Fuzzing with  Fandango

Tutorial and Reference

José Antonio Zamudio Amaya
Valentin Huber
Alexander Liggesmeyer
Marius Smytzek
Andreas Zeller

Find Fandango at
https://fandango-fuzzer.github.io/

CONTENTS

Fuzzing with Fandango

Fandango produces myriads of *high-quality random inputs and interactions* to test programs and protocols, giving users unprecedented control over format and shape of the inputs.

Get Started (page 19)

At a Glance Specify the format of your input data in a single file, combining grammars (page 25) (for input syntax) and constraints (page 45) (for arbitrary input features). Constraints come as *Python code*, so there are no limits to what you can specify.

About Fandango » (page 5)

Tutorial Produce valid inputs at high speeds, quickly covering the entire input space. Test with extreme *and* uncommon *values* (page 37), uncovering bugs before your users do. Tie in *Python data generators and fakers* (page 67) to obtain realistic inputs.

Tutorial » (page 19)

Reference Check program *outputs* (page 165) for correctness. Test and mock clients and servers using *protocol testing* (page 233). Create and check *binary* strings, using *bit fields and bit sequences* (page 129). Use *regular expressions* (page 75) for quick and easy specifications.

Reference » (page 293)

Fandango is a project of the CISPA Helmholtz Center for Information Security[1] to facilitate highly efficient and highly customizable software testing.

This research was funded by the European Union (ERC "Semantics of Software Systems", S3[2], 101093186). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

[1] https://www.cispa.de/
[2] https://www.cispa.de/s3

Part I

About Fandango

CHAPTER

ONE

ABOUT FANDANGO

Given the specification of a program's input language, Fandango quickly generates myriads of valid sample inputs for testing.

The specification language combines a *grammar* with *constraints* written in Python, so it is extremely expressive and flexible. Most notably, you can define your own *testing goals* in Fandango. If you need the inputs to have particular values or distributions, you can express all these right away in Fandango.

Fandango supports multiple modes of operation:

- By default, Fandango operates as a *black-box* fuzzer - that is, it creates inputs from a `.fan` Fandango specification file.
- If you have *sample inputs*, Fandango can *mutate* these to obtain more realistic inputs.
- Fandango can also produce *interactions* for *protocol* fuzzing - that is, it acts as a client or server producing and reacting to interactions according to specification.

Fandango comes as a portable Python program and can easily be run on a large variety of platforms.

Under the hood, Fandango uses sophisticated *evolutionary algorithms* to produce inputs, it starts with a population of random inputs, and evolves these through mutations and cross-over until they fulfill the given constraints.

Fandango is in active development! Features planned for 2026 include:

- coverage-guided testing
- code-directed testing
- high diversity inputs

and many more.

1.1 The Fandango Paper

To learn more about how Fandango works, start with the ISSTA 2025 paper "FANDANGO: Evolving Language-Based Testing"[3] by Zamudio Amaya, Smytzek, and Zeller [ZASZ25]:

To cite this paper, use this BibTeX reference:

```
@article{zamudio2025fandango,
author = {Zamudio Amaya, Jos\'{e} Antonio and Smytzek, Marius and Zeller, Andreas},
title = {{FANDANGO}: {E}volving Language-Based Testing},
year = {2025},
issue_date = {July 2025},
```

(continues on next page)

[3] https://doi.org/10.1145/3728915

(continued from previous page)

```
publisher = {Association for Computing Machinery},
address = {New York, NY, USA},
volume = {2},
number = {ISSTA},
url = {https://doi.org/10.1145/3728915},
doi = {10.1145/3728915},
abstract = {Language-based fuzzers leverage formal input specifications␣
↪(languages) to generate arbitrarily large and diverse sets of valid inputs for a␣
↪program under test. Modern language-based test generators combine grammars and␣
↪constraints to satisfy syntactic and semantic input constraints. ISLA, the␣
↪leading input generator in that space, uses symbolic constraint solving to solve␣
↪input constraints. Using solvers places ISLA among the most precise fuzzers but␣
↪also makes it slow.    In this paper, we explore search-based testing as an␣
↪alternative to symbolic constraint solving. We employ a genetic algorithm that␣
↪iteratively generates candidate inputs from an input specification, evaluates␣
↪them against defined constraints, evolving a population of inputs through␣
↪syntactically valid mutations and retaining those with superior fitness until␣
↪the semantic input constraints are met. This evolutionary procedure, analogous␣
↪to natural genetic evolution, leads to progressively improved inputs that cover␣
↪both semantics and syntax. This change boosts the efficiency of language-based␣
↪testing: In our experiments, compared to ISLA, our search-based FANDANGO␣
↪prototype is faster by one to three orders of magnitude without sacrificing␣
↪precision.    The search-based approach no longer restricts constraints to␣
↪constraint solvers' (miniature) languages. In FANDANGO, constraints can use the␣
↪whole Python language and library. This expressiveness gives testers␣
↪unprecedented flexibility in shaping test inputs. It allows them to state␣
↪arbitrary goals for test generation: ''Please produce 1,000 valid test inputs␣
↪where the voltage field follows a Gaussian distribution but never exceeds 20 mV.'
↪'},
journal = {Proc. ACM Softw. Eng.},
month = jun,
articleno = {ISSTA040},
numpages = {23},
keywords = {Language-based testing, fuzzing, test generation}
}
```

Note that the first author's first name is "José Antonio", and his last name is "Zamudio Amaya".

1.2 Link to Fandango

To link to Fandango, use its official URL:

https://fandango-fuzzer.github.io

1.3 Read More

The core idea of Fandango, namely combining grammars and constraints, was introduced as *language-based software testing* by Steinhöfel and Zeller [SteinhofelZ24] and first implemented in the *ISLa* framework [SteinhofelZ22]. Both of these laid the foundation for Fandango.

The work on Fandango is funded by the ERC S3 project "Semantics of Software Systems"[4]; the S3 grant proposal (available via the above link) lists several ideas that have been realized in Fandango (and a few more).

The work on Fandango is also related to *mining grammars* from programs and inputs. Important works in the field include Bettscheider and Zeller [BZ24], Gopinath, Mathis, and Zeller [GMZ20], Schröder and Cito [SchroderC22], and Kulkarni, Lemieux, and Sen [KLS22].

1.4 Acknowledgments

Fandango is a project of the CISPA Helmholtz Center for Information Security[13] to facilitate highly efficient and highly customizable software testing.

This research was funded by the European Union (ERC "Semantics of Software Systems", S3[14], 101093186). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

[4] https://cispa.de/s3
[13] https://www.cispa.de/
[14] https://www.cispa.de/s3

CHAPTER

TWO

RELEASE NOTES

This document lists major changes across releases.

Added in version 1.1: (January 2026)

- Much enhanced *protocol fuzzing* (page 233):
  - `fandango talk` now systematically covers states and messages
  - `fandango talk` now keeps on producing diverse interactions until stopped or 100% coverage is reached
  - Much extended documentation with *FTP* (page 259) and *DNS* (page 277) case studies
  - Protocol fuzzing is now out of beta
- [development] Major internal refactorings and code quality improvements.
- [development] Added support for the `uv` package manager, including appropriate lock files to ensure compatible and fixed dependency versions

Added in version 1.0: (June 2025)

- New command `fandango talk` for *checking outputs* (page 165) and *testing protocols* (page 233) (beta).
- Much faster parser for `.fan` files, using C++ code.
- Support for *regular expressions* (page 75) for producing and parsing.
- Support for *soft constraints and optimization* (page 50) (`maximizing` / `minimizing`).
- Using `*`/`**` expressions for Python-style *quantifiers* (page 98) is now operational.
- f-strings in Python code are now supported.
- Support for *`libfuzzer` harnesses* (page 34).
- New command `fandango convert` for *converting ANTLR and other input specifications* (page 103).
- New command `fandango clear-cache` for *clearing the parser cache* (page 341).
- Improved bit parser.
- Lots of minor and major improvements and bug fixes across the board.

Changed in version 1.0:

- We now apply Python *end-of-line rules to grammar parsing* (page 305). End continuation lines with `\` or use parentheses.
- Fuzzing by default is now "infinite", producing results until stopped. Specify `-n N` to obtain `N` outputs.

Added in version 0.8: (April 2025)

- First public beta release.

- `fandango fuzz` and `fandango parse` commands.

CHAPTER

THREE

FANDANGO FAQ

This document holds frequently asked questions about Fandango.

Where does the name Fandango come from?

Fandango is an Iberian dance[15]. We chose the name because of the lively nature of the dance. Also, in the Hacker file[16] (also known as The New Hacker's Dictionary), the term Fandango on Core[17] is listed as a term for corrupted main (core) memory, so it is also related to bugs.

Where would one get Fandango grammars and constraints from?

Besides writing them manually, one can use *converters* (page 103) to generate them from other formal specifications such as ANTLR. Recent research has shown various methods to extract grammars from existing programs ([BZ24], [GMZ20], [SchroderC22], [KLS22]) In a few years, we expect that LLMs will be able to help with writing input specifications just like they help in synthesizing code - and then, Fandango will be there to leverage your specs.

Why not use Python (or any program) to generate outputs in the first place?

Regular programs either *parse* or *produce* inputs. Fandango specifications (including constraints) allow for both in a single, concise representation. Since Fandango can use its specs to parse *and* produce inputs, it can easily mutate existing inputs without any implementation effort. Finally, you do not have to deal with implementing an appropriate algorithm to achieve goals such as constraints or input diversity; Fandango does all of this for you.

What other alternatives are there to specify input languages?

Apart from program code (see above), not many. We believe that Fandango is the only approach to allow a full specification of inputs that can be used for both *parsing* and *producing*; indeed, it can specify inputs from the bit layer up to the application layer, including how these layers translate into each other.

What's the difference to coverage-guided fuzzing?

A specification-based fuzzer such as Fandango is a *blackbox* fuzzer. It does not require feedback (such as coverage) from the program to be tested, nor does it require sample inputs. On the other hand, the constraints used by Fandango do not preclude coverage guidance. Stay tuned for future extensions.

How well does Fandango scale?

Programs are the most complex inputs known to mankind; and Fandango very likely will not be able to achieve both 100% precision and 100% diversity, (Nor does any other method on this planet, by the way.) But any file format whose complexity is lower (or if you can live with much lower precision or diversity) should work with Fandango. Please share your experiences!

Who made the Fandango teaser video?

The Fandango teaser video[18] was produced by the Public Relations team at CISPA, based on a script by Andreas Zeller and starring José Antonio Zamudio Amaya. Andreas also did the narration in Attenborough style; no AI or other post-processing was involved.

[15] https://en.wikipedia.org/wiki/Fandango
[16] http://www.catb.org/jargon/
[17] http://www.catb.org/jargon/html/F/fandango-on-core.html
[18] https://www.youtube.com/watch?v=JXMk-XhuKPY

CHAPTER

FOUR

CONTRIBUTING TO FANDANGO

Welcome! Fandango is a community project that aims to work for a wide range of developers. If you're trying out Fandango, your experience and what you can contribute are important to the project's success.

4.1 Code of Conduct

Everyone participating in Fandango, and in particular in our issue tracker, pull requests, and chat, is expected to treat other people with respect and more generally to follow the guidelines articulated in the Python Community Code of Conduct[19].

4.2 Getting Started with Development

4.2.1 Step 1: Fork the Fandango repository

Within GitHub, navigate to the Fandango GitHub repository[20] and fork the repository.

4.2.2 Step 2: Clone the Fandango repository and enter into it

```
$ git clone https://github.com/fandango-fuzzer/fandango.git
```

```
$ cd fandango
```

4.2.3 Step 3: Create, then activate a virtual environment

```
$ python3 -m venv .venv
```

```
$ source .venv/bin/activate
```

For Windows, use

```
$ python -m venv .venv
```

[19] https://www.python.org/psf/codeofconduct/
[20] https://github.com/fandango-fuzzer/fandango

```
$ . venv/Scripts/activate
```

For more details, see https://docs.python.org/3/library/venv.html#creating-virtual-environments

4.2.4 Step 4: Install development tools

Install `antlr` and other system-level tools:

```
$ make system-dev-tools
```

4.2.5 Step 5: Install Fandango

ti Install your local copy of Fandango:

```
$ python -m pip install -e .
```

To install additional dependencies for unit tests, code checks, running the evaluation, and building the Fandango book, install Fandango like this (or with the subsection of dependencies you need):

```
$ python -m pip install -e ".[test,development,evaluation,book]"
```

> **Tip**
>
> If subsequent tests fails due to dependency breakage, use `uv` to create the virtual environment with `uv.lock`, which provides a snapshot of the dependencies that are tested on our CI:
>
> ```
> $ uv sync --locked --all-extras
> ```

If you don't need the (much faster) C++ parser, build Fandango without it:

```
$ FANDANGO_SKIP_CPP_PARSER=1 python -m pip install -e .
```

Reset the shell PATH cache (not necessary on Windows):

```
$ hash -r
```

That's it! You just have installed your personal copy of Fandango. You can invoke it as `fandango` and can alter its code as you like.

