM  E1 E2 ... En
```

Additional space characters don't matter. Alternatively, the format used for the export of integral lists to `Mathematica` may be used in input files.

3.10 Setup sector mappings

The sector mapping files of the integral families contain information about the sectors and their relations. These files can be setup by using one of the jobs `setup_sector_mappings` or `setup_sector_mappings_alt`. They also can be edited by hand. The following entries occur in the sector mapping files:

- `name:` name of the integral family
- `zero_sectors:` specifies zero sectors
- `sectors_without_graph:` specifies unphysical sectors
- `sector_relations:` shifts of the loop momenta with $|det| = 1$ that map sectors to lower (target) sectors
- `sector_symmetries:` shifts of the loop momenta with $|det| = 1$ that map a sector to itself

The job `setup_sector_mappings` sets up the files by first constructing graphs for sectors and then using graph and matroid based algorithms to derive shift and symmetry relations. Zero sectors are found during graph construction, e.g. when a sector does not have enough independent propagator momenta, or when the the corner integral of the sector is reduced to zero by a small explicit reduction.

An important fact is that a sector can have different graphs which all have the same (squared) momenta assigned. The graphs can be sorted lexicographically w.r.t. to their canonical label (a unique representation, chosen to be an adjacency list). The graph that is constructed for a sector is chosen to be the minimal one w.r.t. to its canonical label. The default canonical label is computed by allowing permutations of external nodes. Graphs with external edges can be isomorphic up to a crossing which leave the canonical label invariant, e.g. if two external edges are both attached to the same vertex. To choose a unique graph with external edges a second canonical label is computed by fixing the external nodes for all graphs which are minimal w.r.t. the canonical label with permutation of all nodes.

To find the minimal graph of a sector two methods are provided. With the default options,

```
minimize_graphs_by_twists: true
construct_minimal_graphs: false
```

one graph per sector is constructed and then further processed by a matroid based algorithm that performs all relevant twists, see our paper. The minimal graph is then chosen to represent the sector. The second method, activated with

```
construct_minimal_graphs: true
```

generates all possible graphs by construction and the minimal one is chosen. Since no twists have to be performed anymore they can be turned off with:

```
minimize_graphs_by_twists: false
```

This second algorithm can take longer than using the twists. It can be used to check the twist algorithm.

Once a unique graph per sector has been found they can be tested for isomorphisms such that shifts of the loop momenta can be derived. Shifts between isomorphic sectors which have an absolute value of the determinat not equal to one are rejected. Therefore, the set of *target sectors*, sectors to which other sectors are mapped to, can still contain sectors which are isomorphic as graphs. The target sectors also may contain some zero sectors.

When deriving shifts between sectors with external legs the two isomorphic graphs can still differ by a relative crossing of the external legs. Then, the derived shift of the loop momenta maps the sector to the crossed target sector, see section 3.2.3. `Reduze` does not perform reductions for integrals of crossed sectors but instead derives those reduction results from the reduction of the uncrossed sectors by tranforming the kinematic invariants of the coefficients of the integrals accordingly. Reduction results for crossed sectors will be written to disk only if the involved crossing is not equivalent to the identity crossing. Integrals of crossed sectors, where the crossing is equivalent to the identity, are immediatly replaced by the correponding uncrossed integrals since the kinematic invariants are not changed. The relative crossing between two isomorphic graphs does not have to be unique. E.g. if some external legs are attached to the same vertex the derived relative crossing still can be altered by a permutation of those two external legs. Such different crossings can be found by inspecting the node symmetry group of one of the graphs and choosing the minimal crossing. This can be useful if the minimal crossing is equivalent to the identity crossing. With the default option

```
minimize_target_crossings: true
```

a minimal crossing for the shift target sector will be found.

The sector symmetries, shifts in the loop momenta which map a sector to itself, are derived by comparing the node permutation of the symmetry group (automorphism group of a graph) and permuting multi-edges. The symmetry group is calculated by allowing permutations of external legs and rejecting all node permutations which lead to a crossing not equivalent to the identity. Therefore, the symmetry shifts can also contain permutations of external momenta but they do not alter the kinematic invariants. This method does not guarantee that *all* sector symmetries are found.

The job `setup_sector_mappings_alt` does not construct graphs but tries to find shifts between sectors with a combinatorial shift–finder algorithm. This method usually takes much longer but ensures that all shifts with determinant equal to one have been found. It is useful for tests and works also for sectors without graph representation.

3.11 Treatment of crossings and sector relations in reductions

Crossings and sector relations can be exploited by `Reduze` to restrict the explicit reduction of (IBP) identities to a minimal number of sectors. Reductions for further, "redundant" sectors will be derived from these explicit reduction results. Moreover, if the integral families are defined accordingly, the performance for the remaining explicit system solving can benefit from high permutation symmetries (few integrals) and natural variables (no crossed kinematics).

The following table describes how `Reduze` obtains reductions, e.g. when performing the job `reduce_sectors`.

zero sector	crossing of family	sector ID	treated by	on disk
yes	any	any	direct elimination	no
no	non-minimal	any	map to minimal crossed	no
no	minimal	minimal	apply crossing to uncrossed	yes
no	minimal	non-minimal	apply permutation or shift	no
no	no	minimal	explicit reduction	yes
no	no	non-minimal	apply permutation or shift	no

It should not be important to know these details, if one extracts reductions via the job `select_reductions`.

3.12 Resume and combine reduction runs

`Reduze` has capabilities to resume aborted runs. In order to use these features, one should set the flag

