LMU-ASC 51/11
TUM-HEP 818/11
DESY-11-187
LPSC-11220

The Orbifolder:
A Tool to study the Low Energy Effective Theory of Heterotic Orbifolds

H.P. Nilles[a], S. Ramos-Sánchez[b], P.K.S. Vaudrevange[c,d,e], A. Wingerter[f]

[a]*Bethe Center for Theoretical Physics and Physikalisches Institut der Universität Bonn, Nussallee 12, 53115 Bonn, Germany*
[b]*Department of Theoretical Physics, Physics Institute, UNAM, Mexico D.F. 04510, Mexico*
[c]*Deutsches Elektronen-Synchrotron DESY, Notkestraße 85, 22607 Hamburg, Germany*
[d]*Physik-Department T30, Technische Universität München, James-Franck-Straße, 85748 Garching, Germany*
[e]*Arnold-Sommerfeld-Center for Theoretical Physics, Theresienstraße 37, 80333 München, Germany*
[f]*Laboratoire de Physique Subatomique et de Cosmologie, UJF Grenoble 1, CNRS/IN2P3, INPG, 53 Avenue des Martyrs, F-38026 Grenoble, France*

Abstract

The `orbifolder` is a program developed in `C++` that computes and analyzes the low-energy effective theory of heterotic orbifold compactifications. The program includes routines to compute the massless spectrum, to identify the allowed couplings in the superpotential, to automatically generate large sets of orbifold models, to identify phenomenologically interesting models (e.g. MSSM-like models) and to analyze their vacuum-configurations.

Keywords: Orbifold, string compactification, extra dimensions, particle spectrum, MSSM

Program Summary

Program title: `orbifolder`
Program obtainable from: `http://projects.hepforge.org/orbifolder/`
Distribution format: tar.gz
Programming language: `C++`
Computer: Personal computer
Operating system: Tested on Linux (Fedora 15, Ubuntu 11, SuSE 11)
Word size: 32 bits or 64 bits
External routines: None
Dependencies: Boost, GSL
Typical running time: Less than a second per model.
Nature of problem: Calculating the low energy spectrum of heterotic orbifold compactifications.
Solution method: Quadratic equations on a lattice; representation theory; polynomial algebra.

1. Introduction

String theory is a candidate for a consistent unified quantum theory of gravity and gauge interactions and could thus provide us with an ultraviolet completion for models of particle physics. The search for 4-dimensional string vacua resembling the standard model (SM) (or its supersymmetric extension (MSSM)) is therefore one of the central questions in string theory research. Over the years, a wide landscape of 4-dimensional string models has emerged and it remains to be seen how particle physics phenomena can be incorporated within the scheme of string theory. Some hints point to a unification of gauge couplings within the framework of exceptional groups (as e.g. E_8) but a direct road from strings to particle physics has not yet been identified. It is perhaps the time to step back, collect and classify existing model constructions and try to identify properties relevant for a description of nature.

Here we present and analyze a specific approach that was studied already in the 1980s and has led to interesting results since then: orbifold compactification [1, 2, 3] of the heterotic strings [4, 5]. The reason for the success of this approach is "computability" paired with geometric intuition. Exact tools of conformal field theory are here at our disposal [6, 7] that are usually not available in approaches based on compactification on smooth manifolds. Besides, in particular $E_8 \times E_8$ as a gauge group allows a perturbative inclusion of the standard model gauge group (as well as possibly grand unification).

A toroidal orbifold is flat with the exception of a number of fixed points or fixed tori. It gives rise to a picture called the heterotic brane world scenario [8, 9, 10]: fields can either live in the 10-dimensional bulk (untwisted sector) or can be localized at these fixed points or fixed tori (twisted sectors). The relative location of these fields as well as the local gauge structure determines many properties of the 4-dimensional string vacua and is the source of geometric intuition for model building. The orbifold point is a point of enhanced symmetries (in the moduli space of compactification) and those symmetries might be relevant for a description of nature. Models of particle physics seem to require many (discrete) symmetries, e.g. flavour symmetries or symmetries to prevent too fast proton decay. Some of these symmetries could be slightly broken to provide us with small parameters relevant for the description of hierarchies as e.g. observed in the spectrum of quark and lepton masses. This supports our hope that orbifold compactification is not just an approximation with increased "computability", but that it provides a realistic compactification: nature might have chosen to live close to the orbifold point with enhanced symmetries.

Explicit orbifold model constructions in the last 5 to 10 years have been extremely successful [11, 12, 13, 14] (see [15, 16] for earlier reviews). Many of the properties of particle physics can be incorporated in the scheme. This includes grand unification, gauge-Yukawa unification, satisfactory Yukawa textures, solutions to the μ-problem, the creation of hierarchies and a successful incorporation of neutrino Majorana masses. Discrete symmetries are identified to solve the flavour problem and to avoid too fast proton decay. These properties have been identified in the so called Minilandscape (based on the Z_6-II orbifold) [17, 18, 19] and subsequent work [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33].

The search for such models requires a computer assisted scan that incorporates the rules for a consistent string theory construction (as e.g. modular invariance). The purpose of this work is to make the tools and techniques available to the public. We hope that this will give more people the opportunity to contribute to this exciting field of model constructions.

We present the `orbifolder`, a program developed in `C++` that allows the calculation of the low-energy spectrum of heterotic orbifold constructions. The program includes routines to compute the massless spectrum and to identify the allowed couplings of the superpotential. It allows the construction of arbitrary orbifold models, the identification of phenomenologically interesting models and a classification of their vacuum configurations.

The `orbifolder` can be considered in some aspects as the stringy analog of programs such as `SoftSusy`,

SuSpect and SPheno: The latter are devoted to the detailed computation of particle spectra, interactions and phenomenological quantities, using as input a high scale supersymmetric model and imposing low-energy constraints. Analogously, the orbifolder takes as a starting point the 10D heterotic strings and, provided some geometric input describing the features of the six compactified dimensions, computes the (massless) spectrum, interactions and symmetries of the resulting lower-energy effective 4D field theory.

After a short introduction to heterotic orbifold compactification in Section 2, we explain how to download and install the C++ program in Section 3. Section 4 discusses the explicit recipe to run the program, while in Section 5 we conclude and give an outlook for future research. Technical details are relegated to the appendices and to the webpage [39, §Complementary notes].

2. Heterotic Orbifold Compactifications

In this section we give a brief introduction to heterotic orbifolds, in order to introduce our notation and conventions used in the orbifolder. For more details on orbifold compactifications, we refer to the reviews [15, 16, 11, 13, 14].

In the context of heterotic string compactifications, we define an orbifold as the quotient of six-dimensional real space $\mathbb{R}^6$ divided by the so-called *space group* S, where the quotient is taken using the equivalence relation $X \sim gX$ for all $g \in S$ and $X \in \mathbb{R}^6$. More specifically, the space group is chosen to consist of two parts:

- discrete rotations that form the so-called *point group* P. For simplicity, we choose P to be Abelian. To obtain $\mathcal{N} = 1$ supersymmetry in 4D, P is either $\mathbb{Z}_M$ or $\mathbb{Z}_M \times \mathbb{Z}_N$ generated by rotation matrices θ and ω, where we use $\omega = 1$ for $\mathbb{Z}_M$. These matrices can be represented by so-called *twist vectors* v_1 and v_2 that give the three rotational phases in the three complex planes and the sum over all entries integer to ensure $\mathcal{N} = 1$. For example, $v_1 = \left(0, \frac{1}{3}, \frac{1}{3}, -\frac{2}{3}\right)$ and $v_2 = 0$ for the $\mathbb{Z}_3$ orbifold.

- translations generated by the vectors $e_\alpha \in \mathbb{R}^6$, for $\alpha = 1, \ldots, 6$. They form a 6D lattice denoted by Γ and hence define a six-torus. Elements of P must map the lattice Γ to itself.

In detail, an element of the space group is of the form $g = \left(\theta^k \omega^l, n_\alpha e_\alpha\right) \in S$, where $k, l \in \mathbb{Z}$, $n_\alpha \in \mathbb{Z}$ (or in some cases $n_\alpha \in \mathbb{Q}$) and summation over $\alpha = 1, \ldots, 6$. It acts on $X \in \mathbb{R}^6$ as $gX = \theta^k \omega^l X + n_\alpha e_\alpha$. Using these definitions, we can deal with all 6D Abelian and toroidal orbifolds including the cases of roto-translations and freely acting involutions (see e.g. Ref. [34]).

Due to modular invariance, the geometric action of the space group S has to be embedded into the gauge degrees of freedom of the heterotic string, denoted by X^I, $I = 1, \ldots, 16$ in the bosonic formulation. We restrict ourselves to the case of shift embedding, where $\theta \hookrightarrow V_1$, $\omega \hookrightarrow V_2$ and $e_\alpha \hookrightarrow W_\alpha$ for $\alpha = 1, \ldots, 6$. Then, the action of S on $X \in \mathbb{R}^6$ induces an action on X^I as

$$g\,X = \theta^k \omega^l X + n_\alpha e_\alpha \quad \Rightarrow \quad g\,X^I = X^I + k V_1^I + l V_2^I + n_\alpha W_\alpha^I \,, \tag{1}$$

where $I = 1, \ldots, 16$. The vectors V_1, V_2 and W_α are called *shifts* and *Wilson lines*, respectively. They are constrained by modular invariance [2, 35, 36], e.g.

$$M\,(V_1^2 - v_1^2) = 0 \bmod 2 \quad \text{and} \quad N_\alpha W_\alpha^2 = 0 \bmod 2 \,, \tag{2}$$

where M is the order of $\mathbb{Z}_M$ and N_α the order of W_α. The combined group formed by the space group and its action on the gauge degrees of freedom is called the *orbifold group*.

From a model-building standpoint, we have now introduced all the input data to define a heterotic orbifold model: the space group S (consisting of the point group P and the lattice Γ) and its embedding as shifts V_1, V_2 and Wilson lines W_α.

We close this section with a very brief summary of the construction of massless spectra of heterotic orbifolds. Given the input data, one distinguishes between two kinds of closed (and massless) strings on the orbifold: first of all ordinary closed strings, also called *untwisted strings*, being the remnants of the 10d $E_8 \times E_8$ or SO(32) vector multiplet and the gravity multiplet. Secondly, there are closed strings from the *twisted sectors* which close only up to the action of the orbifold group. To construct them, one needs to identify the inequivalent non-trivial space group elements as the constructing elements of twisted strings: i.e. for $g \in S$ one can define a twisted boundary condition $X(\tau, \sigma + \pi) = gX(\tau, \sigma)$ on the string world-sheet. Then, using standard CFT techniques, the equations for massless right- and left-movers with boundary condition g read

$$\frac{\left(q + v_g\right)^2}{2} - \frac{1}{2} + \delta c = 0 \quad \text{and} \quad \frac{\left(p + V_g\right)^2}{2} + \tilde{N} - 1 + \delta c = 0 \tag{3}$$

where p is from the $E_8 \times E_8$ or $\mathrm{Spin}(32)/\mathbb{Z}_2$ weight lattice, q from the vector or spinor weight lattice of SO(8), δc denotes the shift in the zero-point energy and $\tilde{N}$ the number operator of left-moving oscillators. Furthermore, we define the *local twist* $v_g = kv_1 + lv_2$ and the *local shift* $V_g = kV_1 + lV_2 + n_\alpha W_\alpha$. In the final step, one builds massless states as tensor products of massless right- and left-movers such that they are invariant under the full orbifold action, i.e.

$$|q + v_g\rangle_\mathrm{R} \otimes |p + V_g\rangle_\mathrm{L} \quad \text{or} \quad |q + v_g\rangle_\mathrm{R} \otimes \tilde{\alpha}|p + V_g\rangle_\mathrm{L} \,, \tag{4}$$

where $\tilde{\alpha}$ denotes possible oscillator excitations. We define the shifted momenta $q_\mathrm{sh} = q + v_g$ and $p_\mathrm{sh} = p + V_g$, where p_sh describes the transformation properties under gauge transformations. The states Eq. (4) correspond to massless fields of the 4D effective field theory. They carry gauge charges (from p_sh), discrete R charges, modular weights (from q_sh and possible oscillator excitations) and discrete non-R charges (from the constructing element $g \in S$).