4.3 Running Tests

Running the full test suite can take a while, and usually is not necessary when preparing a pull request. Once you file a pull request, the full test suite will run on GitHub. You'll then be able to see any test failures, and make any necessary changes to your pull request.

However, if you wish to do so, you can run the full test suite like this:

```
$ make tests
```

or simply

```
$ pytest
```

To run tests in parallel, you can use pytest-xdist[21] (installed with the `test` dependencies):

```
$ pytest -n auto
```

4.3.1 Pre-commit

Fandango provides a config file for `pre-commit`. To install it, run the following:

```
$ pip install pre-commit
$ pre-commit install
```

This will run code formatting and type checking before every commit.

> **Warning**
>
> The *tests* (page 14) are not run as part of pre-commit, because they take significant time to complete. Run them manually.

`uv.lock` is a snapshot of the (actual) dependencies used in this project. After updating the dependencies in `pyproject.toml`, make sure `uv.lock` is up-to-date with `uv lock`.

4.4 First Time Contributors

If you're looking for things to help with, browse our issue tracker[22]!

You do not need to ask for permission to work on any of these issues. Just fix the issue yourself, *try to add a unit test* (page 14) and open a pull request.

To get help fixing a specific issue, it's often best to comment on the issue itself. You're much more likely to get help if you provide details about what you've tried and where you've looked (maintainers tend to help those who help themselves).

4.5 Contributing Code

Even more excellent than a good bug report is a fix for a bug, or the implementation of a much-needed new feature. We'd love to have your contributions.

We use the usual GitHub pull-request flow, which may be familiar to you if you've contributed to other projects on GitHub. For the mechanics, see GitHub's own documentation[23].

Anyone interested in Fandango may review your code. One of the Fandango core developers will merge your pull request when they think it's ready.

If your change will be a significant amount of work to write, we highly recommend starting by opening an issue laying out what you want to do. That lets a conversation happen early in case other contributors disagree with what you'd like to do or have ideas that will help you do it.

[21] https://pytest-xdist.readthedocs.io/en/stable/
[22] https://github.com/fandango-fuzzer/fandango/issues
[23] https://help.github.com/articles/using-pull-requests/

The best pull requests are focused, clearly describe what they're for and why they're correct, and contain tests for whatever changes they make to the code's behavior. As a bonus these are easiest for someone to review, which helps your pull request get merged quickly! Standard advice about good pull requests for open-source projects applies.

4.5.1 Changing Parser Files

If your contribution involves changing the ANTLR `.g4` parser files, you need to *regenerate* and *recompile* the parser code. For this, you need a C++ compiler such as `clang`.

Step 1: Generate the parser code

To (re)generate the parser code, use

```
$ make parser
```

Step 2: Compile the parser code

The `pip` command takes care of (re)compiling the parser code. Simply *reinstall Fandango* (page 14).

> **Note**
>
> If compiling the C++ code for parsing fails, Fandango uses (slower) Python code instead.

4.5.2 Contributing to Documentation

If you want to contribute to this very tutorial and reference, be sure to preview your changes. Use

```
$ make html
```

to create an HTML version of the documentation in `docs/_build/html`.

4.6 Attributions

This guide was forked from the mypy contribution guide[24], which is licensed under the terms of the MIT license.

[24] https://github.com/python/mypy/blob/master/CONTRIBUTING.md

Part II

First Steps

CHAPTER

FIVE

TUTORIAL CONVENTIONS

5.1 Conventions in this Documentation

5.1.1 Shell Commands

Fandango is mostly operated via the *command line*. A command `fandango help` is shown as follows:

```
$ fandango help
```

Note

In this tutorial, a `$` at the beginning of a command stands for your input prompt. Do not enter it yourself.

Tip

Hovering the mouse pointer over the command offers an option to copy the command to the clipboard.

5.1.2 Code

File contents (mostly Fandango `.fan` files) are shown like this:

```
<start> ::= <fandango>
```

We often only show excerpts; the complete examples can be downloaded, as in digits.fan.

Executable code examples (mostly in Python) are shown like this:

```
from fandango import Fandango
```

Enter these at a Python prompt, or include them in your Python programs.

Tip

Hovering the mouse pointer over the code offers an option to copy the code to the clipboard.

5.1.3 Callouts

This documentation uses the following conventions for *callouts*:

There are also *margin notes*, which show interesting background information.

Tip

Show helpful information or another way to do something.

Note

Bring additional information to the readers' attention.

Important

Bring important information to the readers' attention.

Under Construction

Indicate that some parts of the documentation or the implementation are under construction.

Caution

Tell the reader to proceed carefully.

Warning

Warn about possible irreversible damage, such as permanent data loss.

Pretty much all of these occur in the chapter on *Fuzzing Strategies* (page 37).

Danger

Warn about hazards that may lead to death or serious injury.

5.1.4 Quizzes

This documentation also uses *quizzes* – that is, questions for the reader. The solutions are initially hidden, but can be unhidden by clicking on them.

For instance, what does a *Warning* callout indicate?

Solution

It indicates a step that might cause irreversible damage, such as permanent data loss.

5.1.5 Changes

These hints highlight changes across versions:

Added in version 1.0: Explanation of the new feature.

Changed in version 1.0: Explanation of the change.

Deprecated since version 1.0: Explanation of the deprecation.

CHAPTER

SIX

INSTALLING FANDANGO

To install Fandango, you first need to install Python[25]. Then, run the following command:

```
$ pip install fandango-fuzzer
```

Note

In this tutorial, a `$` at the beginning of a command stands for your input prompt. Do not enter it yourself.

You can check if everything works well by running

```
$ fandango --help
```

Entering `fandango --help` should result in an output like this:

```
usage: fandango [-h] [--version] [--verbose | --quiet]
                [--parser {python,cpp,legacy,auto}]
                {fuzz,parse,talk,convert,clear-cache,shell,help,copyright,version}␣
↪...

The access point to the Fandango framework

options:
  -h, --help            show this help message and exit
  --version             Show version number.
  --verbose, -v         Increase verbosity. Can be given multiple times (-vv).
  --quiet, -q           Decrease verbosity. Can be given multiple times (-qq).
  --parser {python,cpp,legacy,auto}
                        Parser implementation to use (default: 'auto': use C++
                        parser code if available, otherwise Python).

commands:
  {fuzz,parse,talk,convert,clear-cache,shell,help,copyright,version}
                        The command to execute.
    fuzz                Produce outputs from .fan files and test programs.
    parse               Parse input file(s) according to .fan spec.
    talk                Interact with programs, clients, and servers.
    convert             Convert given external spec to .fan format.
    clear-cache         Clear the Fandango parsing cache.
    shell               Run an interactive shell (default).
    help                Show this help and exit.
```

(continues on next page)

[25] https://www.python.org/

(continued from previous page)

```
    copyright           Show copyright.
    version             Show version.

Use `fandango help` to get a list of commands.
Use `fandango help COMMAND` to learn more about COMMAND.
See https://fandango-fuzzer.github.io/ for more information.
```

If this did not work, try out an alternate option; see *Installing Fandango* (page 295).

CHAPTER

SEVEN

A FIRST FANDANGO SPEC

To create test inputs, Fandango needs a *Fandango spec* – a file that describes how the input should be structured.

A Fandango specification contains three types of elements:

- A *grammar* describing the *syntax* of the inputs to be generated;
- Optionally, *constraints* that specify additional properties; and
- Optionally, *definitions* of functions and constants within these constraints.

Only the first of these (the *grammar*) is actually required.

> For writing extended Fandango Specs, check out our *Fandango language server and an IDE extension* (page 343).

Here is a very simple Fandango grammar that will get us started:

```
<start> ::= <digit>+
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

This grammar defines a *sequence of digits*:

- The first line in our grammar defines `<start>` – this is the input to be generated.
- What is `<start>`? This comes on the right-hand side of the "define" operator (`::=`).
- We see that `<start>` is defined as `<digit>+`, which means a non-empty sequence of `<digit>` symbols.

> In Fandango grammars, you can append these operators to a symbol:
>
> - `+` indicates *one or more* repetitions;
> - `*` indicates *zero or more* repetitions; and
> - `?` indicates that the symbol is *optional*.

- What is a `<digit>`? This is defined in the next line: one of ten alternative strings, separated by `|`, each representing a single digit.

To produce inputs from the grammar, Fandango

- starts with the start symbol (`<start>`)

- repeatedly replaces symbols (in `<...>`) by *one of their definitions* (= one of the alternatives separated by `|`) on the right-hand side;
- until no symbols are left.

So,

- `<start>` first becomes `<digit><digit><digit>...` (any number of digits, but at least one);
- each `<digit>` becomes a digit from zero to nine;
- and in the end, we get a string such as `8347`, `66`, `2`, or others.

Let us try generating inputs with this right away by invoking Fandango (page 27).

CHAPTER

EIGHT

INVOKING FANDANGO

8.1 Running Fandango on our Example Spec

To run Fandango on *our "digits" example* (page 25), create or download a file digits.fan with the "digits" grammar in the current folder.

Fandango specs have a `.fan` extension.

Then, you can run Fandango on it to create inputs. The command we need is called `fandango fuzz`, and it takes two important parameters:

- `-f` followed by the `.fan` file – for us, that is `digits.fan`;
- `-n` followed by the number of inputs we want to have – say, 10.

This is how we invoke Fandango:

```
$ fandango fuzz -f digits.fan -n 10
```

And this is what we get:

```
9528583126282757265
4507045774332
671089728108
504478448900
189096
45
1217144886715
590493224717393
01399616320777822
97814
```

Success! We have created 10 random sequences of digits.

Warning

Be aware that `.fan` files can contain Python code that is *executed when being loaded*. This code can execute arbitrary commands.

Caution

Only load `.fan` files you trust.

8.2 More Commands and Options

Besides `fandango fuzz`, Fandango supports more commands, and besides `-f` and `-n`, Fandango supports way more options. To find out which commands Fandango supports, try

```
$ fandango --help
```

Important

Fandango commands not detailed in this documentation are *experimental* – do not rely on them.

To find out which option a particular command supports, invoke the command with `--help`. For instance, these are all the options supported by `fandango fuzz`:

```
$ fandango fuzz --help
```

```
usage: fandango fuzz [-h] [-f FAN_FILE] [-c CONSTRAINT] [-S START_SYMBOL]
                     [--max MAXCONSTRAINT] [--min MINCONSTRAINTS] [-I DIR]
                     [--file-mode {text,binary,auto}] [--no-cache]
                     [--no-stdlib] [-s SEPARATOR] [-d DIRECTORY]
                     [-x FILENAME_EXTENSION]
                     [--format {string,bits,tree,grammar,value,repr,none}]
                     [--validate] [-N MAX_GENERATIONS] [--infinite]
                     [--population-size POPULATION_SIZE]
                     [--elitism-rate ELITISM_RATE]
                     [--crossover-rate CROSSOVER_RATE]
                     [--mutation-rate MUTATION_RATE]
                     [--random-seed RANDOM_SEED]
                     [--destruction-rate DESTRUCTION_RATE]
                     [--max-repetition-rate MAX_REPETITION_RATE]
                     [--max-repetitions MAX_REPETITIONS]
                     [--max-node-rate MAX_NODE_RATE] [--max-nodes MAX_NODES]
                     [-n DESIRED_SOLUTIONS] [--best-effort]
                     [-i INITIAL_POPULATION] [--progress-bar {on,off,auto}]
                     [--warnings-are-errors] [--party PARTY] [-o OUTPUT]
                     [--input-method {stdin,filename,libfuzzer}]
                     [command] ...

options:
  -h, --help            show this help message and exit
  -o, --output OUTPUT   Write output to OUTPUT (default: stdout).

Fandango file settings:
  -f, --fandango-file FAN_FILE
                        Fandango file (.fan, .py) to be processed. Can be
                        given multiple times. Use '-' for stdin.
  -c, --constraint CONSTRAINT
                        Define an additional constraint CONSTRAINT. Can be
```

(continues on next page)

(continued from previous page)

```
                        given multiple times.
  -S, --start-symbol START_SYMBOL
                        The grammar start symbol (default: '<start>').
  --max, --maximize MAXCONSTRAINT
                        Define an additional constraint MAXCONSTRAINT to be
                        maximized. Can be given multiple times.
  --min, --minimize MINCONSTRAINTS
                        Define an additional constraint MINCONSTRAINT to be
                        minimized. Can be given multiple times.
  -I, --include-dir DIR
                        Specify a directory DIR to search for included
                        Fandango files.
  --file-mode {text,binary,auto}
                        Mode in which to open and write files (default is
                        'auto': 'binary' if grammar has bits or bytes, 'text'
                        otherwise).
  --no-cache            Do not cache parsed Fandango files.
  --no-stdlib           Do not include the standard Fandango library.