```
conditional: true
```

for the jobs in the job files. With this setting it should be possible to continue an aborted run by just restarting `Reduze` with the same job file. In case of an MPI build the process numbers may differ between the runs. While per default existing result files will just be overwritten, the `conditional` option tells `Reduze` not to regenerate an already existing result file. This mechanism is based on the existence of files alone and ignores any details on *how* the file was generated. To avoid trouble due to incompletely written files, data files are always written to a file with `.tmp` suffix first and then renamed.

It is even possible to resume a reduction for a single sector from an aborted run. For this to work, `Reduze` must be build with database support and the reduction job has to be setup to use the database with transactions enabled. Note that it might take quite some time to recover a large database from an aborted run.

It can be useful to incrementally compute reductions. For instance, one can first compute reductions for integrals with up to $s = 3$ numerator powers, and in a second run extend the solutions to cover also integrals with $s = 4$. The easiest way to achieve this is to rename the `reductions` directory from the $s \leq 3$ run to some other name and

supply this name via the `alternative_input_directory` option to `reduce_sectors` for the new run with $s = 4$ identities. The option `alternative_input_directory` may also be used to feed additional identities from other sources to the reduction of a selection of sectors.

3.13 QGRAF input

`Reduze` can be used to shift loop momenta such that Feynman diagrams generated by `QGRAF` [6] match sectors of the user defined integral families. `Reduze` requires the list of Feynman diagrams to be given in `YAML` format. In order to generate this input file with `QGRAF`, an appropriate `QGRAF` style file `reduze.sty` is supplied. It is known to work with `QGRAF` 3.1.1. For specific applications in QCD or QED, interferences of loop with tree diagrams can be computed with `Reduze` up to insertion of masters. Please see example 2 for reference.

3.14 Choice of master integrals

`Reduze` has built-in rules for the integral ordering. For sectors with more than one master integrals these rules will automatically select a particular basis for the master integrals which are the "lowest" unreducible integrals of the sector. This choice is obviously not unique and one typically wishes to select specific integrals as masters. This might be because one knows solutions for some integrals or the solutions are easier to obtain because e.g. the differential equations decouple in some way. `Reduze` offers features which effectively allow to perform a basis change. The idea is that the main reduction work is performed using the predefined ordering. At a later stage the existing reductions may be translated to the new conventions by specifying a file with `preferred_integrals` (typically the master integrals chosen by the user), which will be considered "lower" than any integral not in this list. This is demonstrated in example 3.

3.15 Advanced run options

The file job file may contain these further options for its top level `YAML` map in addition to the mandatory `jobs` entry:

```
timeout: 3600         # in seconds, terminates program after approx. 1 hour
max_parallel_jobs: 2  # does not run more than 2 jobs in parallel
time_interval_analysis: 120   # in seconds, rebalances workers each 2 minutes
max_workers_release: 5        # reassigns at most 5 workers each 2 minutes
```

3.16 Examples

We provide the following examples to demonstrate how different tasks can be solved with `Reduze`:

`1_reductions`: This example demonstrates how to perform a reduction and export results for a user specified list of integrals. `Reduze` is not designed to directly generate reductions for a list of integrals alone. Instead, it reduces ranges of integrals and the user can pick the reduction he actually needs later on. Note that the job `select_reductions` will provide reductions also for many integrals for which no explicit solution is written to the `reductions` directory, such as for integrals from zero sectors or integrals whose reductions are easy to generate from other reductions.

`2_diagrams`: This example demonstrates how to compute interference terms from Feynman diagrams generated by `QGRAF`.

`3_masters`: This example demonstrates how to compute differential equations for master integrals. The jobs used in the example are somewhat more involved than necessary to serve as a template also for more complicated sectors. For a sector with several master integrals the job file allows to select the integrals in the file `masters.curr.m` as masters such that also the necessary basis change will be performed.

To run an example, please copy the corresponding directory and execute `make` in the directory. The examples are also the basis for the automated check targets used by `Reduze`.

Bibliography

[1] C. W. Bauer, A. Frink and R. Kreckel, "Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language," arXiv:cs/0004015, `http://www.ginac.de` .

[2] R. H. Lewis, "Computer Algebra System Fermat," `http://www.bway.net/lewis/` .

[3] J.A.M. Vermaseren, Symbolic Manipulation with `FORM`, Version 2, CAN, Amsterdam, 1991; "New features of FORM" [math-ph/0010025].

[4] Wolfram Research, Inc., Mathematica, Version 7.0, Champaign, IL (2008).

[5] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter, J. McCarron and P. DeMarco, "Maple 10 Programming Guide," Maplesoft, Waterloo ON, Canada, (2005).

[6] P. Nogueira, "Automatic Feynman graph generation," J. Comput. Phys. 105 (1993) 279-289.

[7] `YAML`: YAML Ain't Markup Language, `http://yaml.org` .

[8] J. Beder, `yaml-cpp`, "A YAML parser and emitter for C++," `http://code.google.com/p/yaml-cpp` .