3. Download and Installation

The minimal requirements for compiling the `orbifolder` are the Boost C++ Libraries version ≥ 1.0 [37] and the GNU Scientific Library (GSL) version ≥ 1.9 [38]. All components should come preinstalled on a standard Linux distribution. If this should not be the case, they can easily be installed.

On a `yum`-based distribution like e.g. Fedora, the command "`yum -y install gsl gsl-devel boost boost-devel`" will install the corresponding libraries (recommended). Alternatively, one can also directly install from source or use the other download options available [37, 38].

The `orbifolder` is free software under the copyleft of the GNU General Public License and can be downloaded from [39]:

http://projects.hepforge.org/orbifolder/

To install the program, download the file `orbifolder-1.0.tgz` to a directory of your choice, open a shell and enter the following commands at the prompt:

```
tar xfvz orbifolder-1.0.tar.gz
cd orbifolder-1.0
./configure
```

```
make
make install
```

Note that the version number "1.0" may change over time and should be substituted accordingly. The first line unpacks the tar-ball and creates a subdirectory structure with the source code. The second line changes to the installation directory. The third line starts the configuration script that checks system requirements and generates the `Makefile`. The fourth line compiles the code, and finally the fifth line installs it on your system. After successful compilation and installation, the main program (named `orbifolder`) will be available in the current directory. We have disabled custom installation using the `--prefix` switch in the configuration script. The main program `orbifolder` can simply be copied to the directory of the user's choice by the standard shell commands. Detailed installation instructions can also be found in the README file in the installation directory and on the website [39].

We have tested the installation process on

- a 32-bit system running Linux Ubuntu 11.04 with Boost 1.42 and GSL 1.14,
- a 64-bit system running Linux SuSE 11 with Boost 1.36 and GSL 1.11,
- a 64-bit system running Linux Fedora 15 with Boost 1.46 and GSL 1.14,

and ascertained that our code compiles correctly. Should there arise any problems during the installation, we request that the user send us the file `config.log` and the output of the `make` command by email (orbifolder@projects.hepforge.org).

4. How to run the program

There are three main ways to gain access to the `orbifolder`: through the `prompt`, through a web interface and directly through the `C++` source code. In the following we will present them in detail.

4.1. The prompt

The `orbifolder` can be controlled using a Linux-style command line called the `prompt`. The `prompt` offers an interactive access to almost all variables and functions of the `orbifolder`. It has the structure of a file system where orbifold models appear as directories. In the following we explain how to start and use the `prompt`.

4.1.1. How to get started

We begin with a small tutorial, see Tab. 4 in the additional material [39, §Complementary notes] for a sample input and output. In general, the `prompt` can be started using the command `./orbifolder` or `./orbifolder [model file]`. In the former case, no model is loaded automatically. In the latter case, the details of a model contained in the plain-text-based `model file` are loaded (Further properties of *model files* are explained in Section 4.4.2). As an example, run the program using the command

```
./orbifolder modelZ3.txt
```
(5)

with parameter `modelZ3.txt` in order to load the standard embedding model of the $\mathbb{Z}_3$ orbifold from this file. Having started the program, one enters the `prompt` in its main directory `/>`. The command

```
dir
```
(6)

lists all commands and subdirectories of the current directory. In our example, there is one subdirectory in the main directory `/>` called `Z3StandardEmbedding` which corresponds to the $\mathbb{Z}_3$ model loaded. In general, a (sub-) directory `A` can be accessed using the command `cd A`. In our example,

```
cd Z3StandardEmbedding
```
(7)

results in the output `/Z3StandardEmbedding>` from where one can access, analyze or change the details of the $\mathbb{Z}_3$ standard embedding model. Again, type in the command `dir` to see all commands and subdirectories of the current directory. For all orbifold model directories there are five subdirectories,

```
/model>, /gauge group>, /spectrum>, /couplings> and /vev-config>,
```
(8)

containing commands of the respective category:

- `/model>`: Print and change the input data of the current orbifold model. See Appendix B.2.2.
- `/gauge group>`: Print details on the gauge group, change the U(1) basis and find accidental U(1) symmetries. See Appendix B.2.3.
- `/spectrum>`: Print details on the spectrum of massless fields. See Appendix B.2.4.
- `/couplings>`: Create and analyze the superpotential and the resulting effective mass matrices. See Appendix B.2.5.
- `/vev-config>`: Define and analyze various vev-configurations. See Appendix B.2.6. Each configuration is specified by the distinction between observable and hidden sector of the gauge group and the assignment of labels and vacuum expectation values to the fields (labels are assigned in a subdirectory called `/labels>`, see Appendix B.2.7).

Again, one can access these directories using the `cd` command. For example, try `cd gauge group` to enter the subdirectory `/gauge group>` and use the commands `print gauge group` and `print simple roots` to see the gauge group $(E_6 \times SU(3) \times E_8)$ and (a choice of) the corresponding simple roots. In order to go back one directory one uses the command `cd ..` at the prompt. Next, try the subdirectory `/spectrum>` and use the command `print summary` to get a summary table of all massless matter fields. The command `cd ~` is used to go back to the main directory `/>`. Further standard commands of the `prompt` are described in Tab. 1; see also Appendix B for a glossary of commands.

command	description
dir	display commands and subdirectories of the current directory
cd A	change the current directory to `A` (if `A` exists)
cd ..	go back one directory
cd ~	go back to the main directory `/>`
exit	exit the program if no process is running; use the parameter `orbifolder` to enforce exit (also inside a script)

Table 1: Some standard commands in the `prompt`.

4.1.2. How to create new orbifold models

Being in the main directory `/>` of the `prompt` new orbifold models can be accessed basically in three ways:

- `load orbifolds(Filename)`: Load orbifold models from a model file. For example, load the $\mathbb{Z}_6$–II orbifold MSSM of [40] using the command `load orbifolds(modelBHLR.txt)`.
- `create orbifold(A) with point group(M,N)`: Create a $\mathbb{Z}_M$ or $\mathbb{Z}_M \times \mathbb{Z}_N$ orbifold named `A` by specifying M and N (set $N = 1$ for $\mathbb{Z}_M$). After entering the new directory using `cd A`, one is asked to specify more details on the model like shifts and Wilson lines, see Appendix B.2.2.
- `create random orbifold from(A)`. See below.

Orbifold models can be created randomly by using the command

```
create random orbifold from(A)
```
(9)

in the main directory `/>`. The parameter `A` must be either the name of an existing (loaded or previously created) orbifold model or `*` for any existing orbifold model in the `orbifolder`. The command starts a new process that runs in the background so that one can continue to work with the `prompt` (see Appendix B.1.5 for more details on processes). One can specify several optional parameters:

- `if(...)`: Specify the desired properties of the model: `inequivalent` in order to choose only models with inequivalent spectra and `SM`, `PS` or `SU5` for models with a (net) number of `X` generations of Standard Model (`SM`), Pati-Salam (`PS`) or SU(5) gauge group plus vector-like exotics, where `X` is 3 by default and can be changed using the parameter `Xgenerations`; c.f. the command `analyze config` in Appendix B.2.6.
- `save to(Filename)`: Save the models with the desired properties to a model file.
- `use(1,1,0,1,...)`: Eight digits for two shifts plus six Wilson lines; either 1 if the corresponding shift/Wilson line shall be taken from model `A`, or 0 if it shall be created randomly.
- `#models(X)`: Define how many models (with the desired properties, if specified) shall be created randomly. Use `#models(all)` to create as many models as possible. If `#models(X)` is not used, only one model shall be created.
- `print info`: Print a short summary of the spectrum immediately when a new model with the desired properties has been found.
- `load when done`: Load the created models into the `orbifolder` after the process has finished.
- `do not check anomalies`: Use this parameter to speed up the process.
- `compare #couplings of order(X)`: If only `inequivalent` models are saved, this parameter refines the comparison between two models: compare in addition the number of all superpotential couplings up to the specified order `X`. Slows down the process considerably.

Examples. A typical example of how to use this command looks like

```
create random orbifold from(Z3StandardEmbedding) if(inequivalent)
    save to(Z3NewModels.txt) use(1,1,0,0,0,0,0,0) #models(10) print info
```
(10)

executed from the main directory `/>`. In this case a new process is started that constructs new $\mathbb{Z}_3$ orbifold models using both shifts (i.e. $V_2 = 0$) but no Wilson lines from model `Z3StandardEmbedding`, saves only inequivalent models to a model file named `Z3NewModels.txt`, prints a brief summary of each new model and stops after creating ten inequivalent models. In a second example, ten random models of `SM` (Standard Model) type are created starting from the $\mathbb{Z}_6$–II orbifold MSSM of [40] by using the command

```
create random orbifold from(Z6IIOrbifold_BHLR) if(inequivalent SM)
    save to(Z6IINewMSSMModels.txt) use(1,1,1,1,1,1,0,0) #models(10)
    print info load when done
```
(11)

in the main directory `/>`. The parameter `use(1,1,1,1,1,1,0,0)` specifies that only the Wilson lines W_5 and W_6 are created randomly, i.e. the shifts and other Wilson lines are taken from the original model. Furthermore, only `inequivalent` standard models (`SM`) are printed, saved to file `Z6IINewMSSMModels.txt` and finally loaded into the `orbifolder` after the process has finished. Note that these MSSM models should be part of the Mini-Landscape [17, 19].

4.2. The web interface

The `orbifolder` can also be accessed through a user-friendly web interface. The `orbifolder on-line` makes extensive use of the `prompt` explained in Section 4.1 rendering it available to any user with access to an internet browser, such as Mozilla Firefox version ≥ 3.6, Google Chrome version ≥ 2.1, and Internet Explorer ≥ 8.0. Consequently, the program is also available to users of all kinds of smartphones.

One of the advantages of the web interface is that one does not need to install any program on the local computer to be able to use most of the functions of the `orbifolder`. Another advantage is that it can be executed from platforms that work with the most popular operating systems (without further auxiliary applications): Windows, Linux and Mac.