Output settings:
  -s, --separator SEPARATOR
                        Output SEPARATOR between individual inputs. (default:
                        newline).
  -d, --directory DIRECTORY
                        Create individual output files in DIRECTORY.
  -x, --filename-extension FILENAME_EXTENSION
                        Extension of generated file names (default: '.txt').
  --format {string,bits,tree,grammar,value,repr,none}
                        Produce output(s) as string (default), as a bit
                        string, as a derivation tree, as a grammar, as a
                        Python value, in internal representation, or none.
  --validate            Run internal consistency checks for debugging.

Generation settings:
  -N, --max-generations MAX_GENERATIONS
                        Maximum number of generations to run the algorithm
                        (ignored if --infinite is set).
  --infinite            Run the algorithm indefinitely.
  --population-size POPULATION_SIZE
                        Size of the population.
  --elitism-rate ELITISM_RATE
                        Rate of individuals preserved in the next generation.
  --crossover-rate CROSSOVER_RATE
                        Rate of individuals that will undergo crossover.
  --mutation-rate MUTATION_RATE
                        Rate of individuals that will undergo mutation.
  --random-seed RANDOM_SEED
                        Random seed to use for the algorithm. You probably
                        also want to specify 'PYTHONHASHSEED=<some-value>' to
                        achieve full reproducibility.
  --destruction-rate DESTRUCTION_RATE
                        Rate of individuals that will be randomly destroyed in
                        every generation.
  --max-repetition-rate MAX_REPETITION_RATE
                        Rate at which the number of maximal repetitions should
                        be increased.
  --max-repetitions MAX_REPETITIONS
```

(continues on next page)

(continued from previous page)

```
                        Maximal value the number of repetitions can be
                        increased to.
  --max-node-rate MAX_NODE_RATE
                        Rate at which the maximal number of nodes in a tree is
                        increased.
  --max-nodes MAX_NODES
                        Maximal value, the number of nodes in a tree can be
                        increased to.
  -n, --desired-solutions, --num-outputs DESIRED_SOLUTIONS
                        Number of outputs to produce.
  --best-effort         Produce a 'best effort' population (may not satisfy
                        all constraints).
  -i, --initial-population INITIAL_POPULATION
                        Directory or ZIP archive with initial population.
  --progress-bar {on,off,auto}
                        Whether to show the progress bar. 'auto' (default)
                        shows the progress bar only if stderr is a terminal.

General settings:
  --warnings-are-errors
                        Treat warnings as errors.

Party settings:
  --party PARTY         Only consider the PARTY part of the interaction in the
                        .fan file.

command invocation settings:
  --input-method {stdin,filename,libfuzzer}
                        When invoking COMMAND, choose whether Fandango input
                        will be passed as standard input (`stdin`), as last
                        argument on the command line (`filename`) (default),
                        or to a libFuzzer style harness compiled to a shared
                        .so/.dylib object (`libfuzzer`).
  command               Command to be invoked with a Fandango input.
  args                  The arguments of the command.
```

Some of these options are very useful, such as `-o` and `-d`, which redirect the inputs generated towards individual files. You can also specify a *command* to be executed with the inputs Fandango generated.

We will go through these commands and options in due time. For now, let us get back to our specifications and actually fuzz with Fandango (page 31).

CHAPTER

NINE

FUZZING WITH FANDANGO

9.1 Creating a Name Database

Let us now come up with an input that is slightly more complex. We want to create a set of inputs with *names of persons* and their respective *age*. These two values would be *comma-separated*, such that typical inputs would look like this:

```
Alice Doe,27
John Smith,45
...
```

This makes the overall format of our input look like this:

```
<start> ::= <person_name> "," <age>
```

with `<age>` again being a sequence of digits, and a `<person>`'s name being defined as

```
<person_name> ::= <first_name> " " <last_name>
```

where both first and last name would be a sequence of letters - first, an uppercase letter, and then a sequence of lowercase letters. The full definition looks like this:

In Fandango specs, symbol names are formed like identifiers in Python - that is, they consist of letters, underscores, and digits.

Use *regular expressions* (page 75) to specifying character ranges.

```
<start> ::= <person_name> "," <age>
<person_name> ::= <first_name> " " <last_name>
<first_name> ::= <name>
<last_name> ::= <name>
<name> ::= <ascii_uppercase_letter><ascii_lowercase_letter>+
<age> ::= <digit>+
```

The symbols `<ascii_uppercase_letter>`, `<ascii_lowercase_letter>`, and `<digits>` are predefined in the *Fandango Standard Library* (page 313); they are defined exactly as would be expected.

Create or download a file `persons.fan` and run Fandango on it:

```
$ fandango fuzz -f persons.fan -n 10
```

Your output will look like this:

```
Triguoynjdafetoekxm Kgldymexdtwqpws,1111
```

```
Ysatabpgc Fwgyum,5533268758134511915
Sscjyblupxuf Rpupnzhankokj,80516481580108
Yyfckfqij Xkkvcly,4
Ndnrjb Hpobzggdxhgyqaeagt,04993563161446399139
```

```
Sajjtpwkfuqardizte Jldkytibfata,29383683791097737795
Sxrruy Jlmpxu,83778072578645626
Sv Fni,5636
Slooxugkztrcezizfofl Qywektqhsdvewurw,9183716800854532083
Zkhsocun Uoowdydtwjvbcyo,603288
```

Such random names are a typical result of *fuzzing* – that is, testing with randomly generated values. How do we get these into a program under test?

9.2 Feeding Inputs Into Programs

So far, we have had Fandango simply *output* the strings it generates. This is nice for understanding and debugging, but not necessarily useful for processing these inputs further. Fandango provides a number of means to help you process its output.

9.3 Feed Inputs into a Single file

Fandango has a `-o` option that directs its output into a single file. To store the generated strings in a file named `persons.txt`, use `-o persons.txt`:

```
$ fandango fuzz -f persons.fan -n 10 -o persons.txt
```

By default, the generated strings are separated by newlines. With the `-s` option, you can define an alternate separator. To have the outputs generated separated by `:`, for instance, use

```
$ fandango fuzz -f persons.fan -n 10 -o persons.txt -s ':'
```

9.4 Feed Inputs into Individual Files

With the `-d` option, Fandango can store its strings into individual files in the directory given after `-d`. They can then be picked up for post-processing. To store all outputs in a directory named `persons`, for instance, use `-d persons`:

```
$ fandango fuzz -f persons.fan -n 10 -d persons
```

Fandango will create the directory and store the inputs as `Fandango-0001.txt`, `Fandango-0002.txt`, etc.

Note

If the directory is already present, Fandango will use that directory.

Caution

Fandango will overwrite files `Fandango-0001.txt`, `Fandango-0002.txt`, etc., but leave other files in the directory unchanged.

If you want a different file extension (for instance, because `.txt` is not suitable), Fandango provides a `--filename-extension` option to set a different one.

9.5 Invoking Programs Directly

You can have Fandango invoke *programs directly*, and have Fandango feed them the generated inputs. The programs are specified as arguments on the command line.

There are three ways to pass the input into programs, on the command line, via standard input, and in-process to a shared object with a libFuzzer style harness.

Refer to this example[26] for further details on how a harness can be compiled to interface with each mode.

9.5.1 Passing Inputs on the Command Line

When Fandango invokes programs, you can pass input as a file name as last argument on the program's command line. This is the default, and can also be obtained with the `--input-method=filename` option.

For instance, to test the `wc` program with its own `-c` option and the inputs Fandango generates, use

```
$ fandango fuzz -f persons.fan -n 10 wc -c
```

The `wc` program is then invoked as `wc -c FILE_1`, `wc -c FILE_2`, etc., where each `FILE` contains an individual input from Fandango.

9.5.2 Passing Inputs via Standard Input

As an alternative, Fandango can pass inputs via standard input. For this, use the `--input-method=stdin` option.

For instance, to test the `cat` program with its own `-n` option and the inputs Fandango generates, use

```
$ fandango fuzz -f persons.fan -n 10 --input-method=stdin cat -n
```

The `cat` program is then invoked repeatedly, each time passing a new Fandango-generated input as its standard input.

[26] https://github.com/fandango-fuzzer/fandango/tree/main/evaluation/experiments/libfuzzer-harness

9.5.3 Calling a libFuzzer style harness directly

Fandango further supports calling a libFuzzer style harness directly. If you are able to compile your binary to a shared object (`.so` on Linux or `.dylib` on macOS), Fandango can load the binary and directly call the function. This requires some extra work initially but removes the need to start a new process for each input evaluation, thus improving performance significantly. Fandango will exit on the first crashing input. `--input-method=libfuzzer` is only supported in combination with `--file-mode=binary`, since libFuzzer style harnesses expect binary data.

Fandango is invoked as follows:

```
$ fandango fuzz -f persons.fan -n 10 --input-method=libfuzzer --file-mode=binary ./
↪harness.{so,dylib}
```

Warning

If your target crashes when using the libFuzzer interface, the fuzzer process will exit as well, losing the input if not otherwise specified. Consider using the *LibAFL integration* (page 34) for much more robust target and fuzzing process handling.

Note

The libFuzzer interface is untested on Windows so far due to a lack of hardware. If you would like to test it, try un-skipping `test_output_with_libfuzzer_harness` in `test_cli.py`. You may need to adjust how the C file is compiled.

9.6 Using Fandango in LibAFL

With libafl-fandango-pyo3 (crates.io[27], repository[28]), it is possible to use Fandango in conjunction with the LibAFL[29] ecosystem. This makes target handling and evaluation much more flexible.

Refer to the documentation of libafl-fandango-pyo3[30] for instructions on how to use it and a couple of example fuzzers showing how to integrate it into LibAFL-based fuzzers.

9.7 Executable `.fan` files

On a Unix system, you can turn a `.fan` file into an *executable file* by placing a line

```
#!/usr/bin/env -S fandango fuzz -f
```

at the top.

If you set its "executable" flag with `chmod +x FILE`, you can then directly execute the `.fan` file as a command as if it were prefixed by `fandango fuzz -f`.

As an example, let us create a file `fuzz-persons.fan`:

[27] https://crates.io/crates/libafl-fandango-pyo3
[28] https://github.com/riesentoaster/libafl-fandango-pyo3
[29] https://github.com/aflplusplus/libafl
[30] https://github.com/riesentoaster/libafl-fandango-pyo3

```
$ (echo '#!/usr/bin/env -S fandango fuzz -f'; cat persons.fan) > fuzz-persons.fan
$ chmod +x fuzz-persons.fan
```

You can now invoke the file, even with extra arguments:

```
$ ./fuzz-persons.fan -n 1
```

```
Xglsl Iawicgwb,5337
```

Tip

The `env` command varies greatly across Unix flavors and versions. If the above does not work on your system, try skipping the `-S` option, such that the first line reads:

```
#!/usr/bin/env fandango fuzz -f
```

9.8 Running Fuzzing Campaigns

So far, we have only produced a limited number of outputs. This can be done in two ways: By limiting the number of generations for the evaluation or by capping the number of solutions:

```
$ fandango fuzz -f persons.fan -n 10 # to limit the number of solutions
$ fandango fuzz -f persons.fan -N 10 # to limit the number of generations
```

To run fuzzing continuously, consider specifying `--infinite` to keep Fandango producing inputs indefinitely. You likely want to combine this with one of the configurations where Fandango *executes your target directly* (page 33) like so:

```
$ fandango fuzz -f persons.fan --infinite --input-method=libfuzzer --file-mode=binary␣
↪./harness.{so,dylib}
```

CHAPTER

TEN

SOME FUZZING STRATEGIES

The idea of fuzzing is to have inputs that are *out of the ordinary*, so we can detect errors in input parsers and beyond. But let us again have a look at the inputs we generated so far:

```
$ fandango fuzz -f persons.fan -n 10
```

```
Cidixrhubl Gyjif,404598
Fslnpedi Zptqgmdfh,45909060041605930
Tpdwezopdhwvfd Pphlc,56805179915190843
Nidmugrsuppbecyuh Stnfmebutmovogpwzqvn,207075874
```