On the less bright side, one shortcoming of this version is that it is not recommended to execute time-consuming instructions since short interruptions in the internet service may affect the results. Furthermore, a command running during more than 60 minutes is disabled automatically to avoid overload of the server. Finally, for security reasons, commands involving file manipulation are extremely limited. Specifically, the parameters and commands `@begin/@end print to file(Filename)`, `save to(Filename)`, `load/save couplings(Filename)`, `load/save labels(Filename)` are disabled.

The `orbifolder on-line` can be used on our main page [39]

http://projects.hepforge.org/orbifolder/

which redirects to any of our mirror servers. To gain access, you must click on the link `orbifolder on-line`. This starts an `orbifolder` session on the selected server. The new page consists of three parts:

- `History`: The result of the latest instructions is shown. The button `download history` provides an RTF file containing the full history, i.e. not only the result of the latest instructions, but the result of all the commands used during the current session. The user can resize this window.

- `List of Commands`: This is the input area, where the commands of the `prompt` are typed. To execute them, it is necessary to click on the button `execute commands`. An additional help in this section is the

button `upload commands`. This button can be used when files (not larger than 100Kb) in plain-text format containing (lists of) admissible commands have been prepared. The button `download commands` provides an RTF file containing the list of all the commands typed during the active `orbifolder` session.

Lists of useful commands and their use is provided in Section 4.1 and Appendix B.

- `Help`: The bottom part contains a list of all available commands for the current directory. Each command in the displayed list is a link to a more precise description of its use.

To terminate an `orbifolder on-line` session, it suffices to click on the upper button `EXIT`. The buttons provided on the resulting page allow the user to download the full history of the current session and the complete list of successfully executed commands.

Occasionally, errors in the input data may cause the `orbifolder` to crash. In those cases, the web interface shall close your session, giving you the opportunity to download the list of instructions (button `download commands`) to be used if you restart the `orbifolder`. Please, make sure that the downloaded list of commands does not contain the instructions that led to the failure of the program. We encourage users to contact the authors reporting any failure in the program, preferably by using the link `contact us` on the main page of the web interface.

4.3. The C++ source code

The `orbifolder` is written in `C++`, distributed over several files. Many physical quantities, as briefly introduced in Section 2, have been encapsulated into classes, for example

- `CSpaceGroup` for the space group S and `CSpaceGroupElement` for its elements $g \in S$,
- `CTwistVector` for the twists v_i and `CShiftVector` for the corresponding shifts V_i,
- `CWilsonLines` for the set of six Wilson lines W_α,
- `COrbifoldGroup` for the orbifold group,
- `CMasslessHalfState` for finding massless left- and right-movers, i.e. solutions to Eq. (3),
- `CHalfState` for the weights of `CMasslessHalfState` sorted with respect to their transformation properties under the elements of the centralizer,
- `CState` for orbifold-invariant tensor products of massless left- and right-movers, see Eq. (4),
- `COrbifold` for the full orbifold compactification,
- `CField` for a field of the effective 4-dimensional theory,
- `CMonomial` for (gauge invariant) monomials of fields corresponding to $D = 0$ solutions,

and many more. In addition there are technical classes. For example, there are several classes devoted to group theory (like `dynkin`, `freudenthal`, `gaugeGroupFactor` and `gaugeGroup`), the class `CAnalyseModel` contains functions to analyze models for their phenomenological properties, the class `CPrint` contains all printing commands and the class `CPrompt` contains the source code of the `prompt`. As there are in total more than 40 classes we cannot explain them in detail here. We give further details of some of the more important classes in Appendix A and a short example program in Appendix A.1.

4.4. Files defining an orbifold model

There are two files that define an orbifold model: i) The *geometry file* contains, as the name suggests, the geometrical information about the orbifold, such as the space group, and ii) the *model file* contains shifts and Wilson lines, i.e. the action of the orbifold on the gauge sector of the heterotic string. In the following we give some more details. Examples are given in the additional material [39, §Complementary notes].

4.4.1. The geometry file

The geometry file basically contains information about

- the space group, i.e. the order of the twist(s), the six lattice vectors of Γ and the generators of the space group,
- the discrete (R and non-R) symmetries of the orbifold (important for the computation of allowed superpotential couplings),
- the inequivalent fixed points specified by their constructing elements $\left(\theta^k\omega^l, n_\alpha e_\alpha\right)$ and
- for each constructing element a list of centralizer elements, i.e. elements $h \in S$ with $[h, g] = 0$.

We give two examples in the additional material [39, §Complementary notes]: Tab. 1 gives a detailed description of a geometry file using the $\mathbb{Z}_3$ example and Tab. 2 explains how to create a new geometry file of, for example, model (1-3) of Ref. [34].

4.4.2. The model file

The model file contains a list of orbifold models, where each model is specified by

- the name of the model (will be used as the name of the corresponding directory in the `prompt`),
- the name of the geometry file,
- the type of heterotic string (i.e. $\mathrm{Spin}(32)/\mathbb{Z}_2$ or $\mathrm{E}_8 \times \mathrm{E}_8$),
- two shifts V_1, V_2 (where the $V_2 = 0$ for $\mathbb{Z}_M$ orbifolds),
- six Wilson lines W_α,
- optionally, the parameters of (generalized) discrete torsion a, b_α, c_α and $d_{\alpha\beta}$ as defined in Ref. [36],
- optionally, some of the U(1) generators,
- optionally, one can specify a script that is executed automatically after the model has been loaded.

An example for the case of a $\mathbb{Z}_3$ orbifold with standard embedding is given in the additional material [39, §Complementary notes].

5. Conclusions and outlook

With the tools provided here it should be possible to thoroughly investigate the landscape (in particular the MSSM-like landscape) of orbifold compactification of the heterotic string. This "heterotic braneworld" provides a coherent geometric picture of MSSM-like models. Crucial properties of the scheme depend on the geography of fields in extra dimensions. This leads to the concept of "Local Grand Unification" and a geometric understanding of e.g. Yukawa couplings, the μ-term, neutrino masses and proton decay. Detailed properties of models can be computed reliably within the context of conformal field theory.

The models constructed here should be compared (and possibly related) to other regions of the MSSM-like landscape, as e.g. fermionic formulations [41, 42, 43], tensoring of conformed field theories [44], smooth compactifications of the heterotic string [45, 46, 47, 48, 49, 50], type II (intersecting) brane-models [51, 52, 53, 54, 55], M- and F-theory constructions [56, 57, 58, 59, 60, 61, 62]. It would be interesting to identify similarities and differences of the corresponding schemes. All these cases rely on a (sometimes hidden) geometric interpretation that defines the properties of the models such as the appearance of grand unification, values of Yukawa-couplings and the existence of hierarchies.

One of the important observations in the framework of the heterotic braneworld concerns the crucial role played by discrete (gauge) symmetries. They arise as remnants of the gauge symmetry as well as symmetries due to the special location of fields in extra dimensions. They control properties of the scheme, as e.g. flavor universality and the question of proton stability. At the orbifold point we encounter an enhancement of discrete symmetries and particle spectra. These symmetries are a basic ingredient of model building. Slightly broken, they might give us an explanation for the appearance of hierarchies in particle physics (as e.g. the ratio of Yukawa couplings). At the orbifold point we can rely on exact conformal field theory techniques that could be useful to understand the blow-up procedure [6, 7, 63, 64, 65, 30, 66] of orbifold singularities in a controlled way and thus connect to smooth compactifications.

With a better knowledge of the MSSM-landscape we might hope to relate the various constructions and improve the calculational power in those models where we still have to rely on an effective low-energy supergravity and/or large volume approximation. The future of the field requires reliable calculational tools (as e.g. conformed field theory techniques) which are up to now only applicable in some corners of the landscape. But we might be lucky and nature might have chosen to live close to such a corner.

Acknowledgments

We would like to thank Michael Blaszczyk, Nana Geraldine Cabo Bizet, Stefan Förste, David Grellscheid, Tomáš Ježo, Stefan Groot Nibbelink, Michael Ratz, Fabian Rühle and Jesús Torrado-Cacho for useful discussions. P.V. would like to thank LMU Excellent for support. This material is partly based upon work done at the Aspen Center for Physics and supported by the National Science Foundation under Grant No. 1066293. This research was supported by the DFG cluster of excellence Origin and Structure of the Universe, the SFB-Transregio TR33 "The Dark Universe" and TR27 "Neutrinos and Beyond" (Deutsche Forschungsgemeinschaft) and the European Union 7th network program "Unification in the LHC era" (PITN-GA-2009-237920). S. R-S. was partially supported by CONACyT project 82291 and DGAPA project IA101811.

Appendix A. Glossary of relevant classes

The `orbifolder` makes extensive use of the class structure offered by `C++`. The information of an orbifold model is distributed according to the following classes.

Class `CAnalyseModel`. The class `CAnalyseModel` contains several functions that analyze the phenomenology of orbifold models, mainly for the cases of the Standard Model, Pati Salam or SU(5) gauge group.

Class `CField`. A `CField` object contains all physical information about a massless field, such as the representation, the U(1) charges, the $q_{\rm sh}$ charges, the localization and its vev. In addition, it contains a list of indices which specify its weights $p_{\rm sh}$ (stored in the member variable `vector<CVector> Weights` of the associated `CMasslessHalfState` object).

Class `CFixedBrane`. A `CFixedBrane` object contains all information about all (untwisted or twisted) strings with constructing element $g = \left(\theta^k\omega^l, n_\alpha e_\alpha\right) \in S$. The left-moving part of the string is computed using the local shift $V_g = kV_1 + lV_2 + n_\alpha W_\alpha$. Hence, the solutions of the equation for massless left-movers, Eq. (3), are stored here in a vector of `CMasslessHalfState` objects, one entry for each different choice of oscillator excitation. After the massless solutions have been identified, they are sorted with respect to their centralizer eigenvalues and stored in a corresponding vector of `CHalfState` objects (one entry for each different choice of $\tilde{N}$ with different eigenvalues). In the last step, the massless right-moving `CHalfState` objects from `CSector` are tensored together with the massless left-moving `CHalfState` objects stored here to form orbifold-invariant string states. These states are stored in `vector<CState> InvariantStates`.

Class `CHalfState`. A `CHalfState` object descends from a `CMasslessHalfState` object by sorting the massless solutions (i.e. the weights $q_{\rm sh}$ or $p_{\rm sh}$ for right- or left- movers) with respect to their centralizer-eigenvalues. The indices of the weights (as listed in `vector<CVector> Weights` of the corresponding `CMasslessHalfState` object) having the same eigenvalues `vector<double> Eigenvalues` are stored in `vector<unsigned> Weights`.

Class `CMasslessHalfState`. A `CMasslessHalfState` object stores the solutions of the equation for massless right- or left-movers, respectively, see Eq. (3). The constructor `CMasslessHalfState(MoversType Type, const S_OscillatorExcitation &Excitation)` needs two parameters: the first parameter `MoversType` can be either `LeftMover` or `RightMover` and the second one specifies the oscillator excitation. Then, one can call the member function `bool SolveMassEquation(const CVector &constructing_Element, const SelfDualLattice &Lattice)` to create the solutions of Eq. (3), where `constructing_Element` denotes the local twist v_g or the local shift V_g and `SelfDualLattice` can be either `E8xE8`, `Spin32` or `SO8`. The solutions are stored in `vector<CVector> Weights`.