```
Osjgkgzy Elvffsitgazbm,893199717388965
Wy Hofn,01720
Qkmpkswuiqdr Qwusytpa,843
Ahonqqtc Xhrtrjixfdzf,5
Wqflvn Eyntmbwyvezkaqt,3946957
Eveeojahlogodg Nytziduyzinphuowxa,569520653129821
```

Despite clearly looking non-natural to humans, the strings we generated so far are unlikely to trigger errors in a program, because programs typically treat all letters equally. So let us bring a bit more *weirdness* into our inputs.

Danger

Don't feed such fuzz inputs into other people's systems; this can have unpredictable consequences.

Warning

In most countries, even *trying* to feed such inputs into other people's systems will get you into jail.

10.1 Looooooong Inputs

One way to increase the probability of detecting bugs is to test with *long* inputs. While processing data, many programs have limited storage for individual data fields, and thus need to cope with inputs that exceed this storage space. If programmers do not care for such long inputs, serious errors may follow. So-called *buffer overflows* have long been among the most dangerous vulnerabilities, as they can be exploited to gain unauthorized access to systems.

We can easily create long inputs by specifying the *number of repetitions* in our grammar:

- Appending `{N}` to a symbol, where `N` is a number, makes Fandango repeat this symbol `N` times.

- Appending `{N,M}` to a symbol makes Fandango repeat this symbol between `N` and `M` times.
- Appending `{N,}` to a symbol makes Fandango repeat this symbol at least `N` times.

> The `+`, `*`, and `?` suffixes are actually equivalent to `{1,}`, `{0,}`, and `{0,1}`, respectively.

If, for instance, we want the above names to be 100 characters long, we can set up a new rule for `<name>`

```
<name> ::= <ascii_uppercase_letter><ascii_lowercase_letter>{99}
```

and the lowercase letters will be repeated 99 times.

This is the effect of this rule:

```
Ewezlydwkuwwiqvwnoifwyyecfncwagbhewpqfmkgnjvoaiazatrwfmvanpgskenkxnwxyfqytkdmjptrlcpagenvchurdnam
 ↪Tdiqylyjxraaxcbzxrmmzfneofpcwhumdzcvmfkmkslllhohmiyclngheexyghmnjpnjyxjvqgtkelvplvgmathqbkeomfq
 ↪5
```

```
Znsdxdfqaaxaudsllnsjacpkzjwmrhbjgzdppfdmlielwbepbzlkohuneqoygbtjqbttzvxbozfoerquxwiisbnseljullwfa
 ↪Uzfapcbkqgxyxbcadpbnwtsbgmuzeoloqknjihaltmlekzfdpomakxetnzqzcxanxtowkuqsuutkbwhhqipbmomrugnudge
 ↪0
```

```
Ujyxjbypirlixjfubvkrippmrihyexeljohsbllmyzpezynkkgmjtgeftneqwccfgxoaixcarjpzcvrdbgitfxladrjklucdt
 ↪Uwsdcgrdwfcxukdsdpxhewmedgobopfdsvmlkrpmnygldvgceugkdmjvhqwycakwjooracbwvujkfudpqrxucglemlqrjqt
 ↪6
```

```
Wkaqylosqpaudaklbuvdkjmortrclrhahccjklmydfrkhdmduavecsesqecfavisgflbzossiqkpohneloivervkeaxgckyau
 ↪Vynkwnsmrrigvzmjvsczeafarjxatcfwmfmknecprwhibqznyjfwvznvjctzvpzgmlrfrwmplhthuhephotiygasifobirl
 ↪9
```

```
Itvkglifvzrxjgdpgqbrgjoaguwbjeyepwuhlvguqkojjtadkcfjoldvhbmsudkiplasvvfzojttotvyqmzcufdasaqrwyjze
 ↪Tdecootsdnwzmtrkawweamdwvbtgkogsjljmwmphzbkltadgjxychkekstlmxnmuqqtnkhpvdfskawghcpgnztckvwpkyvp
 ↪8
```

```
Qyakuozxhtvkmbwbgvlxcmtyriiegnwqlnvlolxpcwlpukoibjbovjrtaupvjrrqliecgleghpfttgjkfubxginrazrwdpiqa
 ↪Bizmdijzidetypmqdllaynjvrisvfpeacblyoujcobmcbqhhoaqnxgejlyhatswtcxmidnszulyvztvtzzsmtoeypfrfwgh
 ↪8
```

```
Wwjymhwzsyphukmugtvgeiwrdgpgbnfpauasfykzvsqshihsveglgpsshvomtsgdoonhmswwrnbxglrnkfvakupbpwqydjptz
 ↪Mgnwlyhaesxayekgvphhtxefemzzkygiinzumljoryjronemdlspsnsslrrolxwrcarkewothoivmfzcykdmasglwfndybx
 ↪1
```

```
Waitjrojajevvzylhxfzvgxxviqfsmayltmjmkblxkkbeziajiugitliijpcaxqpvsvvkwilhvuysqietuiefcxqtxxlyzvyc
 ↪Nphrzjjlqzojdzcuuqotcturdartqszqivtlbmwwlvdckrmbulmkywgjagbzgitlirdumagxealplrugjlhoecgqtxfdvmc
 ↪4
```

```
Wqheimsdxcgazrscbnchmxkvqwvasfglxiahnqqithtsfbnbrsonidqpwzchdraoooovntuuixcspmqbsaocdfyohsshzrccdd
 ↪Ryynealtydtggeuykijanxujzjzcfgxzjkewwvvkbhhopoifyclcyaganeewvgtrzjbmymfjglhzithgptcukakomnhgzom
 ↪8
```

```
Nwyanhxxcgeddwdwrwwkaworqndspgrofdqeqzfqedpoaarxytcpxihujzkrakytthutjjhhhnozkdkkpwdcpiryixiddrbnp
↪Wznvzwidyxkuzmjbesjhfewumuffyzfnvwiflyoahcznxxbjwgcmtcfwjknidffmhoplfyerlgwthitjskjdttiziyvwfrb
↪7
```

We see that the names are now much longer. For real-world fuzzing, we may try even longer fields (say, 1,000) to test the limits of our system.

Tip

Programmers often make "off-by-one" errors, so if an input is specified to have at most `N` characters, you should test this exact boundary - say, by giving `N` and `N+1` characters.

Danger

Don't try this with other people's systems; the consequences are unpredictable.

10.2 Unusual Inputs

Having "weird" inputs also applies to numerical values. Think about our "age" field, for instance. What happens if we have a person with a negative age?

Note that the grammar already can produce ages way above 100.

Try it yourself and modify `persons.fan` such that it can also produce negative numbers, as in

```
Qntmsrit Xeqsehooxhtgehcnbnxow,9971010
```

```
Cmbnargrekqxdpzx Dewlmfpjfnpmkflvx,-1840773
Qrnua Cfoppteltfyizqoagujk,-297885171624
Htiaspfulqqlnqiqz Ha,69468213766702900031
Iksrofmbnsureuatp Psbix,-2257245774
```

```
Zyvq Mutaltikpyzxabpqpl,765598896112149
Xoxf Touwyxpjczhqlepzbhk,-3380
Jknffbacujjyeaslmt Jsqivzqoi,9965661
Mbbiytevijwlpylbvpii Wbtdodsnnkcric,04726006368291568 74
```

```
Khspftwcxznmsnr Omflmufhhentcsujokq,95483932935301902
```

Did you succeed? Compare your answer against the solution below.

Solution

You can, for instance, change the `<age>` rule such that it introduces an alternative for negative numbers:

```
<age> ::= <digit>+ | "-" <digit>+
```

Another way to do it is to use the `?` modifier, which indicates an optional symbol:

```
<age> ::= "-"? <digit>+
```

Other kinds of unusual inputs would be character sets that are out of the ordinary - for instance, Chinese or Hebrew characters - or plain Latin characters if your system expects Chinese names. A simple Emoji, for instance, could be enough to cause the system to fail.

In Python, try `float('nan') * float('inf')` and other variations.

For numbers, besides being out of range, there are a few constants that are interesting. Some common number parsers and converters accept values such as `Inf` (infinity) or `NaN` (not a number) as floating-point values. These actually *are* valid and have special rules – anything multiplied with `Inf` also becomes infinitely large (`Inf` times zero is zero, though); and any operation involving a `NaN` becomes `NaN`.

Fortunately, number converters typically treat `NaN` as invalid.

Imagine what happens if we manage to place a `NaN` value in a database? Any computation involving this value would also become a `NaN`, so in our example, the average age of persons would become `NaN`. The `NaN` could even go viral across Excel sheets, companies, shareholder reports, and eventually the stock market. Luckily, programs are prepared against that - or are they?

Danger

Don't try this with other people's systems; the consequences are unpredictable.

10.3 String Injections

Another kind of attack is to insert special *strings* into the input – strings that would be interpreted by the program not as data, but as *commands*. A typical example for this is a *SQL injection*. Many programs use the SQL, the *structured query language*, as a means to interact with databases. A command such as

```
INSERT INTO CUSTOMERS VALUES ('John Smith', 34);
```

would be used to save the values `John Smith` (name) and `34` (age) in the `CUSTOMERS` table.

A SQL injection uses specially formed strings and values to subvert these commands. If our "age" value, for instance, is not `34`, but, say

```
34); CREATE TABLE PWNED (Phone CHAR(20)); --
```

then creating the above `INSERT` command with this special "age" could result in the following command:

Two dashes (`--`) start a comment in SQL.

```
INSERT INTO CUSTOMERS VALUES ('John Smith', 34); CREATE TABLE PWNED (Phone CHAR(20));␣
↪--);
```

and suddenly, we have "injected" a new command that will alter the database by adding a `PWNED` table.

How would one do this with a grammar? Well, for the above, it suffices to have one more alternative to the `<age>` rule.

Solution

Here's how one could change the `<age>` rule:

```
<age> ::= <digit>+ | "-" <digit>+
        | <digit>+ "); CREATE TABLE PWNED (Phone CHAR(20)); --"
```

Try adding such alternatives to *all* data fields processed by a system; feed the Fandango-generated inputs to it; and if you then find a `PWNED` table on your system, you know that you have a vulnerability.

Danger

Don't try this with other people's systems; the consequences are unpredictable.

Part III

Advanced Input Generation

CHAPTER

ELEVEN

SHAPING INPUTS WITH CONSTRAINTS

11.1 The Limits of Grammars

So far, all the operations we have performed on our data were *syntax-oriented* - that is, we could shape the format and structure of our values, but not their *semantics* - that is, their actual meaning. Consider our *Person/Age dataset from the previous section* (page 31). What would we do if we want the "age" in a specific range? Or we want it to be odd? Or we want the age to be distributed in a certain way?

All of this refers to *context-free grammars*, which are the ones Fandango uses.

Some of these can be obtained by altering the grammar - limiting the age to two digits at most, for instance, will keep the value below 100 - but others cannot. Properties that cannot be expressed in a grammar are called *semantic properties* - in contrast to *syntactical properties*, which is precisely what grammars are good for.

11.2 Specifying Constraints

Fandango solves this problem through a pretty unique feature: It allows users to specify *constraints* which inputs have to satisfy. These thus narrow down the set of possible inputs.

Constraints are *predicates over grammar symbols*. Essentially, you write a Boolean expression, using grammar symbols (in `<...>`) to refer to individual elements of the input.

As an example, consider this Fandango constraint:

```
int(<age>) < 50
```

This constraint takes the `<age>` element from the input and converts it into an integer (all symbols are strings in the first place). Inputs are produced only if the resulting value is less than 50.

We can add such constraints to any .fan file, say the `persons.fan` file from the previous section. Constraints are preceded by a keyword `where`. So the line we add reads

```
where int(<age>) < 50
```

and the full `persons.fan` file reads

```
<start> ::= <person_name> "," <age>
<person_name> ::= <first_name> " " <last_name>
<first_name> ::= <name>
<last_name> ::= <name>
<name> ::= <ascii_uppercase_letter><ascii_lowercase_letter>+
<age> ::= <digit>+
where int(<age>) < 50
```

If we do this and run Fandango, we obtain a new set of inputs:

```
$ fandango fuzz -f persons.fan -n 10
```

```
Amuiisviu Mgdexudoteg,29
Foy Kruwrbhgzzc,32
```

```
Lctpri Fdxtbpouecllth,8
```

```
Uqihnphzvpwnyproceqp Umspwp,1
```

```
Ibggaokkyai Jjzfvfibosr,1
```

```
Ibggaokkyai Jjzfvfibosn,1
Uqihnphzvpwryproceqp Umspwp,1
Amuiisviu Krmtbplhbaohygws,29
Gvmfjhtpuxirhoawqx Qoswflaniphxcgcddgi,8
Amuiisviu Nunplookxeylqomtihv,29
```

We see that all persons produced now indeed have an age of less than 50. Even if an age begins with `0`, it still represents a number below 50.

The language Fandango uses to express constraints is Python, so you can make use of arbitrary Python expressions. For instance, we can use Python Boolean operators (`and`, `or`, `not`) to request values in a range of 25-45:

Interestingly, having symbols in `<...>` does not conflict with the rest of Python syntax. Be sure, though, to leave spaces around < and > operators to avoid confusion.