Class `COrbifold`. A `COrbifold` object contains all information about a single orbifold compactification. The main member variable is `vector<CSector> Sectors`, i.e. a vector of M times N CSector objects, one for each sector of a $\mathbb{Z}_M\times\mathbb{Z}_N$ orbifold. The first element corresponds to the untwisted sector and the rest to the various twisted sectors.

Class `COrbifoldGroup`. A `COrbifoldGroup` object basically contains the space group and its gauge embedding as objects of class `CSpaceGroup`, `CShiftVector` and `CWilsonLines`. Furthermore, it contains a vector of all inequivalent constructing elements (`vector<COrbifoldGroupElement>`) and a corresponding vector of centralizer elements (`vector<vector<COrbifoldGroupElement> >`).

Class `CPrint`. The `CPrint` class contains all printing commands. For the constructor `CPrint(OutputType output_type, ostream *out)` one needs to specify the `OutputType`, being either `Tstandard`, `Tmathematica`, or `Tlatex`, and a `ostream` object to set the destination of the output, either to the screen using `&cout` or to a file using an `ofstream` object.

Class `CSector`. A `CSector` object contains all information about an untwisted or twisted sector. It is mainly specified by the local twist v_g (or in other words by k and l since $v_g = kv_1 + lv_2$). As the oscillator excitations and the right-moving part of the string only depend on the local twist, see Eq. (3), they are identical for all strings from a given sector. Hence, this data is stored in `CSector`.

Class `CSpaceGroup`. A `CSpaceGroup` object contains the details about the space group of an orbifold model, as explained in Section 2. All constructing and centralizer elements are stored in `CSpaceGroupElement` objects. Additionally, it includes the geometrical information of the compact space, such as the 6D lattice, its symmetries and the order of the associated Wilson lines.

Class `CState`. A `CState` object is basically an orbifold-invariant combination of a massless right-moving `CHalfState` object and a massless left-moving `CHalfState` object.

Appendix A.1. Example source code

We present a sample program that computes and analyzes the spectrum of the $\mathbb{Z}_6$–II orbifold model of [9]. In the source code distribution, the corresponding file is `src/examples/samplemain01.cpp`.

```
#include <stdio.h>
#include "cprompt.h"
using namespace std;

int main(int argc, char *argv[])
{
  ifstream in("modelKRZ_A1.txt");
  if((!in.is_open()) || (!in.good()))
    exit(1);

  CPrint Print(Tstandard, &cout);
  string ProgramFilename = "";

  COrbifoldGroup OrbifoldGroup;
  if (OrbifoldGroup.LoadOrbifoldGroup(in, ProgramFilename))// load from file
  {
    cout << "\n-> Model file \"modelKRZ_A1.txt\" loaded." << endl;
    COrbifold KRZ_A1(OrbifoldGroup);                      // create the orbifold
    cout << "-> Orbifold \"KRZ_A1\" created.\n" << endl;

    cout << "-> Print shift and Wilson lines:" << endl;
    Print.PrintShift(OrbifoldGroup.GetShift(0));
    Print.PrintWilsonLines(OrbifoldGroup.GetWilsonLines(), true);

    cout << "\n-> Print spectrum, first with and then without U(1) charges:" << endl;
    Print.PrintSummaryOfVEVConfig(KRZ_A1.StandardConfig);  // print with U(1)s
    SConfig VEVConfig = KRZ_A1.StandardConfig;             // create new vev-config.
    VEVConfig.ConfigLabel = "TestConfig";                  // rename new vev-config.
```

```
29      VEVConfig.SymmetryGroup.observable_sector_U1s.clear(); // change obs. sector
30      Print.PrintSummaryOfVEVConfig(VEVConfig);              // print without U(1)s
31
32      cout << "-> Analyze model:" << endl;
33      vector<SConfig> AllVEVConfigs;
34      bool SM = true;                                        // look for SM
35      bool PS = true;                                        // look for PS models
36      bool SU5 = true;                                       // look for SU(5) models
37      // analyze the configuration "KRZ_A1.StandardConfig" of "KRZ_A1"
38      // and save the result to "AllVEVConfigs"
39      CAnalyseModel Analyze;
40      Analyze.AnalyseModel(KRZ_A1, KRZ_A1.StandardConfig, SM, PS, SU5,
41      AllVEVConfigs, Print);
42      if (SM || PS || SU5) // if one of the three possibilities is true
43      {
44        cout << "-> Model has 3 generations plus vector-like exotics:" << endl;
45        const size_t s1 = AllVEVConfigs.size();              // print all new configs.
46        for (unsigned i = 0; i < s1; ++i)
47          Print.PrintSummaryOfVEVConfig(AllVEVConfigs[i], LeftChiral, true);
48      }
49    }
50    return EXIT_SUCCESS;
51  }
```

First, we define an `ifstream` object called `in` that contains the model file `modelKRZ_A1.txt`. Then, we define the orbifold group as a `COrbifoldGroup` object and load the content of `in`. If no error occurs, the orbifold is defined as a `COrbifold` object. After calling the constructor of `COrbifold` with a `COrbifoldGroup` parameter, the spectrum of the orbifold model is computed and checked for consistency. Next, we print the shift, the Wilson lines and the massless spectrum of the model using the `CPrint` class. In the last part, a `CAnalyseModel` object is constructed in order to analyze the phenomenological properties of the vev-configuration `KRZ_A1.StandardConfig`.

Appendix B. Glossary of commands

In this appendix, we give short explanations for all commands of the `prompt` and of the web interface. In Appendix B.1 we start with some concepts and general commands. Then, in Appendix B.2 we list all commands available in the various directories of the `prompt`.

Appendix B.1. Concepts and general commands

The basic quantities of the `prompt` are fields of the 4D effective field theory (Appendix B.1.1). In order to access them easily one can define sets of fields (Appendix B.1.2). Furthermore, gauge invariant monomials of fields are used to describe solutions of the $D = 0$ condition (Appendix B.1.3). For many commands dealing with fields one can use the parameter `if(condition)` to choose only those fields that fulfill the condition (Appendix B.1.4). Finally, this section describes the concept of processes (Appendix B.1.5), the use of vectors (Appendix B.1.6), how to change the typesetting to `mathematica` or to `latex` style (Appendix B.1.7), the use of system commands and variables (Appendix B.1.8).

Appendix B.1.1. Field labels

For a given orbifold model and vev-configuration, fields of the 4D effective field theory are tagged with labels, for example `q_1`, `q_2` and `q_3` for the three left-handed quark doublets. A label consists of a name (`q`) and a generation index (`1`, `2` and `3` in our example). One can access several fields simultaneously using their common name, for example `q` for the three quarks. Furthermore, one can access all fields of a given model using `*`. In addition, one can obtain the intersection of all fields named `A` but not named `B` using `A-B`. Examples for intersections are: `n-n_1` to get all fields named `n` except for `n_1` and `*-n` for all fields except for the ones named `n`.

Field labels are stored in the currently used vev-configuration of the orbifold model. They can be viewed and changed in the directory `/vev-config/labels>`, see Appendix B.2.7. Note that in a given vev-configuration one can define several labels for each field.

Finally, in `mathematica` typesetting, for example, the label `q_1` is displayed as `fldq1`.

Appendix B.1.2. Sets of fields

One can access several fields simultaneously not only by their field labels but also using sets of fields. These sets are stored in the currently used vev-configuration of the orbifold model. (Consequently, one cannot access a set in a different vev-configuration than in the one where it was created.) For more details on vacua, see Appendix B.2.6. Note that sets are on the same footing as field labels[1]. I.e. one can build intersections like:

- `A-B` for the intersection of two sets `A` and `B`,
- `*-A` for the intersection of all fields `*` and a set `A`,
- `A-q` for the intersection of a set `A` and all fields of name `q` or
- `q-A` for the intersection of all fields of name `q` and a set `A`.

The commands to create and manipulate sets are displayed in any orbifold model directory of the prompt using the command

```
help sets .
```
(B.1)

The commands are:

Command `create set(SetLabel)`. Create an empty set with name `SetLabel` and save it in the currently used vev-configuration. Optionally, this command allows for the parameters `from monomials` or `from monomial(MonomialLabel)` in which case all fields from either all monomials or only from monomial `MonomialLabel` will be inserted into the new set. See Appendix B.1.3 for more details on monomials.

Command `delete set(SetLabel)`. Delete the set `SetLabel` of the currently used vev-configuration.

Command `delete sets`. Delete all sets of the currently used vev-configuration.

Command `insert(fields) into set(SetLabel)`. Insert `fields` into the set `SetLabel`. Optionally, the parameter `if(condition)` can be used to insert only those `fields` into the set `SetLabel` that satisfy the condition. For details on `if(condition)` see Appendix B.1.4.

[1] Both, fields and sets of fields, will be denoted as `fields` in the explanations of the following sections.

command	description
create set(Test)	create an empty set named `Test`
insert(F_1 F_2 F_3 F_4 F_5) into set(Test)	insert `F_i`, `i` = $1, \ldots, 5$ into the set `Test`
remove(*) from set(Test) if(#osci. != 0)	remove fields with non-zero number operator $\tilde{N}$ (i.e. `F_5`)
print sets	print all sets

Table B.2: Short example for the use of set-commands in the directory `/Z3StandardEmbedding>` of the $\mathbb{Z}_3$ standard embedding model (using the standard labels `F_i` of the vev-configuration `TestConfig1`).

Command `print set(SetLabel)`. Print the content of the set `SetLabel`.

Command `print sets`. Print all sets defined in the currently used vev-configuration. One can use the optional parameter `if not empty` to print only the non-empty sets.

Command `remove(fields) from set(SetLabel)`. Remove `fields` from the set `SetLabel`. Optionally, the parameter `if(condition)` can be used to remove only those fields that satisfy the condition.

Command `#fields in set(SetLabel)`. Count the number of fields in the set `SetLabel`.

A short example showing some of the basic commands for sets is given in Tab. B.2.

Appendix B.1.3. Gauge invariant monomials

(Holomorphic) gauge invariant monomials (short: monomials) are used to describe solutions to the $D = 0$ supersymmetry condition [67, 68, 69, 70]. A (sub-)set of solutions can be found using the command `find D-flat(fields)` described in Appendix B.2.6. More details and examples can be seen using the command

```
help monomials
```
(B.2)

in any orbifold model directory.

Appendix B.1.4. If conditions

Many commands that deal with fields allow for the parameter `if(condition)` (or several copies thereof) so that only those fields are chosen that fulfill all the conditions. An explicit example was already given in Tab. B.2. In general, a condition consists of three parts:

- the left hand side gives the variable (e.g. `Q_i` for the fields `i`-th U(1) charge, `vev` for the fields vacuum expectation value or `#osci.` for the number of oscillators),
- the middle gives the comparison operator (e.g. `==` for equal or `!=` for unequal) and
- the right hand side gives a value (e.g. a rational number or 0).

More details and examples can be seen using the command

```
help conditions
```
(B.3)

in any orbifold model directory.