```
25 <= int(<age>) and int(<age>) <= 45
```

and we obtain these inputs:

```
Lpa Yldjqgcxhcgc,030
```

```
Yu Jzupdjzi,35
```

```
Opombxfufrsbyfqu Yfizpsrssnxddoqyulmb,45
```

```
Opomxxfufrsbyfqu Yfizpsrssnxddoqyulmb,45
Lpa Zhmurwsruhucypqrytnyd,030
```

```
Feypjzlcka Sbleccv,35
```

```
Epa Yldjqgcxhcgc,030
Yu Jzupdjzi,30
```

```
Nllmyqujgwq Tz,37
```

```
Yu Sbleccv,35
```

Start with `persons.fan` and add a constraint such that we generate people whose age is a multiple of 7, as in

```
Keh Lcpkedvfk,9626488949
```

```
Ldqnh Gmjxaecvjzhavvjvtnkn,3259841131232
Uuig Ozlajzuuvialqpgrymfvt,950957959329
```

```
Pwdnyuckmfutywxid Zmquddjgwf,421662075856
```

```
Jgbrntihpbtuljhgnwhp Fcskovvyfyhqqbpwrjd,356384004648
```

```
Izieujllhdagf Apmongzooszc,0
```

```
Ppqrkyjx Hgrxn,587546935031
```

```
Dpc Asvuazkzdwxtbotoadkg,1482613508193444935
Mwnucrofnhr Yhcavg,1254380267688751
```

```
Poceupurynathfncuy Zekrspeqcusrtgixzeel,80505782
```

(Hint: The modulo operator in Python is `%`).

Solution

This is not too hard. Simply add

```
where int(<age>) % 7 == 0
```

as a constraint.

11.3 Constraints and `DerivationTree` types

Whenever Fandango evaluates a constraint, such as

```
int(<age>) > 20
```

the type of `<age>` is actually not a string, but a `DerivationTree` object - *a tree representing the structure of the output.* (page 87). You can use `DerivationTree` objects as other basic python data types by converting them using (`int(<age>)`, `str(<age>)`, `bytes(<age>)`).

- You can then invoke most *string, bytes and int methods* on them (`<age>.startswith('0')`) (see *Details* (page 325))
- You can *compare* them against each other (`<age_1> == <age_2>`) as well as against other strings (`<age> != "19"`)

One thing you *cannot* do, though, is *passing them directly as arguments to functions* that do not expect a `DerivationTree` type. This applies to the vast majority of Python functions.

Important

If you want to pass a symbol as a function argument, convert it to the proper type (`int(<age>)`, `float(<age>)`, `str(<age>)`) first. Otherwise, you will likely raise an internal error in that very function.

Important

On symbols, the `[...]` operator operates differently from strings - it returns a *subtree* of the produced output: `<name>[0]` returns the `<first_name>` element, not the first character. If you want to access a character (or a range of characters) of a symbol, convert it into a string first, as in `str(<name>)[0]`.

We will learn more about derivation trees, `DerivationTree` types, and their operators in *Accessing Input Elements* (page 87).

11.4 Constraints on the Command Line

If you want to experiment with constraints, keeping on editing `.fan` files is a bit cumbersome. As an alternative, Fandango also allows to specify constraints on the command line. This is done with the `-c` (constraint) option, followed by the constraint expression (typically in quotes).

Starting with the original `persons.fan`, we can thus apply age constraints as follows:

```
$ fandango fuzz -f persons.fan -n 10 -c '25 <= int(<age>) and int(<age>) <= 45'
```

Constraints can be given multiple times, so the above can also be obtained as

```
$ fandango fuzz -f persons.fan -n 10 -c '25 <= int(<age>)' -c 'int(<age>) <= 45'
```

Important

On the command line, always put constraints in single quotes (`'...'`), as the angle brackets might otherwise be interpreted as I/O redirection.

When do constraints belong in a `.fan` file, and when on the command line? As a rule of thumb:

- If a constraint is *necessary* for obtaining valid input files (i.e. if the inputs would not be accepted otherwise), it belongs into the `.fan` file.
- If a constraint is *optional*, for instance for shaping inputs towards a particular goal, then it can also go on the command line.

11.5 How Fandango Solves Constraints

How does Fandango obtain these inputs? In a nutshell, Fandango is an *evolutionary* test generator:

1. It first uses the grammar to generate a *population* of inputs.
2. It then checks which individual inputs are *closest* in fulfilling the given constraints. For instance, for a constraint `int(<X>) == 100`, an input where `<X>` has a value of 90 is closer to fulfillment than one with value of, say 20.

Selecting the best inputs is also known as “survival of the fittest”

3. The best inputs are selected, the others are discarded.
4. Fandango then generates new *offspring* by *mutating* the remaining inputs, recomputing parts according to grammar rules. It can also exchange parts with those from other inputs; this is called *crossover*.

5. Fandango then repeats Steps 2-4 until all inputs satisfy the constraints.

All of this happens within Fandango, which runs through these steps with high speed. The `-v` option (verbose) produces some info on how the algorithm progresses:

```
$ fandango -v fuzz -f persons.fan -n 10 -c 'int(<age>) % 7 == 0'
```

```
fandango:INFO: ---------- Parsing FANDANGO content ----------
fandango:INFO: <stdlib>: loading cached spec from /Users/zeller/Library/Caches/
 ↪Fandango/6ce667e6ac3405c5ae9aee6e21c982a086df69bc50a44ec8794a3ccad7b79284.pickle
fandango:INFO: persons.fan: loading cached spec from /Users/zeller/Library/Caches/
 ↪Fandango/a05ebfcfcae6fc3d5e0da576ca445a8983e6fbb3ec5b98aede2948b1edb8dae5.pickle
fandango:INFO: <string>: loading cached spec from /Users/zeller/Library/Caches/
 ↪Fandango/7e6ba852352a9eaf1c7d60eaefdedcf429bba2bbded84cfe344a57ce0ed81f3d.pickle
fandango:INFO: int(<age>) % 7 == 0: loading cached spec from /Users/zeller/Library/
 ↪Caches/Fandango/f4f4cd0f76ec36bab3cc0c5d0de2e3653a40cbf869a76e8d8a3dc3f4579eed99.
 ↪pickle
```

```
fandango:INFO: File mode: text
fandango:INFO: ---------- Initializing base population ----------
fandango:INFO: ---------- Initializing FANDANGO algorithm ----------
fandango:INFO: ---------- Done initializing base population ----------
fandango:INFO: ---------- Generating  for 500 generations----------
fandango:INFO: Generating (additional) initial population (size: 100)...
```

```
Nyfnqxhilqnz Zfbtsopbce,115446316000638
Lqauvzfldnav Rravsh,180800401783752107
Xtaw Qepnsfv,56018457995827792472
```

```
Uwlrbwb Kzoqqtz,0
```

```
Kjw Mkpkjmikd,85914288944357384938
```

```
Necqabsqsxbaqmtmlpu Pnmkh,82656083111
```

```
Xjfxcxjvxy Oa,0
```

```
Sbubtwiv Zwicsbkz,0
```

```
Axcszobhxeehh Gunxpjwvkmcqjrwqb,65016185577
Ech Tkqpinevgbbttcrsbp,61894284094584991
```

Note

The `-v` option comes right after `fandango` (and not after `fandango fuzz`), as `-v` affects all commands (and not just `fuzz`).

11.6 The Fandango Progress Bar

While Fandango is solving constraints, you may see a *progress bar* in the terminal. The progress bar looks like this:

The progress bar is composed of three parts. On the leftmost side, we have the *Fandango logo* ("🐓 Fandango"), followed by a *generation counter* ("6/500") showing how often (6) the population has evolved (out of a maximum 500).

Most of the line, however, is filled by a *fitness visualization* illustrating how the fitness is distributed across the inputs in the population. Each fraction of the line corresponds to an equal fraction of individual inputs. Hence, a 1/70 of the line (typically one character) stands for 1/70 of the population.

The *color* of each fraction how *fit* the inputs in the fraction are - on a scale from *bright green* (perfect fitness, fulfilling the given constraints) to *dark red* (very little fitness, far away from fulfilling the constraints). Depending on its capabilities, your terminal may also show shades between these colors. Inputs that do not satisfy the constraints at all (zero fitness) are shown in gray.

In the above example, we can see that Fandango already has produces a few inputs that satisfy the constraints; a few more are close and may get there through further evolution.

> **Note**
>
> By default, the progress bar only shows up if
>
> - Fandango's standard error is a terminal;
> - Fandango is not run within Jupyter notebook (Jupyter cannot interpret the terminal escape sequences); and
> - Fandango logging is turned off (it also writes to standard error).
>
> The option `--progress-bar=on` turns on the progress bar even if the above conditions are not met. The option `--progress-bar=off` turns the progress bar off.

11.7 Soft Constraints and Optimization

So far, we have seen constraints that *have* to be satisfied for Fandango to produce a string. On top, Fandango also supports so-called "soft" constraints that Fandango *aims* to satisfy as good as it can. These "soft" constraints come in two forms:

- **maximizing** constraints: These constraints specify an *expression* whose value should be as *high* as possible
- **minimizing** constraints: These constraints specify an expression whose value should be as *low* as possible.

Such soft constraints are specified

- on the command line, using `--maximizing EXPR` and `--minimizing EXPR`, respectively; or
- in the `.fan` file, introducing them with `minimizing` and `maximizing` (instead of `where`), respectively.

If, for instance, you want Fandango to maximize the `<age>` field, you write

```
$ fandango fuzz -f persons.fan --maximize 'int(<age>)'
```

```
Riozhxwlzkluuaulqv Xoodnb,78354674710126481503
```

```
Riozhxwlzkluuaulqv Roodnb,78354674710126481503
```

```
Riozhxwlzkluuaulqv Xoodnb,78354674710126481506
```

```
Riozhxwlzkluuaulqv Xoodnb,78354674710126781503
Riozhxwlzkluuaulqv Roodnb,78358674710126481503
```

```
Riozhxwlzkluuaulqv Xoodnb,78754674710126481503
```

```
Riozhxwlzkluuaulqv Roodnb,88354674710126481503
```

```
Riozhxulzkluuaulqv Roodnb,88354674710126481503
```

```
Riozhxwlzkluuaulqv Roodnb,88354674710126481505
```

```
Riozhxwlzkluuaulqv Roodnq,88354674710126481503
```

Conversely, minimizing the `<age>` field yields

```
$ fandango fuzz -f persons.fan --minimize 'int(<age>)'
```

```
Geetlkpeghae Je,632
```

```
Hmowlveudbr Gbcouhvseglizvrbt,60
Wl Cvfzrlqzqewf,60
Nlvefdzfr Doglwil,33
```

```
Vqp Ulbbgqfsvkscwtp,3
```

```
Cjsq Kcsufqbxblxioewngryls,3
```

```
Efidxazvtv Uhkc,2
```

```
Ianetxmhxxnk Uhkc,2
```

```
Efidxazvtv Uhec,2
```

```
Hmowlveudbr Gbcouhvseglizvrbt,00
```

Alternatively, you could also add to the `.fan` file:

```
maximizing int(<age>)
```

or

```
minimizing int(<age>)
```

respectively.

To express optional goals (i.e., real “soft” constraints), simply use a *Boolean* expressions as the expressions for `--maximize` or `--minimize`. Then, Fandango will aim to maximize (or minimize) its value.

> **Note**
>
> Remember that in Python `True` is equivalent to 1, and `False` is equivalent to 0; therefore, "maximizing" a Boolean value means that Fandango will aim to solve it.

Here is an example of a "soft" Boolean constraints, aiming to obtain names that start with "F":

```
$ fandango fuzz -f persons.fan --maximize '<name>.startswith("A")' -n 10
```

```
Ynlcmksqusl Xwzhcrdw,010909
Wayhcme Xjwxwnruekscyezsbciz,797488
Phyhtrifkzutx Mlqmvdxhhdfkijgg,10925964679
```

```
Smbxlqygcleie Ejuebhayaxermgtt,75131578489
Yzplhvqtyx Puoqlqqnaglykhtpmqufx,8750286855156
Twsvdmtpdk Usmvy,50
Sztzbz Nicpq,04673934268818150210
```

```
Vhtjzawqfg Tqddpcjkjzlpjhwzdjkgv,163790964627716002
It Cbj,04808
Ccgbyycuatm Hvxyic,7222900
```

As you see, "soft" constraints are truly optional :-)

11.8 When Constraints Cannot be Solved

Normally, Fandango continues evolving the population until all inputs satisfy the constraints. Some constraints, however, may be difficult or even impossible to solve. After a maximum number of generations (which can be set using `-N`), Fandango stops and produces the inputs it has generated so far. We can see this if we specify `False` as a constraint:

The `-N` option limits the number of generations - the default is 500.