Appendix B.1.5. Processes

The following commands start new processes that run in the background so that one can continue to work with the `prompt`:

- `/> create random orbifold from(OrbifoldLabel)`
- `/A/couplings> create coupling(fields)`
- `/A/vev-config> find D-flat(fields)`
- `/A/gauge group> find accidental U1s`

Each process has an ID, the so called PID. Similar to the Linux command line one can see all running processes using the command `ps` and terminate a process with PID `A` using `kill(A)`. One can also kill all active processes using the command `kill(all)`. In a script the command `wait(X)` might be useful in order to check every `X` seconds if all processes have finished and to continue with the next commands afterwards. More details can be seen using the command

```
help processes
```
(B.4)

in any orbifold model directory.

Appendix B.1.6. Vectors

Many commands need a vector of rational numbers as a parameter. Examples include the commands `set shift V(i) = <16D vector>` and `set torsion b = <6D vector>`. In these cases there are several possibilities of how to write the vector. For example, the following forms of a `<4D vector>` are possible:

$$(1/3\ 1/1\ 0/1\ 0/1) \;=\; (1/3\ 1\ 0\ 0) \;=\; (1/3, 1, 0, 0) \;=\; (1/3, 1, 0\hat{}2) \;=\; 1/3(1\ 3\ 0\hat{}2) \tag{B.5}$$

In addition, for the first four forms of the example-vector, one can leave the brackets away.

Appendix B.1.7. Output for mathematica or in latex style

Often it is useful to transfer data from the `orbifolder` to mathematica, for example, in order to use `STRINGVACUA` [71], `SINGULAR` [72], `NonAbelianHilbert` [69, 70] or `DiscreteBreaking` [73, 74]. Therefore, many commands allow for the parameter

```
@mathematica
```
(B.6)

so that the output of the command will be printed in a mathematica compatible style (if available). For example,

```
print list of charges @mathematica
```
(B.7)

in the directory `/spectrum>`. Similarly, the parameter `@latex` can be used in order to get the output in latex code (again, if available). In addition, one can set the default typesetting to `mathematica`, `latex` or back to `standard` using the commands

`@typesetting(mathematica)`, `@typesetting(latex)` or `@typesetting(standard)`, (B.8)

respectively. Finally, the parameter `no output` can be used to suppress the output of the current command.

Appendix B.1.8. System commands and variables

System commands start with the symbol `@` and are used to change the output's style and destination. Moreover, the `prompt` allows for some pre-defined variables which are particularly usefull in scripts. They start and end with the symbol `$`.

Command `@typesetting(Type)`. Change the output's typesetting, see Appendix B.1.7.

Command `@begin print to file(A)`. Start printing output to file `A` and not to the screen. In contrast, one can use the parameter `to file(A)` so that the output of only the current command is printed to file, e.g. `print summary to file(Summary.txt)`.

Command `@end print to file`. Stop printing output to file.

Command `@status`. Display the destination of the output (e.g. `screen`) and the style of the typesetting (i.e. `standard`, `latex` or `mathematica`).

Variables. There are three pre-defined variables: `$OrbifoldLabel$`, `$VEVConfigLabel$` and `$Directory$`. When executed, a variable is replaced by a corresponding string, being the label of the current orbifold model, the label of the current vev-configuration or the path of the current directory, respectively. They are particularly usefull in scripts, e.g. used as `to file($OrbifoldLabel$.txt)`.

Appendix B.2. The directories

The structure of the `prompt` consists of a main directory `/>` and subdirectories that correspond to orbifold models. Each orbifold model directory has further subdirectories `/model>`, `/gauge group>`, `/spectrum>`, `/couplings>` and `/vev-config>`. They offer commands of the respective category. In this section we give an alphabetically ordered glossary of directory-commands and explain their use in detail.

Appendix B.2.1. The main directory `/>`

In the main directory one can basically create, load and save orbifold models.

Command `create orbifold(OrbifoldLabel) with point group(M,N)`. Create an empty orbifold model directory for an orbifold of specified point group orders (use `N`= 1 for $\mathbb{Z}_M$ orbifolds).

Command `create random orbifold from(OrbifoldLabel)`. Randomly create new orbifold models. Details are given in Section 4.1.2. More details and examples can be seen using the main directory's `/>` command

```
help create random .
```
(B.9)

Command `delete orbifold(OrbifoldLabel)`. Delete the orbifold model directory `OrbifoldLabel`.

Command `delete orbifolds`. Delete all orbifold model directories.

Command `load orbifolds(Filename)`. Load all orbifold models from the model file named `Filename`.

Command `load program(Filename)`. Load a script from file `Filename` and execute the commands contained in that file.

Command `save orbifolds(Filename)`. Save all orbifold models of the main directory to a model file named `Filename`.

Appendix B.2.2. The directory `/model>`

In the directory `/model>` the input data (e.g. point group, twists, shifts, Wilson lines, etc.) of the current orbifold model can be displayed and changed.

Command `create suborbifold with factor(i)`. Starting from an orbifold with space group S, one can create a so-called suborbifold based on a subgroup $S' \subset S$.

For $\mathbb{Z}_M$ orbifolds the subgroup $S' \subset S$ is specified by one number, i being a divisor of M. Denote the $\mathbb{Z}_M$ twist generator of the space group by $g \in S$. Then, the subgroup $S' \subset S$ is based on the twist generator $g^i \in S' \subset S$ which generates $\mathbb{Z}_{M/i}$. One can use the optional parameter `and(j)` to choose $g^i \in S' \subset S$ and $g^j \in S' \subset S$ (with i, j coprime and i, j divide M) to generator $\mathbb{Z}_{M/i} \times \mathbb{Z}_{M/j}$.

In the case of $\mathbb{Z}_M \times \mathbb{Z}_N$ orbifolds (with twist generators $g_1, g_2 \in S$) one has to specify two numbers `create suborbifold with factor(i,j)` (where i divides M and j divides N) for the new generator $g_1^i g_2^j$ of the subgroup $S' \subset S$. Again, one can use the optional parameter `and(k,l)` to specify a second generator $g_1^k g_2^l$.

Note that this command is particularly useful to analyze the 6D orbifold GUT limit of an orbifold model. For example, start with the $\mathbb{Z}_6$–II orbifold MSSM of [40]. Then the command `create suborbifold with factor(2)` will produce the 6D $\mathbb{Z}_3$ orbifold GUT limit as analyzed in [75].

Command `print available space groups`. Print a list of all geometry files compatible with the specified point group. The geometry files are searched by the `orbifolder` in the directory `/localdirectory/Geometry>` (of the local PC). For more details on the content of geometry files, see Section 4.4.1.

Command `print discrete symmetries`. Print the discrete (R and non-R) symmetries as defined in the geometry file. Note that R symmetries need to be defined in the geometry file in order to be used in the computation of allowed superpotential couplings.

Command `print discrete torsion`. Print the (generalized) discrete torsion parameters a, b_α, c_α and $d_{\alpha\beta}$ as defined in Ref. [36].

Command `print massless A-movers`. where `A` can be `left` or `right`. Print the massless left- or right movers before some of them are projected out by the action of the centralizer.

Command `print orbifold label`. Print the orbifold label (i.e. the name of the current orbifold directory).

Command `print point group`. Print the point group. The output reads, for example, `Point group is Z_3`.

Command `print shift`. Print the shift(s) as 16D vector(s). The output reads e.g.

```
V_1 = ( 1/3 1/3 -2/3 0 0 0 0 0) ( 0 0 0 0 0 0 0 0) .        (B.10)
```

Command `print space group`. First, print the point group and the root-lattice. Next, print the generators of the space group. The output reads e.g.

```
Space group based on Z_3 point group and root-lattice of SU(3)^3.      (B.11)
Generators are:
(1, 0) ( 0, 0, 0, 0, 0, 0)
(0, 0) ( 1, 0, 0, 0, 0, 0) ···
```

where `(k,l) (n_1, ..., n_6)` corresponds to the element $\left(\theta^k\omega^l, n_\alpha e_\alpha\right)$ of the space group. Note that roto-translations and freely-acting involutions are allowed as generators of the space group. For example in Ref. [34], one of the generators of the (0-2) model reads `(0,1) (0,0,0,0,1/2,0)` corresponding to $\left(\omega, \frac{1}{2}e_5\right)$ and one of the generators of the (1-1) model reads `(0,0) (0,1/2,0,1/2,0,1/2)` corresponding to $\left(\mathbb{1}, \frac{1}{2}(e_2+e_4+e_6)\right)$.

Command `print twist`. Print the twist(s) as four-dimensional vector(s). The output reads e.g.

```
v_1 = ( 0 1/3 1/3 -2/3) .
```
(B.12)

Command `print Wilson lines`. Print the relations among the Wilson lines (e.g. $W_1 = W_2$ for $\mathbb{Z}_3$), their order (e.g. order 3 for $3W_i \in \Lambda$ for $\mathbb{Z}_3$) and the Wilson lines themselves as 16D vectors. The output reads, for example,

```
Wilson lines identified on the orbifold:
W_1 = W_2, W_3 = W_4, W_5 = W_6
Allowed orders of the Wilson lines:  3 3 3 3 3 3
W_1 = ( 0 0 0 0 0 0 0 0) ( 0 0 0 0 0 0 0 0)
...
```
(B.13)

Command `print #SUSY`. Print the number of supersymmetry in 4D. The output reads, for example,

```
N = 1 SUSY in 4D.
```
(B.14)

Command `set heterotic string type(type)`. Define the 16D gauge lattice of the heterotic orbifold model. Here, `type` can be `E8xE8` or `Spin32`.

Command `set shift standard embedding`. Choose $V = (v^1, v^2, v^3, 0^{13})$ for $\mathbb{Z}_M$ orbifolds or $V_1 = (v_1^1, v_1^2, v_1^3, 0^{13})$, $V_2 = (v_2^1, v_2^2, v_2^3, 0^{13})$ for $\mathbb{Z}_M \times \mathbb{Z}_N$ orbifolds. (The notation 0^{13} means 13 times the entry "0".)

Command `set shift V = <16D vector>` *or* `set shift V(i) = <16D vector>`. Define the shift vector V of $\mathbb{Z}_M$ orbifold models or one of the two shift vectors V_i (with `i`$= 1, 2$) of $\mathbb{Z}_M \times \mathbb{Z}_N$ orbifold models as 16D vector, see Eq. (1). For more details on vectors see Appendix B.1.6.

Command `set torsion a = n/d, b = <6D vector>, c = <6D vector>` *or* `d = <15D vector>`. Set the (generalized) discrete torsion parameters as defined in [36], i.e. a, b_α, c_α and $d_{\alpha\beta}$ (for $\alpha,\beta = 1,\ldots,6$; $d_{\alpha\beta} = -d_{\beta\alpha}$ has 15 components). Note that the parameters are not checked for modular invariance and hence might cause inconsistent spectra. For more details on vectors see Appendix B.1.6.

Command `set WL W(i) = <16D vector>`. Define the Wilson line W_i as a 16D vector (with `i`$= 1,\ldots,6$), see Eq. (1). For more details on vectors see Appendix B.1.6.

Command `use space group(i)`. with `i`$= 1,\ldots$. Load the space group from the `i`-th geometry file, where the index `i` corresponds to the position in the list of geometry files as displayed using the command `print available space groups`.

Appendix B.2.3. The directory `/gauge group>`

In this directory one can print and change some details of the gauge group for the currently used vev-configuration. In more detail, one can display the U(1) generators and the simple roots, change the basis of U(1) generators, define a B-L generator and identify accidental U(1) symmetries of the superpotential. Note that all gauge-group-editing commands are not available in the vev-configuration `StandardConfig1`.

Command `delete accidental U1s`. Delete the accidental U(1) charges of all fields.

Command `find accidental U1s`. Take the superpotential (as far as it has been created in the directory `/couplings>`, see Appendix B.2.5) and identify its accidental U(1) symmetries. The command starts a new process that runs in the back of the `prompt`. The results are saved in the currently used vev-configuration. Presumably, the accidental U(1) symmetries will be broken explicitly by higher order terms, but nevertheless might be of phenomenological relevance, e.g. for the strong CP problem [27] and proton decay [31].

Optionally, one can use the parameter `fields with zero charge(fields)` in order to find only those accidental U(1) symmetries under which the fields of `fields` are uncharged. On a technical level, this is achieved by inserting, during this analysis, each field of `fields` as a linear term into the superpotential.

Command `load accidental U1s(Filename)`. Load accidental U(1) charges from a file named `Filename`.

Command `print anomalous space group element`. Print details on discrete anomalies using the discrete symmetries defined in the geometry file and identify the so-called anomalous space group element [76].

Command `print anomaly info`. Print details on gauge and gravitational anomalies and check their universality relations. In detail, in the case of $\mathcal{N} = 1$ SUSY in 4D, beside the pure non-Abelian anomalies, the relations[2]

$$\frac{1}{24}\mathrm{tr}\, Q_i = \frac{1}{6|t_i|^2}\mathrm{tr}\, Q_i^3 = \frac{1}{2}\mathrm{tr}\, \ell Q_i = \frac{1}{2|t_j|^2}\mathrm{tr}\, Q_j^2 Q_i = \begin{cases} \text{const.} \neq 0 & \text{if } i = 1, \text{i.e. } i = \text{anom} \\ 0 & \text{otherwise} \end{cases} \tag{B.15}$$

(with $i \neq j$) are verified, where t_i is a 16D vector corresponding to the i-th U(1) generator so that a field with shifted left-moving momentum p_{sh} carries the charge $Q_i = p_{\text{sh}} \cdot t_i$ and tr sums over the contributions from all massless left-chiral fields.

Command `print B-L generator`. Print the $\mathrm{U}(1)_{B-L}$ generator as a 16D vector.

Command `print FI term`. Print the Fayet-Iliopoulos term (i.e. $\mathrm{tr} Q_{\text{anom}}$ as in Eq. (B.15)), if there is an anomalous U(1).

Command `print gauge group`. Print the observable and hidden part of the gauge group for the currently used vev-configuration.

Command `print simple root(i)`. Print the `i`-th simple root as 16D vector.

Command `print simple roots`. Print (a choice of) simple roots of all non-Abelian gauge group factors as 16D vectors.

[2] In the `orbifolder`, the convention for the quadratic Dynkin index is such that $\ell = 1$ for the fundamental representation of SU(N) groups.

Command `print U1 generator(i)`. Print the `i`-th U(1) generator as 16D vector.

Command `print U1 generators`. Print all U(1) generators as 16D vectors.

Command `save accidental U1s(Filename)`. Save the accidental U(1) charges to a file named `Filename`.

Command `set B-L = <16D vector>`. Define $U(1)_{B-L}$ as a 16D vector. Since in the `orbifolder` all U(1) generators are demanded to be orthogonal to each other, but $U(1)_{B-L}$ is in general not orthogonal to hypercharge, B-L is stored as an additional vector. One can use the optional parameter `allow for anomalous B-L` if $U(1)_{B-L}$ is allowed to mix with the anomalous U(1). For more details on vectors see Appendix B.1.6.

Command `set U1(i) = <16D vector>`. Change the basis of U(1) generators by specifying a 16D vector as the i-th generator. The new generator must be orthogonal to all simple roots and to the j-th U(1) generator, for $j < i$. Note that the k-th U(1) generators with $k > i$ will be changed automatically such that, at the end, all generators are orthogonal to each other. For more details on vectors see Appendix B.1.6.

Appendix B.2.4. The directory `/spectrum>`

This directory offers access to all information about the massless spectrum. In detail, for each massless field one can obtain the SUSY multiplet type (i.e. for $\mathcal{N} = 1$ supersymmetry in 4D: `left-chiral`, `right-chiral`, `vector` and `modulus`), the localization (corresponding to its constructing space group element), the shifted left-moving momentum p_{sh}, the non-Abelian representation, the U(1) charges, the B-L charge (if defined), the shifted right-moving momentum q_{sh}, the oscillator excitations, the R charges, modular weights (if defined), the label of the field and finally its vev. Note that complex conjugate representations are printed as negative integers; for example, `-3` denotes the conjugate fundamental representation $\overline{\mathbf{3}}$ of SU(3).

Command `find potential blowup modes(fields)`. Print a list of potential blow-up modes considering `fields` only, i.e. print all `fields` for each fixed brane/point.

Command `find random blowup modes(fields)`. print a random list of blow-up modes `fields`, one per fixed brane/point. The result can be saved to a set `SetLabel` using the optional parameter `save to set(SetLabel)`.

Command `print(fields)`. Print some details of `fields`. There is one optional parameter, `with internal information`, that displays some internal information about how the fields' data can be accessed in the `C++` source code of the `orbifolder`.

Command `print all states`. Print details of all fields (including left-chiral superfields, vector superfields of the (non-Abelian) gauge bosons, moduli and their CPT-partners).

Command `print list of empty fixed branes`. Print a list of all fixed branes/points that potentially could carry left-chiral fields but are empty for the current orbifold model.

Command `print list of charges(fields)`. Print gauge and discrete charges of `fields`. For example,

```
( -1/3 -1/3 2/3 0 0 0 0 0) ( 0 0 0 0 0 0 0 0) ( -1/3 2/3 -1/3) ( 2 0 0 1) "F_10"
( -1/3 2/3 -1/3 0 0 0 0 0) ( 0 0 0 0 0 0 0 0) ( -1/3 2/3 -1/3) ( 2 0 0 1) "F_10"
( 2/3 -1/3 -1/3 0 0 0 0 0) ( 0 0 0 0 0 0 0 0) ( -1/3 2/3 -1/3) ( 2 0 0 1) "F_10"
```
(B.16)

for the field `F_10` of the $\mathbb{Z}_3$ orbifold with standard embedding. Each line consists of three parts:

- first the 16D shifted left-moving momentum p_{sh} (printed as two eight-dimensional vectors in the case of $\text{E}_8 \times \text{E}_8$),
- the R charges: $(R_1, R_2, R_3) = (-\frac{1}{3}, \frac{2}{3}, -\frac{1}{3})$ in the example,
- the charges with respect to the discrete non-R symmetries: $(k, n_1 + n_2, n_3 + n_4, n_5 + n_6,) = (2, 0, 0, 1)$ in the example,
- the label of the corresponding field,

where, in general, the R and non-R symmetries must be specified in the geometry file, see Section 4.4.1. Note that it can be very helpful to use the optional parameter `@mathematica` for this command in order to transfer the information about the fields to mathematica, see Appendix B.1.7.

Command `print summary`. Print the gauge group and a summary table of the massless matter fields. Important optional parameters are:

- `of sectors`
- `of fixed points`
- `of fixed point(X)`
  where the fixed point `X` can be specified in three ways: i) using `k,l,n1,n2,n3,n4,n5,n6`, ii) using `loc of F_i` where `F_i` is the label of a twisted field and iii) by a fixed point label as specified in the directory `/vev-config/labels>`.
- `of sector T(k,l)` where k and l label the $\theta^k \omega^l$ twisted sector. Use `of sector T(0,0)` for the untwisted sector.

In all these cases one can use in addition the following, optional parameters:

- `with labels`: print the currently used labels of the fields as specified in the directory `/vev-config/labels>`.
- `no U1s`: do not print the U(1) charges of the fields.
- type of SUSY multiplet, e.g. `vector`, `gravity`, `modulus` or `anykind` for all types; if not specified the left-chiral fields are printed only.

More details and examples can be seen using the command

```
help print summary
```
(B.17)

in the directory `/spectrum>`.

Command `tex table(fields)`. Print a latex table with information about `fields`. The table contains the (gauge) charges with respect to the observable sector of the currently used vev-configuration and the discrete charges as specified in the geometry file (see Section 4.4.1). One can use the optional parameter `print labels(i,j,..)` in order to list the `i`-th, `j`-th ... label(s) of the fields.

Appendix B.2.5. The directory `/couplings>`

The directory `/couplings>` allows to identify and analyze allowed terms of the superpotential (i.e. terms that are invariant under all gauge and discrete symmetries). The (in general moduli-dependent) coefficients are not computed. Furthermore, one can analyze mass matrices (e.g. of vector-like exotics).

Note that couplings and mass matrices are stored in the currently used vev-configuration. Hence, they can only be accessed in the vev-configuration where they have been defined. For simplicity, the abbreviations `mm` for `mass matrix` and `mms` for `mass matrices` apply to all commands.

Command `auto create mass matrix(A B)`. Create the couplings relevant for the effective mass matrix $M_{ij}A_iB_j$. Optionally, one can specify the the label of those fields whose vevs generate M_{ij} using the parameter `singlet(N)` (with default value `N=n`) and the maximal order `X` in singlets `N` using the parameter `max order(X)`.

Command `create coupling(fields)`. Find the allowed superpotential-couplings between `fields` and store the result in the currently used vev-configuration. For example, all trilinear couplings are created using the command

```
create coupling(* * *) .                                    (B.18)
```

Optionally, one can restrict `fields` using the parameter `allowed fields(...)`, e.g. the command `create coupling(n n n) allowed fields(SetA)` creates all trilinear couplings of fields `n` from `SetA`.

Command `find(fields)`. Displays a list of allowed couplings involving the fields `fields`.

Command `find effective(fields)`. As `find(fields)`, but only the effective couplings, i.e after replacing fields with non-zero vev by their vevs.

Command `load couplings(Filename)`. Load couplings from file `Filename` into the currently used vev-configuration. This command is disabled in the web interface.

Command `mass matrix(A B)`. Create the mass matrix $M_{ij}A_iB_j$ (from the current superpotential) and save it in the currently used vev-configuration.

Command `print effective superpotential`. Similar to the command `print superpotential` but print only the effective couplings, i.e after replacing fields with non-zero vev by their vevs.

Command `print list of mass matrices`. Print all mass matrices of the currently used vev-configuration.

Command `print mass matrix(i)`. Print the `i`-th mass matrix. The optional parameter `max order(X)` specifies the order `X` in the fields up to which a matrix entry shall be printed explicitly.

Command `print superpotential`. Print the superpotential of the currently used vev-configuration.

Command `print vanishing couplings`. Print all cases of vanishing couplings as lists of highest weights in Dynkin labels. See the command `remove vanishing couplings`.