```
$ fandango -v fuzz -f persons.fan -n 10 -c 'False' -N 50
```

```
fandango:INFO: ---------- Parsing FANDANGO content ----------
fandango:INFO: <stdlib>: loading cached spec from /Users/zeller/Library/Caches/
 ↪Fandango/6ce667e6ac3405c5ae9aee6e21c982a086df69bc50a44ec8794a3ccad7b79284.pickle
```

```
fandango:INFO: persons.fan: loading cached spec from /Users/zeller/Library/Caches/
 ↪Fandango/a05ebfcfcae6fc3d5e0da576ca445a8983e6fbb3ec5b98aede2948b1edb8dae5.pickle
fandango:INFO: <string>: loading cached spec from /Users/zeller/Library/Caches/
 ↪Fandango/7e6ba852352a9eaf1c7d60eaefdedcf429bba2bbded84cfe344a57ce0ed81f3d.pickle
fandango:INFO: False: saving spec to cache /Users/zeller/Library/Caches/Fandango/
 ↪a642373e2c47cbe8c115a474f6525845c477777eb8552831cbe7ce7afd173492.pickle
```

```
fandango:INFO: File mode: text
fandango:INFO: ---------- Initializing base population ----------
```

(continues on next page)

(continued from previous page)

```
fandango:INFO: ---------- Initializing FANDANGO algorithm ----------
fandango:INFO: ---------- Done initializing base population ----------
fandango:INFO: ---------- Generating  for 50 generations----------
fandango:INFO: Generating (additional) initial population (size: 100)...
```

```
fandango:INFO: Initial population generated in 1.29 seconds
```

```
fandango:INFO: Generation 1 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 1: Increasing mutation rate from 0.200 to 0.220
fandango:INFO: Generation 1: Decreasing crossover rate from 0.800 to 0.760
fandango:INFO: Generation 1: Increasing MAX_REPETITION from 20 to 30
fandango:INFO: Generation 1 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 2 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 2: Increasing mutation rate from 0.220 to 0.242
fandango:INFO: Generation 2: Decreasing crossover rate from 0.760 to 0.722
fandango:INFO: Generation 2: Increasing MAX_REPETITION from 30 to 45
fandango:INFO: Generation 2 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 3 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 3: Increasing mutation rate from 0.242 to 0.266
fandango:INFO: Generation 3: Decreasing crossover rate from 0.722 to 0.686
fandango:INFO: Generation 3: Increasing MAX_REPETITION from 45 to 68
fandango:INFO: Generation 3 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 4 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 4: Increasing mutation rate from 0.266 to 0.293
fandango:INFO: Generation 4: Decreasing crossover rate from 0.686 to 0.652
fandango:INFO: Generation 4: Increasing MAX_REPETITION from 68 to 102
```

```
fandango:INFO: Generation 4 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 5 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 5: Increasing mutation rate from 0.293 to 0.322
fandango:INFO: Generation 5: Decreasing crossover rate from 0.652 to 0.619
fandango:INFO: Generation 5: Increasing MAX_REPETITION from 102 to 153
fandango:INFO: Generation 5 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 6 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 6: Increasing mutation rate from 0.322 to 0.354
fandango:INFO: Generation 6: Decreasing crossover rate from 0.619 to 0.588
fandango:INFO: Generation 6: Increasing MAX_REPETITION from 153 to 230
```

```
fandango:INFO: Generation 6 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 7 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 7: Increasing mutation rate from 0.354 to 0.390
fandango:INFO: Generation 7: Decreasing crossover rate from 0.588 to 0.559
fandango:INFO: Generation 7: Increasing MAX_REPETITION from 230 to 345
```

```
fandango:INFO: Generation 7 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 8 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 8: Increasing mutation rate from 0.390 to 0.429
fandango:INFO: Generation 8: Decreasing crossover rate from 0.559 to 0.531
fandango:INFO: Generation 8: Increasing MAX_REPETITION from 345 to 518
```

```
fandango:INFO: Generation 8 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 9 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 9: Increasing mutation rate from 0.429 to 0.472
fandango:INFO: Generation 9: Decreasing crossover rate from 0.531 to 0.504
fandango:INFO: Generation 9: Increasing MAX_REPETITION from 518 to 777
fandango:INFO: Generation 9 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 10 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 10: Increasing mutation rate from 0.472 to 0.519
fandango:INFO: Generation 10: Decreasing crossover rate from 0.504 to 0.479
fandango:INFO: Generation 10: Increasing MAX_REPETITION from 777 to 1000
fandango:INFO: Generation 10 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 11 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 11: Increasing mutation rate from 0.519 to 0.571
fandango:INFO: Generation 11: Decreasing crossover rate from 0.479 to 0.455
```

```
fandango:INFO: Generation 11 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 12 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 12: Increasing mutation rate from 0.571 to 0.628
fandango:INFO: Generation 12: Decreasing crossover rate from 0.455 to 0.432
fandango:INFO: Generation 12 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 13 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 13: Increasing mutation rate from 0.628 to 0.690
fandango:INFO: Generation 13: Decreasing crossover rate from 0.432 to 0.411
```

```
fandango:INFO: Generation 13 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 14 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 14: Increasing mutation rate from 0.690 to 0.759
fandango:INFO: Generation 14: Decreasing crossover rate from 0.411 to 0.390
fandango:INFO: Generation 14 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
```

(continues on next page)

(continued from previous page)

```
↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 15 - Average Fitness: 0.01

fandango:INFO: Generation 15: Increasing mutation rate from 0.759 to 0.835
fandango:INFO: Generation 15: Decreasing crossover rate from 0.390 to 0.371

fandango:INFO: Generation 15 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 16 - Average Fitness: 0.01

fandango:INFO: Generation 16: Increasing mutation rate from 0.835 to 0.919
fandango:INFO: Generation 16: Decreasing crossover rate from 0.371 to 0.352

fandango:INFO: Generation 16 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 17 - Average Fitness: 0.01

fandango:INFO: Generation 17: Increasing mutation rate from 0.919 to 1.000
fandango:INFO: Generation 17: Decreasing crossover rate from 0.352 to 0.334
fandango:INFO: Generation 17 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 18 - Average Fitness: 0.01

fandango:INFO: Generation 18: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 18: Decreasing crossover rate from 0.334 to 0.318
fandango:INFO: Generation 18 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 19 - Average Fitness: 0.01

fandango:INFO: Generation 19: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 19: Decreasing crossover rate from 0.318 to 0.302
fandango:INFO: Generation 19 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
↪diversity: 0.01, Population size: 100

fandango:INFO: Generation 20 - Average Fitness: 0.01

fandango:INFO: Generation 20: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 20: Decreasing crossover rate from 0.302 to 0.287

fandango:INFO: Generation 20 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 21 - Average Fitness: 0.01

fandango:INFO: Generation 21: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 21: Decreasing crossover rate from 0.287 to 0.272

fandango:INFO: Generation 21 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 22 - Average Fitness: 0.01

fandango:INFO: Generation 22: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 22: Decreasing crossover rate from 0.272 to 0.259
```

(continues on next page)

(continued from previous page)

```
fandango:INFO: Generation 22 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 23 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 23: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 23: Decreasing crossover rate from 0.259 to 0.246
fandango:INFO: Generation 23 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 24 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 24: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 24: Decreasing crossover rate from 0.246 to 0.234
```

```
fandango:INFO: Generation 24 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 25 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 25: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 25: Decreasing crossover rate from 0.234 to 0.222
fandango:INFO: Generation 25 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
```

```
fandango:INFO: Generation 26 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 26: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 26: Decreasing crossover rate from 0.222 to 0.211
fandango:INFO: Generation 26 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 27 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 27: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 27: Decreasing crossover rate from 0.211 to 0.200
fandango:INFO: Generation 27 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 28 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 28: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 28: Decreasing crossover rate from 0.200 to 0.190
```

```
fandango:INFO: Generation 28 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 29 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 29: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 29: Decreasing crossover rate from 0.190 to 0.181
fandango:INFO: Generation 29 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 30 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 30: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 30: Decreasing crossover rate from 0.181 to 0.172
fandango:INFO: Generation 30 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
```

(continues on next page)

(continued from previous page)

```
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 31 - Average Fitness: 0.01

fandango:INFO: Generation 31: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 31: Decreasing crossover rate from 0.172 to 0.163
fandango:INFO: Generation 31 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 32 - Average Fitness: 0.01

fandango:INFO: Generation 32: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 32: Decreasing crossover rate from 0.163 to 0.155

fandango:INFO: Generation 32 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 33 - Average Fitness: 0.01

fandango:INFO: Generation 33: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 33: Decreasing crossover rate from 0.155 to 0.147
fandango:INFO: Generation 33 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 34 - Average Fitness: 0.01

fandango:INFO: Generation 34: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 34: Decreasing crossover rate from 0.147 to 0.140

fandango:INFO: Generation 34 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 35 - Average Fitness: 0.01

fandango:INFO: Generation 35: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 35: Decreasing crossover rate from 0.140 to 0.133

fandango:INFO: Generation 35 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 36 - Average Fitness: 0.01

fandango:INFO: Generation 36: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 36: Decreasing crossover rate from 0.133 to 0.126
fandango:INFO: Generation 36 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 37 - Average Fitness: 0.01

fandango:INFO: Generation 37: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 37: Decreasing crossover rate from 0.126 to 0.120

fandango:INFO: Generation 37 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 38 - Average Fitness: 0.01

fandango:INFO: Generation 38: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 38: Decreasing crossover rate from 0.120 to 0.114
```

```
fandango:INFO: Generation 38 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 39 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 39: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 39: Decreasing crossover rate from 0.114 to 0.108
```

```
fandango:INFO: Generation 39 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 40 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 40: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 40: Decreasing crossover rate from 0.108 to 0.103
```

```
fandango:INFO: Generation 40 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 41 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 41: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 41: Decreasing crossover rate from 0.103 to 0.100
fandango:INFO: Generation 41 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 42 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 42: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 42: Decreasing crossover rate from 0.100 to 0.100
fandango:INFO: Generation 42 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 43 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 43: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 43: Decreasing crossover rate from 0.100 to 0.100
```

```
fandango:INFO: Generation 43 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 44 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 44: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 44: Decreasing crossover rate from 0.100 to 0.100
```

```
fandango:INFO: Generation 44 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 45 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 45: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 45: Decreasing crossover rate from 0.100 to 0.100
fandango:INFO: Generation 45 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 46 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 46: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 46: Decreasing crossover rate from 0.100 to 0.100
```

```
fandango:INFO: Generation 46 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 47 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 47: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 47: Decreasing crossover rate from 0.100 to 0.100
```

```
fandango:INFO: Generation 47 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 48 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 48: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 48: Decreasing crossover rate from 0.100 to 0.100
fandango:INFO: Generation 48 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 49 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 49: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 49: Decreasing crossover rate from 0.100 to 0.100
fandango:INFO: Generation 49 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Generation 50 - Average Fitness: 0.01
```

```
fandango:INFO: Generation 50: Increasing mutation rate from 1.000 to 1.000
fandango:INFO: Generation 50: Decreasing crossover rate from 0.100 to 0.100
fandango:INFO: Generation 50 stats -- Best fitness: 0.01, Avg fitness: 0.01, Avg␣
 ↪diversity: 0.01, Population size: 100
fandango:INFO: Average fitness of population: 0.01
```

```
fandango:INFO: ---------- Done generating  for 50 generations----------
fandango:INFO: Time taken: 133.48 seconds
fandango:ERROR: Population did not converge to a perfect population
fandango:ERROR: Only found 0 perfect solutions, instead of the required 10
```

As you see, Fandango produces a population of zero. Of course, if the constraint is `False`, then there can be no success.

Tip

Fandango has a `--best-effort` option that allows you to still output the final population.

CHAPTER

TWELVE

THE FANDANGO SHELL

Sometimes, you may find cumbersome to invoke Fandango from the command line again and again. This is especially true when

- You want to explore the effects of different constraints.
- You want to explore the effects of different algorithm settings.

This is why Fandango offers a *shell*, in which

- you can enter commands directly at a Fandango prompt
- you can set and edit parameters *once* for future commands
- settings are preserved during your session
- your command history is preserved across sessions.

12.1 Invoking the Fandango Shell

Invoking the Fandango shell is easy. Just invoke Fandango without a command:

```
$ fandango
```

and you will be greeted by the Fandango prompt

```
(fandango)
```

12.2 Invoking Commands

At the `(fandango)` prompt, you can enter the same commands you already know from the command line, such as

```
(fandango) fuzz -f persons.fan -n 10
```

Tip

Use TAB to complete commands, options, and file names.

and you will get the same results:

```
Dyvxibta Ow,02
Rsyyjybvxgbafdjvlrk Facgcjbc,2
Gebur Zznkyyckxjxcqjsvmam,658536121516
```

```
Fcemmxuubuznnfhz Hn,18792151928850482475
Kfbctrjhxacnhiovcjxyd Tbxakeavasyq,42444232814216192
Ugbvpsbvy Xhchldo,8
Tnj Srjw,25552780938521
```

```
Eurgwgxt Vmdainqukokeswufmrdjb,0705041
Idhokadzdrqsgstdzg Dnkk,249450432217
Qxruoctgsgehrgtvqt Huoolwkxfnxlyyyupox,79261
```

There is an important advantage though: You can set (and edit) a *common environment* for all commands.

12.3 Setting a Common Environment

The `set` command (available only in the Fandango shell) allows you to specify resources and settings that are then applied to all later commands (notably, `fuzz`).

You can, for instance, specify a `.fan` file with

```
(fandango) set -f persons.fan -n 10
```

> **Note**
>
> A big advantage of the `set` command is that the `.fan` file is read only *once*, and then available for all later commands.

After options for resources and settings are set, you can omit them from later `fuzz` commands:

```
(fandango) fuzz
```

> **Note**
>
> If you give `fuzz` additional options, these temporarily *override* the settings given with `set` earlier. As an exception, *constraints* (`-c`) are *added* to those already set.

The options for the `set` command are roughly the same as for the `fuzz` command and include

- `.fan` files (`set -f FILE`)
- constraints (`set -c CONSTRAINT`)
- algorithm settings (`set --mutation-rate=0.2`)

To get a full list of options, try `help set`.

> **Note**
>
> Some command-specific options like `-o` or `-d` (controlling the output of the `fuzz` command) are not available for setting with `set`.

12.4 Retrieving Settings

To retrieve the current settings, simply enter `set`. This will list:

- the grammar and constraints of the current `.fan` file
- as well as any constraints or settings you have made

Here is an example:

```
(fandango) set -N 10
(fandango) set
<start> ::= <person_name> ',' <age>
<person_name> ::= <first_name> ' ' <last_name>
<first_name> ::= <name>
<last_name> ::= <name>
<name> ::= <ascii_uppercase_letter> <ascii_lowercase_letter>+
<age> ::= <digit>+
--max-generations=10
```

12.5 Resetting Settings

To reset all settings to default values, use the `reset` command:

```
(fandango) reset
```

This will

- clear all constraints defined with `set`
- reset all algorithm settings to their default value.

The current `.fan` file stays unchanged.

> **Note**
>
> Loading a new `.fan` file also clears all `set` constraints.

12.6 Quotes and Escapes

The Fandango shell uses the same quoting conventions as a POSIX system shell. For instance, to set a constraint, place it in quotes:

```
(fandango) set -c 'int(<age>) < 30'
```

You can also escape individual characters with backslashes:

```
(fandango) set -c int(<age>)\ <\ 30
```

12.7 Editing Commands

The Fandango shell uses the GNU readline library. Therefore, you can

- Use cursor keys *left* and *right* to edit commands
- Use cursor keys *up* and *down* to scroll through history
- Use the *TAB* key to expand command names, options, and arguments.

The command history is saved in `~/.fandango_history`.

12.8 Invoking Commands from the Shell

In the Fandango shell, you can invoke *system commands* by prefixing them with `!`:

```
(fandango) !ls
LICENSE.md                docs                          requirements.txt
Makefile                 language                 src
README.md                 pyproject.toml                 tests
```

Tip

Use TAB to complete file names.

You can also invoke and evaluate *Python commands* by prefixing them with `/`:

```
(fandango) /import random
(fandango) /random.randint(10, 20)
11
```

If the command you enter has a value, the value is automatically printed.

Note

Invoking system and Python commands is only available when the input is a terminal.

12.9 Changing the Current Directory

To change the current directory, Fandango provides a built-in `cd` command:

The alternative `!cd` does not work, as this changes the directory of the invoked shell.

```
(fandango) cd docs/
```

Without arguments, `cd` switches to the home directory.

Important

This is different from Windows, where `cd` reports the current directory.

12.10 Getting Shell Commands from a File

Instead of entering commands by keyboard, one can also have Fandango read in commands from a file or another command. This is done by redirecting the `fandango` standard input. To have Fandango read and execute commands from, say, `commands.txt`, use

```
$ fandango < commands.txt
```

Fandango can also process the commands issued from another program:

```
$ echo 'fuzz -n 10 -f persons.fan' | fandango
```

```
Ppkiadrczbmajhlogl Xl,791
Ixzorqwichle Qgjustwfwxe,6181803412559
Xvomeasaqcgojqmzpg Xsccqwefcthdhjyfwjns,15108398016128843
Kaekpiy Btkanpe,6061406101039847176
```

```
Dgijckdjmub Rxbti,467
Rpqswu Wfcculdmigfx,795226753308857596
Oxuxlmxefp Wxnb,59843
Xqwnq Mphp,2658
Oplohbiakqps Jxqzfreyfa,3558524768936555264
```

```
Ykrxpcuquffsovtlphyl Ickrbrrghczbscki,570449351694630
```

Tip

The input file can contain blank lines as well as comments prefixed with `#`.

Note

System commands (`!`) and Python commands (`/`) are not available when reading from a file.

12.11 Exiting the Fandango Shell

To exit the Fandango shell and return to the system command line, enter the command `exit`:

```
(fandango) exit
$
```

Entering an EOF (end-of-file) character, typically `Ctrl-D`, will also exit the shell.

```
(fandango) ^D
$
```

In the next section, we'll talk about *custom generators* (page 67).

CHAPTER

THIRTEEN

DATA GENERATORS AND FAKERS

Often, you don't want to generate *totally* random data; it suffices that *some* aspects of it are random. This naturally raises the question: Where can one get non-random, *natural* data from, and how can one integrate this into Fandango?

13.1 Augmenting Grammars with Data

The straightforward solution would be to simply extend our grammar with more natural data. In order to obtain more natural first and last names in our *ongoing names/age example* (page 31), for instance, we could simply extend the `persons.fan` rule

```
<first_name> ::= <name>
```

to

> These are the given names on Pablo Picasso[31]'s birth certificate.

```
<first_name> ::= <name> | "Alice" | "Bob" | "Eve" | "Pablo Diego José Francisco de␣
↪Paula Juan Nepomuceno Cipriano de la Santísima Trinidad"
```

and extend the rule

```
<last_name> ::= <name>
```

to, say,

```
<last_name> ::= <name> | "Doe" | "Smith" | "Ruiz Picasso"
```

then we can have Fandango create names such as

```
Pablo Diego José Francisco de Paula Juan Nepomuceno Cipriano de la Santísima␣
↪Trinidad Pmhmsvwjiz,99673
```

```
Eve Smith,831717
Oklzddjneoojmh Efericfmfxeqfzr,46380377172130378
Eve Doe,231011709915
Zxq Doe,7433
```

(continues on next page)

[31] https://en.wikipedia.org/wiki/Pablo_Picasso

(continued from previous page)

```
Pablo Diego José Francisco de Paula Juan Nepomuceno Cipriano de la Santísima␣
 ↪Trinidad Doe,0794
Pablo Diego José Francisco de Paula Juan Nepomuceno Cipriano de la Santísima␣
 ↪Trinidad Doe,6190301759548427011
Pablo Diego José Francisco de Paula Juan Nepomuceno Cipriano de la Santísima␣
 ↪Trinidad Smith,17870961304167622
Sihiardele Ruiz Picasso,2380341200056489224
```

```
Eve Doe,179811924514827
```

Note that we still get a few "random" names; this comes as specified by our rules. By default, Fandango picks each alternative with equal likelihood, so there is a 20% chance for the first name and a 25% chance for the last name to be completely random.

Note

Future Fandango versions will have means to control these likelihoods.

13.2 Using Fakers

Frequently, there already are data sources available that you'd like to reuse – and converting each of their elements into a grammar alternative is inconvenient. That is why Fandango allows you to *specify a data source as part of the grammar* - as a *Python function* that supplies the respective value. Let us illustrate this with an example.

The Python `faker` module is a great source of "natural" data, providing "fake" data for names, addresses, credit card numbers, and more.

Here's an example of how to use it:

```
from faker import Faker
fake = Faker()
for i in range(10):
  print(fake.first_name())
```

```
Ryan
Charlene
Tammy
Margaret
Angela
Dwayne
Amanda
Charlotte
Jenna
Theresa
```

Have a look at the `faker` documentation[32] to see all the fake data it can produce. The methods `first_name()` and `last_name()` are what we need. The idea is to extend the `<first_name>` and `<last_name>` rules such that they can draw on the `faker` functions. To do so, in Fandango, you can simply extend the grammar as follows:

[32] https://faker.readthedocs.io/

A generator applies to *all* alternatives, not just the last one.

```
<first_name> ::= <name> := fake.first_name()
```

`:=` is the assignment operator in several programming languages; in Python, it can be used to assign values within expressions.

The *generator* `:= EXPR` *assigns* the value produced by the expression `EXPR` (in our case, `fake.first_name()`) to the symbol on the left-hand side of the rule (in our case, `<first_name>`).

Important

Whatever value the generator returns, it must be *parseable* by at least one of the alternatives in the rule. Our example works because `<first_name>` matches the format of `fake.first_name()`.

Tip

If your generator returns a string, a "match-all" rule such as

```
<generated_string> ::= <char>* := generator()
```

will fit all possible string values returned by `generator()`.

We can do the same for the last name, too; and then this is the full Fandango spec persons-faker.fan:

```
from faker import Faker
fake = Faker()

include('persons.fan')

<first_name> ::= <name> := fake.first_name()
<last_name> ::= <name> := fake.last_name()
```

Note

The *Fandango `include()` function* (page 341) *includes* the Fandango definitions of the given file. This way, we need not repeat the definitions from `persons.fan` and only focus on the differences.

Note

Python code (from Python files) that you use in a generator (or in a constraint, for that matter) needs to be imported. Use the Python `import` features to do that.

Important

`include(FILE)` is for Fandango files, `import MODULE` is for Python modules.

This is what the output of the above spec looks like:

```
Amanda Rivas,26904535354230662
Morgan Small,6
David Griffith,09310915853488010
```

```
Joanne Smith,2694180329365350
Ashley Carter,8
Kristen Bean,4027688198944447
Brenda Patel,758426
```

```
Kristin Rowe,797674088703204136
Tyler Reed,4842
John Singleton,4265
```

You see that all first and last names now stem from the Faker library.

13.3 Number Generators

In the above output, the "age" fields are still very random, though. With generators, we can achieve much more natural distributions.

After importing the Python `random` module:

```
import random
```

we can make use of dozens of random number functions[33] to use as generators. For instance, `random.randint(A, B)` return a random integer n such that $A \leq n \leq B$ holds. To obtain a range of ages between 25 and 35, we can thus write:

```
<age> ::= <digit>+ := str(random.randint(25, 35));
```

Important

All Fandango generators must return strings or byte strings.

- Use `str(N)` to convert a number N into a string
- Use `bytes([N])` to convert numbers N into bytes.

The resulting Fandango spec file produces the desired range of ages:

```
Kimberly Campbell,29
Eric Oconnor,34
Brett Robinson,35
```

[33] https://docs.python.org/3/library/random.html

```
Samuel Adams,34
Katherine Lee,34
Christopher Green,26
Brenda Navarro,33
```

```
Amanda Yoder,35
Michelle Bauer,30
Cindy Sanchez,30
```

We can also create a Gaussian (normal) distribution this way:

```
<age> ::= <digit>+ := str(int(random.gauss(35)));
```

`random.gauss()` returns floating point numbers. However, the final value must fit the given symbol rules (in our case, `<digit>+`), so we convert the age into an integer (`int()`).

These are the ages we get this way:

```
Michael Fernandez,36
James Cox,34
Leslie Smith,36
Kimberly Tucker,34
```

```
Katelyn Watkins,35
Stephanie Watkins,34
Patricia Bishop,34
Eric Nolan,34
Kyle Johnson,34
```

```
Cynthia Fowler,35
```

In sec:distributions, we will introduce more ways to obtain specific distributions.

13.4 Generators and Random Productions

In testing, you want to have a good balance between *common* and *uncommon* inputs:

- Common inputs are important because they represent the typical usage, and you don't want your program to fail there;
- Uncommon inputs are important because they uncover bugs you may *not* find during alpha or beta testing, and thus avoid latent bugs (and vulnerabilities!) slipping into production.

We can easily achieve such a mix by adding rules such as

```
<first_name> ::= <name> | <natural_name>
<natural_name> ::= <name> := fake.first_name()
```

With this, both random names (`<name>`) and natural names (`<natural_name>`) will have a chance of 50% to be produced:

```
Srjhexu Mcfarland,7910
Ltdmotlmjqgzpzp Trujillo,5750243
Matthew Roberson,0665638226
April Cochran,949436574264223
```

```
Ashley Reyes,0309493759
Vbyuvyfmfcdpbd Bowers,19750365065
Rvfbohiswghipb Juarez,46
Onrejvrjhmnohxm Ryan,755750
Charles Leonard,8506
```

```
Bnctfu Ramos,275570922838
```

13.5 Combining Generators and Constraints

When using a generator, why does one still have to specify the format of the data, say `<name>`? This is so for two reasons:

> To allow access to the elements of the generator output, Fandango *parses* the output according to the symbol rules.

1. It allows the Fandango spec to be used for *parsing* existing data, and consequently, *mutating* it;
2. It allows additional constraints (page 45) to be applied on the generator result and its elements.

In our example, the latter can be used to further narrow down the set of names. If we want all last names to start with an `S`, for instance, we can invoke Fandango as

> Grammar symbols `<...>` support all Python string methods[34].