Command `remove vanishing couplings`. Remove couplings that vanish because of symmetry/anti-symmetry of repeated identical fields, e.g. let ℓ be an SU(2) doublet, then the gauge invariant coupling $\ell\ell = \ell_i\ell_j\epsilon^{ij} = 0$ vanishes. This command requires additional user input.

Command `save couplings(Filename)`. Save all couplings of the currently used vev-configuration to a file. One can optionally save only couplings of order `X` using the parameter `of order(X)`. This command is disabled in the web interface.

Appendix B.2.6. The directory `/vev-config>`

In this directory one can define several vev-configurations. Each of them is characterized by a choice of hidden and observable gauge group, a labeling of the fields and by their vev. In addition, one can analyze phenomenological properties and supersymmetric configurations ($F = D = 0$) in this directory and determine the unbroken gauge group of a given vev-configuration.

For each orbifold model there are two standard vev-configurations: `StandardConfig1` and `TestConfig1`. The first one cannot be changed, but the latter one can be and is used as default. In both configurations the full gauge group is selected as the observable sector and fields are labeled `F_1`, `F_2`, `F_3`, ..., all with zero vev.

Note that the labels of the fields (see Appendix B.1.1), sets of fields (see Appendix B.1.2), monomials (see Appendix B.1.3), allowed couplings and mass matrices (created in the directory `/couplings>`) are saved in a vev-configuration. Hence, these data can only be accessed in the vev-configuration where they have been defined. In addition, note that all configuration-editing commands are not available in the vev-configuration `StandardConfig1`.

Command `analyze config`. Automatically check whether the current vev-configuration of the orbifold model allows for vacua with Standard Model, Pati-Salam or SU(5) gauge group,

$$\mathrm{SU}(3)_\mathrm{C} \times \mathrm{SU}(2)_\mathrm{L} \times \mathrm{U}(1)_\mathrm{Y} \ , \quad \mathrm{SU}(4) \times \mathrm{SU}(2)_\mathrm{L} \times \mathrm{SU}(2)_\mathrm{R} \quad \text{or} \quad \mathrm{SU}(5) \ , \tag{B.19}$$

respectively, three generations of quarks and leptons and vector-like exotics. In the case the `orbifolder` is not able to identify one of these possibilities for the current orbifold model one obtains the output `No vev-configuration identified`. Otherwise, corresponding new vacua will be created and convenient labels will be assigned to all matter fields (e.g. `q_1`, `q_2` and `q_3` for the three generations of quark doublets).

The command allows for two optional parameters: `print SU(5) simple roots` to print the simple roots of an intermediate SU(5) group that has been used in order to identify the hypercharge generator and `Xgenerations` with `X`= 0, 1, 2, 3, ... to specify the (net) number of generations.

Command `compute F-terms`. Compute the F-terms using the superpotential that was created for the currently used vev-configuration (in the directory `/couplings>`. The optional parameter `max order(X)` allows to set an upper limit `X` on the order of superpotential couplings.

Command `create config(ConfigLabel)`. Create a new vev-configuration. Optionally, one can specify the origin of the new vev-configuration using the parameter `from(AnotherConfigLabel)`. If this parameter is not used the origin of the new vev-configuration is the standard vev-configuration `StandardConfig1`.

Command `delete config(ConfigLabel)`. Delete the vev-configuration `ConfigLabel`.

Command `find D-flat(fields)`. Identify gauge invariant monomials of `fields` as solutions to the $D = 0$ supersymmetry condition, see Appendix B.1.3. This command allows for two parameters: i) `with FI` to allow for monomials with non-vanishing anomalous U(1) charge in order to cancel the FI term and ii) `save to set(SetLabel)` to save those fields in a set called `SetLabel` that are involved in the new monomials.

Command `find unbroken gauge group`. Depending on the vev assignment specified in the currently used vev-configuration, identify broken and unbroken (Abelian and non-Abelian) gauge group factors. In addition, the U(1) charges of all fields are re-computed in the new U(1) basis. There is one optional parameter: `print info` to display some details.

Command `print gauge group`. Print the choice of observable and hidden sector of the currently used vev-configuration, where the hidden sector gauge group factors are marked by brackets, e.g. `[SU(4)]`.

Command `print configs`. Print an overview of all vacua defined for this orbifold model. The currently used vev-configuration is highlighted by an arrow `->` in front, e.g. `-> "TestConfig1"`.

Command `rename config(OldConfigLabel) to(NewConfigLabel)`. Change the name of a vev-configuration from `OldConfigLabel` to `NewConfigLabel`.

Command `select observable sector: parameters`. Assign a choice of observable and hidden gauge groups in the current vev-configuration. Admissible `parameters` are:

- `gauge group(i,j,...)`, where the indices `i,j` $= 1, 2, \ldots$ refer to the different non-Abelian gauge group factors sorted as displayed by `print gauge group`. The indices provided are chosen as part of the observable sector.
- `full gauge group` All non-Abelian group factors are assigned as observable sector.
- `no gauge groups` No non-Abelian group factor is assigned as part of the observable sector.
- `U1s(i,j,...)`, where the indices `i,j` $= 1, 2, \ldots$ refer to the different U(1) gauge symmetries. The indices provided are chosen as part of the observable sector.
- `all U1s` All U(1)s are assigned as part of the observable sector.
- `no U1s` No U(1) is taken for the observable sector.

For example, assuming that the gauge group is $E_6 \times SU(3) \times E_8$, the instruction `select observable sector: gauge group(1,2)` selects $E_6 \times SU(3)$ as the observable and E_8 as the hidden gauge groups.

Command `use config(ConfigLabel)`. Change the currently used vev-configuration to `ConfigLabel`.

Command `vev(fields) = ...`. Change the vevs of `fields` to new values. For example, `vev(*) = 0` turns off the vev of all fields, `vev(SetA) = rand` assigns random vevs to the fields of the set `SetA` and `vev(n_1) = 0.1` sets the vev of `n_1` to 0.1.

Appendix B.2.7. The directory `/vev-config/labels>`

In this directory one can define, for each vev-configuration, appropriate labels for the fields. The main commands are `print labels` and `create labels`. In both cases, a summary table of massless fields is printed, sorted by those representations and U(1) charges that belong to the observable sector of the currently used vev-configuration. The observable sector can be changed using the command `select observable sector:...` in the directory `/vev-config>`, see Appendix B.2.6.

Command `change label(A_i) to(B_j)`. Change the label of the field `A_i` to `B_j`.

Command `assign label(Label) to fixed point(k,l,n1,n2,n3,n4,n5, n6)`. Assign `Label` to the fixed brane/point specified by `(k,l,n1, n2,n3,n4,n5,n6)`.

Command `create labels`. First, a summary table of massless fields is displayed. Then the user is asked to specify a name for each line of the table.

Command `load labels(Filename)`. Load labels from the file named `Filename`. This command is disabled in the web interface.

Command `print labels`. Print a summary table of the currently used labels displaying the gauge representations with respect to the observable sector only.

Command `save labels(Filename)`. Save the labels to the file named `Filename`. This command is disabled in the web interface.

Command `use label(i)`. Change the currently used labels to the `i`-th labeling.

References

[1] L. J. Dixon, J. A. Harvey, C. Vafa, and E. Witten, "Strings on Orbifolds," *Nucl.Phys.* **B261** (1985) 678–686.

[2] L. J. Dixon, J. A. Harvey, C. Vafa, and E. Witten, "Strings on Orbifolds. 2.," *Nucl.Phys.* **B274** (1986) 285–314.

[3] L. E. Ibáñez, H. P. Nilles, and F. Quevedo, "Orbifolds and Wilson Lines," *Phys. Lett.* **B187** (1987) 25–32.

[4] D. J. Gross, J. A. Harvey, E. J. Martinec, and R. Rohm, "The Heterotic String," *Phys.Rev.Lett.* **54** (1985) 502–505.

[5] D. J. Gross, J. A. Harvey, E. J. Martinec, and R. Rohm, "Heterotic String Theory. 1. The Free Heterotic String," *Nucl.Phys.* **B256** (1985) 253.

[6] L. J. Dixon, D. Friedan, E. J. Martinec, and S. H. Shenker, "The conformal field theory of orbifolds," *Nucl. Phys.* **B282** (1987) 13–73.

[7] S. Hamidi and C. Vafa, "Interactions on orbifolds," *Nucl. Phys.* **B279** (1987) 465.

[8] S. Förste, H. P. Nilles, P. K. Vaudrevange, and A. Wingerter, "Heterotic brane world," *Phys.Rev.* **D70** (2004) 106008, hep-th/0406208.

[9] T. Kobayashi, S. Raby, and R.-J. Zhang, "Searching for realistic 4d string models with a Pati-Salam symmetry: Orbifold grand unified theories from heterotic string compactification on a Z(6) orbifold," *Nucl. Phys.* **B704** (2005) 3–55, hep-ph/0409098.

[10] W. Buchmüller, K. Hamaguchi, O. Lebedev, and M. Ratz, "Supersymmetric standard model from the heterotic string," *Phys. Rev. Lett.* **96** (2006) 121602, hep-ph/0511035.

[11] K.-S. Choi and J. E. Kim, "Quarks and leptons from orbifolded superstring,". Lecture Notes on Physics 696, Berlin-Heidelberg, Germany: Springer (2006) 406 p.

[12] H. P. Nilles, S. Ramos-Sánchez, M. Ratz, and P. K. Vaudrevange, "From strings to the MSSM," *Eur.Phys.J.* **C59** (2009) 249–267, 0806.3905. To be published in 'Supersymmetry on the Eve of the LHC', a special volume of EPJC dedicated to the memory of Julius Wess.

[13] P. K. Vaudrevange, "Grand Unification in the Heterotic Brane World," 0812.3503.

[14] S. Ramos-Sánchez, "Towards Low Energy Physics from the Heterotic String," *Fortsch. Phys.* **10** (2009) 907–1036, 0812.3560.

[15] F. Quevedo, "Lectures on superstring phenomenology," hep-th/9603074.

[16] D. Bailin and A. Love, "Orbifold compactifications of string theory," *Phys. Rept.* **315** (1999) 285–408.

[17] O. Lebedev, H. P. Nilles, S. Raby, S. Ramos-Sánchez, M. Ratz, P. K. S. Vaudrevange, and A. Wingerter, "A Mini-landscape of exact MSSM spectra in heterotic orbifolds," *Phys.Lett.* **B645** (2007) 88–94, hep-th/0611095.

[18] O. Lebedev, H. P. Nilles, S. Raby, S. Ramos-Sánchez, M. Ratz, *et al.*, "The Heterotic Road to the MSSM with R parity," *Phys.Rev.* **D77** (2008) 046013, 0708.2691.

[19] O. Lebedev, H. P. Nilles, S. Ramos-Sánchez, M. Ratz, and P. K. Vaudrevange, "Heterotic mini-landscape (II). Completing the search for MSSM vacua in a $\mathbb{Z}_6$ orbifold," *Phys.Lett.* **B668** (2008) 331–335, 0807.4384.

[20] O. Lebedev, H.-P. Nilles, S. Raby, S. Ramos-Sánchez, M. Ratz, *et al.*, "Low Energy Supersymmetry from the Heterotic Landscape," *Phys.Rev.Lett.* **98** (2007) 181602, hep-th/0611203.
[21] J. E. Kim, J.-H. Kim, and B. Kyae, "Superstring standard model from Z(12-I) orbifold compactification with and without exotics, and effective R-parity," *JHEP* **0706** (2007) 034, hep-ph/0702278.
[22] W. Buchmüller, K. Hamaguchi, O. Lebedev, S. Ramos-Sánchez, and M. Ratz, "Seesaw neutrinos from the heterotic string," *Phys. Rev. Lett.* **99** (2007) 021601, hep-ph/0703078.
[23] B. Dundee, S. Raby, and A. Wingerter, "Reconciling Grand Unification with Strings by Anisotropic Compactifications," *Phys. Rev.* **D78** (2008) 066006, 0805.4186.
[24] R. Kappl *et al.*, "Large hierarchies from approximate R symmetries," *Phys. Rev. Lett.* **102** (2009) 121602, 0812.2120.
[25] S. G. Nibbelink, J. Held, F. Ruehle, M. Trapletti, and P. K. S. Vaudrevange, "Heterotic Z6-II MSSM Orbifolds in Blowup," *JHEP* **03** (2009) 005, 0901.3059.
[26] P. Hosteins, R. Kappl, M. Ratz, and K. Schmidt-Hoberg, "Gauge-top unification," *JHEP* **07** (2009) 029, 0905.3323.
[27] K.-S. Choi, H. P. Nilles, S. Ramos-Sánchez, and P. K. S. Vaudrevange, "Accions," *Phys.Lett.* **B675** (2009) 381–386, 0902.3070.
[28] M. Blaszczyk *et al.*, "A Z2xZ2 standard model," *Phys. Lett.* **B683** (2010) 340–348, 0911.4905.
[29] O. Lebedev and S. Ramos-Sánchez, "The NMSSM and String Theory," *Phys. Lett.* **B684** (2010) 48–51, 0912.0477.
[30] M. Blaszczyk, S. G. Nibbelink, F. Ruehle, M. Trapletti, and P. K. S. Vaudrevange, "Heterotic MSSM on a Resolved Orbifold," *JHEP* **09** (2010) 065, 1007.0203.
[31] S. Förste, H. P. Nilles, S. Ramos-Sánchez, and P. K. S. Vaudrevange, "Proton Hexality in Local Grand Unification," *Phys.Lett.* **B693** (2010) 386–392, 1007.3915.
[32] S. L. Parameswaran, S. Ramos-Sánchez, and I. Zavala, "On Moduli Stabilisation and de Sitter Vacua in MSSM Heterotic Orbifolds," *JHEP* **01** (2011) 071, 1009.3931.
[33] R. Kappl *et al.*, "String-derived MSSM vacua with residual R symmetries," *Nucl. Phys.* **B847** (2011) 325–349, 1012.4574.
[34] R. Donagi and K. Wendland, "On orbifolds and free fermion constructions," *J.Geom.Phys.* **59** (2009) 942–968, 0809.0330.
[35] C. Vafa, "Modular Invariance and Discrete Torsion on Orbifolds," *Nucl.Phys.* **B273** (1986) 592.
[36] F. Plöger, S. Ramos-Sánchez, M. Ratz, and P. K. Vaudrevange, "Mirage Torsion," *JHEP* **0704** (2007) 063, hep-th/0702176.
[37] "Boost C++ libraries." http://www.boost.org/, 2011.
[38] M. Galassi et al., *GNU Scientific Library Reference Manual - Third Edition (v1.12)*, 2009. http://www.gnu.org/software/gsl/.
[39] P. Nilles, S. Ramos-Sánchez, P. K. Vaudrevange, and A. Wingerter, "The orbifolder web page," 2011. http://projects.hepforge.org/orbifolder/.
[40] W. Buchmüller, K. Hamaguchi, O. Lebedev, and M. Ratz, "Supersymmetric Standard Model from the Heterotic String (II)," *Nucl.Phys.* **B785** (2007) 149–209, hep-th/0606187.
[41] A. E. Faraggi, D. V. Nanopoulos, and K.-j. Yuan, "A Standard Like Model in the 4D Free Fermionic String Formulation," *Nucl. Phys.* **B335** (1990) 347.
[42] A. E. Faraggi, "Construction of realistic standard - like models in the free fermionic superstring formulation," *Nucl. Phys.* **B387** (1992) 239–262, hep-th/9208024.
[43] B. Assel, K. Christodoulides, A. E. Faraggi, C. Kounnas, and J. Rizos, "Classification of Heterotic Pati-Salam Models," *Nucl. Phys.* **B844** (2011) 365–396, 1007.2268.
[44] T. P. T. Dijkstra, L. R. Huiszoon, and A. N. Schellekens, "Supersymmetric standard model spectra from RCFT orientifolds," *Nucl. Phys.* **B710** (2005) 3–57, hep-th/0411129.
[45] P. Candelas, G. T. Horowitz, A. Strominger, and E. Witten, "Vacuum Configurations for Superstrings," *Nucl. Phys.* **B258** (1985) 46–74.
[46] R. Donagi, Y.-H. He, B. A. Ovrut, and R. Reinbacher, "The spectra of heterotic standard model vacua," *JHEP* **06** (2005) 070, hep-th/0411156.
[47] V. Braun, Y.-H. He, B. A. Ovrut, and T. Pantev, "A heterotic standard model," *Phys. Lett.* **B618** (2005) 252–258, hep-th/0501070.
[48] V. Bouchard and R. Donagi, "An SU(5) heterotic standard model," *Phys. Lett.* **B633** (2006) 783–791, hep-th/0512149.
[49] L. B. Anderson, J. Gray, A. Lukas, and B. Ovrut, "Stabilizing All Geometric Moduli in Heterotic Calabi-Yau Vacua," *Phys. Rev.* **D83** (2011) 106011, 1102.0011.
[50] L. B. Anderson, J. Gray, A. Lukas, and E. Palti, "Two Hundred Heterotic Standard Models on Smooth Calabi-Yau Threefolds," 1106.4804.
[51] R. Blumenhagen, M. Cvetič, P. Langacker, and G. Shiu, "Toward realistic intersecting D-brane models," *Ann. Rev. Nucl. Part. Sci.* **55** (2005) 71–139, hep-th/0502005.
[52] F. Gmeiner, R. Blumenhagen, G. Honecker, D. Lüst, and T. Weigand, "One in a billion: MSSM-like D-brane statistics,"

JHEP **01** (2006) 004, hep-th/0510170.
[53] R. Blumenhagen, B. Kors, D. Lüst, and S. Stieberger, "Four-dimensional String Compactifications with D-Branes, Orientifolds and Fluxes," *Phys. Rept.* **445** (2007) 1–193, hep-th/0610327.
[54] M. R. Douglas and W. Taylor, "The landscape of intersecting brane models," *JHEP* **01** (2007) 031, hep-th/0606109.
[55] F. Gmeiner and G. Honecker, "Millions of Standard Models on Z6-prime?," *JHEP* **07** (2008) 052, 0806.3039.
[56] C. Vafa, "Evidence for F-Theory," *Nucl. Phys.* **B469** (1996) 403–418, hep-th/9602022.
[57] R. Donagi, A. Lukas, B. A. Ovrut, and D. Waldram, "Non-perturbative vacua and particle physics in M-theory," *JHEP* **05** (1999) 018, hep-th/9811168.
[58] R. Donagi, B. A. Ovrut, T. Pantev, and D. Waldram, "Standard models from heterotic M-theory," *Adv. Theor. Math. Phys.* **5** (2002) 93–137, hep-th/9912208.
[59] C. Beasley, J. J. Heckman, and C. Vafa, "GUTs and Exceptional Branes in F-theory - I," *JHEP* **01** (2009) 058, 0802.3391.
[60] C. Beasley, J. J. Heckman, and C. Vafa, "GUTs and Exceptional Branes in F-theory - II: Experimental Predictions," *JHEP* **01** (2009) 059, 0806.0102.
[61] B. S. Acharya, K. Bobkov, G. L. Kane, J. Shao, and P. Kumar, "The G_2-MSSM - An *M* Theory motivated model of Particle Physics," *Phys. Rev.* **D78** (2008) 065038, 0801.0478.
[62] T. Weigand, "Lectures on F-theory compactifications and model building," *Class. Quant. Grav.* **27** (2010) 214004, 1009.3497.
[63] P. S. Aspinwall, "Resolution of orbifold singularities in string theory," hep-th/9403123.
[64] D. Lüst, S. Reffert, E. Scheidegger, and S. Stieberger, "Resolved toroidal orbifolds and their orientifolds," hep-th/0609014.
[65] S. G. Nibbelink, T.-W. Ha, and M. Trapletti, "Toric Resolutions of Heterotic Orbifolds," *Phys.Rev.* **D77** (2008) 026002, 0707.1597.
[66] M. Blaszczyk, N. G. C. Bizet, H. P. Nilles, and F. Ruehle, "A perfect match of MSSM-like orbifold and resolution models via anomalies," 1108.0667.
[67] F. Buccella, J. P. Derendinger, S. Ferrara, and C. A. Savoy, "Patterns of symmetry breakıng in supersymmetric gauge theories," *Phys. Lett.* **B115** (1982) 375.
[68] G. Cleaver, M. Cvetič, J. R. Espinosa, L. L. Everett, and P. Langacker, "Classification of flat directions in perturbative heterotic superstring vacua with anomalous U(1)," *Nucl. Phys.* **B525** (1998) 3–26, hep-th/9711178.
[69] R. Kappl, M. Ratz, and C. Staudt, "The Hilbert basis method for D-flat directions and the superpotential," 1108.2154.
[70] C. Staudt, "NonAbelianHilbert – a software package for constructing gauge invariant monomials in Mathematica,". http://einrichtungen.ph.tum.de/T30e/codes/NonAbelianHilbert/.
[71] J. Gray, Y.-H. He, A. Ilderton, and A. Lukas, "STRINGVACUA: A Mathematica Package for Studying Vacuum Configurations in String Phenomenology," *Comput.Phys.Commun.* **180** (2009) 107–119, 0801.1508.
[72] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 3-1-3 — A computer algebra system for polynomial computations,". http://www.singular.uni-kl.de.
[73] B. Petersen, M. Ratz, and R. Schieren, "Patterns of remnant discrete symmetries," *JHEP* **0908** (2009) 111, 0907.4049.
[74] R. Schieren, "DiscreteBreaking – a Mathematica package for identifying and simplifying discrete symmetries,". http://einrichtungen.physik.tu-muenchen.de/T30e/codes/DiscreteBreaking/.
[75] W. Buchmüller, C. Lüdeling, and J. Schmidt, "Local SU(5) Unification from the Heterotic String," *JHEP* **0709** (2007) 113, 0707.1651.
[76] T. Araki, T. Kobayashi, J. Kubo, S. Ramos-Sánchez, M. Ratz, and P. K. S. Vaudrevange, "(Non-)Abelian discrete anomalies," *Nucl.Phys.* **B805** (2008) 124–147, 0805.0207.