```
$ fandango fuzz -f persons-faker.fan -c '<last_name>.startswith("S")' -n 10
```

and we get

```
Frances Shepard,812415953420500
```

```
Heather Stewart,590
```

```
Julia Shaw,70
```

```
Robert Sheppard,5154656937930953203
```

```
Cynthia Silva,588909
Nicholas Shaw,75949648964849612
```

```
April Sloan,870920536626892702
```

```
Jeffrey Scott,0141213
```

[34] https://docs.python.org/3/library/stdtypes.html#string-methods

```
Michael Shelton,6239749095083322
```

```
Maria Sloan,67882
```

13.6 When to use Generators, and when Constraints

One might assume that instead of a generator, one could also use a *constraint* to achieve the same effect. So, couldn't one simply add a constraint that says

```
<first_name> == fake.first_name()
```

> In case this should work, this is only through some internal Fandango optimization that directly assigns computed values to symbols.

Unfortunately, this does not work.

The reason is that the faker returns *a different value* every time it is invoked, making it hard for Fandango to solve the constraint. Remember that Fandango solves constraints by applying mutations to a population, getting closer to the target with each iteration. If the target keeps on changing, the algorithm will lose guidance and will not progress towards the solution.

Likewise, in contrast to our example in *Combining Generators and Constraints* (page 72), one may think about using a *constraint* to set a limit to a number, say:

```
$ fandango fuzz -f persons-faker.fan -c '<last_name>.startswith("S")' -c 'int(<age>) >
 ↪= 25 and int(<age>) <= 35' -n 10
```

This would work:

```
Morgan Schultz,0028
```

```
Morgan Stanton,0028
Clayton Schultz,0028
```

```
Anthony Schultz,0028
```

```
Clayton Stanton,0028
```

```
Clayton Simpson,0028
```

```
Clayton Scott,0028
Anthony Simpson,0028
```

```
Samantha Schwartz,0028
```

```
Clayton Schultz,0025
```

But while the values will fit the constraint, they will not be randomly distributed. This is because Fandango treats and generates them as *strings* (= sequences of digits), ignoring thur semantics as numerical values. To obtain well-distributed numbers from the beginning, use a generator.

1. If a value to be produced is *random*, it should be added via a *generator*.
2. If a value to be produced is *constant*, it can go into a *generator* or a *constraint*.
3. If a value to be produced must be *part of a valid input*, it should go into a *constraint*. (Constraints are checked during parsing *and* production.)

CHAPTER

FOURTEEN

REGULAR EXPRESSIONS

Although the Fandango grammars cover a wide range of input language features, there are situations where they may be a bit cumbersome to work with. Consider specifying *every digit except for zeros*: this requires you to enumerate all the other digits `1`, `2`, and so on. This is why Fandango also supports *regular expressions*, which allow you to use a concise syntax for character ranges, repeated characters and more. Specifying all digits from `1` to `9`, for instance, becomes the short regular expression `r'[1-9]'`.

14.1 About Regular Expressions

Regular expressions form a language on their own and come with several useful features. To get an introduction to the regular expressions Fandango uses, read the Python Regular Expression HOWTO[35] and check out the Python Regular Expression Syntax[36] for a complete reference.

In Fandango, regular expressions are used for two purposes:

- When *producing* inputs, a regular expression is instantiated into a random string that matches the expression.
- When *parsing* inputs, a regular expression is used to *parse* and *match* inputs.

14.2 Writing Regular Expressions

> For Python aficionados: this is actually a Python "raw string"

In Fandango, a regular expression comes as a string, prefixed with a `r` character. To express that a digit can have the values `0` to `9`, instead of

```
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

you can write

```
<digit> ::= r'[0-9]'
```

which is much more concise.

Likewise, to match a sequence of characters that ends in `;`, you can write

[35] https://docs.python.org/3/howto/regex.html
[36] https://docs.python.org/3/library/re.html#regular-expression-syntax

```
<some_sequence> ::= r'[^;]+;'
```

Besides the `r` prefix indicating a regular expression, it also makes the string a *raw* string. This means that backslashes are treated as *literal characters*. The regular expression `\d`, for instance, matches a Unicode digit, which includes `[0-9]`, and also many other digit characters[37]. To include `\d` in a regular expression, write it *as is*; do not escape the backslash with another backslash (as you would do in a regular string):

The expression `r'\\d'` would actually match a backslash, followed by a `d` character.

```
<any_digit> ::= r'\d'
```

Important

Be aware of the specific syntax of `r`-strings as it comes to backslashes.

One consequence of backslashes being interpreted literally is that you cannot escape quote characters in a regular expression. This causes a problem if you need two kinds of quotes (`"` and `'`) in the same regular expression – say, a rule that checks for forbidden characters.

However, encodings of the form `\xNN` are also interpreted by regular expressions. Hence, if you need quotes, you can use

- `\x22` instead of `"`
- `\x27` instead of `'`

Here is an example:

```
<forbidden_characters> ::= r'[\x22\x27;]'
```

14.3 Fine Points about Regular Expressions

For parsing inputs, Fandango uses the Python `re`[38] module for matching strings against regular expressions; for producing inputs, Fandango uses the Python `exrex`[39] module for generating strings that match regular expressions. All the `re` and `exrex` capabilities and limitations thus extend to Fandango.

[37] https://en.wikipedia.org/wiki/Numerals_in_Unicode
[38] https://docs.python.org/3/library/re.html
[39] https://github.com/asciimoo/exrex

14.3.1 Repetition Limits

Most notably, `exrex` imposes a *repetition limit* of 20 on generated strings that in principle can have arbitrary length; a `+` or `*` operator will not expand to more than 20 repetitions. Thus, a grammar `infinity.fan`

```
<start> ::= r"(abc)+"
```

that in principle, could produce arbitrary long sequences `abcabcabcabc...` will be limited to 20 repetitions at most:

```
$ fandango fuzz -f infinity.fan -n 10
```

```
abcabcabc
abcabcabcabcabcabcabc
abcabcabcabcabc
abcabcabcabcabcabcabcabcabcabc
abc
abcabcabcabcabcabcabcabcabcabcabc
abcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabc
abcabcabcabc
abcabcabcabcabcabcabcabcabcabcabc
abcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabcabc
```

To precisely control the number of repetitions, use the regular expression `{m,n}` construct, limiting the number of repetitions from `m` to `n`. Let us limit the number of repetitions to the range 1…5:

```
<start> ::= r"(abc){1,5}"
```

This is what we get:

```
$ fandango fuzz -f finity.fan -n 5
```

```
abcabcabc
abc
abcabcabcabc
abcabcabcabcabc
abcabc
```

Tip

Remember that *grammars* also have operators `+`, `*`, `?`, and `{N,M}` which apply to the preceding grammar element, and work like their *regular expression* counterparts. Using these, we could also write the above as

```
<start> ::= "abc"+
```

and

```
<start> ::= "abc"{1,5}
```

respectively.

14.3.2 Regular Expressions over Bytes

Regular expressions can also be formed over bytes. See *Bytes and Regular Expressions* (page 133) for details.

14.4 Regular Expressions vs. Grammars

In theory, context-free grammars are a strict *superset* of regular expressions - any language that can be expressed in a regular expression can also be expressed in an equivalent grammar. Practical implementations of regular expressions break this hierarchy by introducing some features such as *backreferences* (check out what `(?P=name)` does), which cannot be expressed in grammars.

In many cases, a grammar can be replaced by a regular expression and vice versa. This raises the question: When should one use a regular expression, and when a grammar? Here are some points to help you decide.

- Regular expressions are often more *concise* (but arguably harder to read) than grammars.
- If you want to *reference* individual elements of a string (say, as part of a constraint now or in the future), use a *grammar*.
- Since their underlying model is simpler, regular expressions are *faster* to generate, and *much faster* to *parse* (page 159) than grammars.
- If your underlying language separates lexical and syntactical processing, use
  - *regular expressions* for specifying *lexical* parts such as tokens and fragments;
  - a *grammar* for the *syntax*; and
  - *constraints* (page 45) for any semantic properties.
- Prefer grammars and constraints over overly complex regular expressions.

Important

Do not use regular expressions for inputs that are *recursive* (page 81). Languages like HTML, XML, even e-mail addresses or URLs, are much easier to capture as grammars.

14.5 Regular Expressions as Equivalence Classes

The choice of grammars vs. regular expressions also affects the Fandango generation algorithm. Generally speaking, Fandango attempts to cover all alternatives of a grammar. If, say, `<digits>` is specified as

```
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

then Fandango will attempt to produce every digit at least once, and also try to cover digit *combinations* up to a certain depth. This is useful if you want to specifically test digit processing, or if each of the digits causes a different behavior that needs to be covered.

If, however, you specify `<digits>` as

```
<digit> ::= r'[0-9]'
```

then Fandango will treat this as a *single* alternative (with all expansions considered semantically equivalent), which once expanded into (some) digit will be considered as covered.

Tip

- If you do want or need to *differentiate* between individual elements of a set (because they would be treated differently), consider *grammar alternatives*.
- If you do *not* want or need to differentiate between individual elements of a set (because they would all be treated the same), consider a *regular expression*.

CHAPTER

FIFTEEN

COMPLEX INPUT STRUCTURES

The *context-free grammars* that Fandango uses can specify very complex input formats. In particular, they allow specifying *recursive* inputs - that is, element types that can contain other elements of the same type again. In this chapter, we explore some typical patterns as they occur within grammars.

15.1 Recursive Inputs

A textbook example of a recursive input formats is an *arithmetic expression*. Consider an operation such as addition (+). Its operands can be *numbers* (`3 + 5`), but can also be *other expressions*, as in `2 + (3 - 8)`. In a grammar for arithmetic expressions, this relationship is expressed as a rule

Why don't we write `<expr> ::= <expr> " + " <expr>;` here? If we did that, a string such as `1 + 2 + 3` would become *ambiguous*: it will either be interpreted (1) as `(1 + 2) + 3` – that is, the left `<expr>` becomes `1 + 2`, and the right `<expr>` becomes `3`; or (2) as `1 + (2 + 3)` - that is, the left `<expr>` becomes `1`, and the right `<expr>` becomes `2 + 3`. Such ambiguities may not be important in *producing* inputs. But when *parsing* inputs, any ambiguities lead to interpretation and performance problems.

```
<expr> ::= <term> " + " <expr>
```

which indicates that the right-hand side of the addition + operator can be *another expression*. This is an example of a *recursive* grammar rule - a rule where an expansion refers back to the defined symbol.

Let us add a definition for `<term>`, too, defining it as a number:

```
<term> ::= <number>
<number>  ::= <digit>+
<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

What is it that `<expr>` can now expand into? Have a look at the above rule first, and then check out the solution.

Solution

Actually, the above rules mean that `<expr>` would expand into an infinitely long string `<term> + <term> + <term> + ...`

What we have here is a case of *infinite recursion* – the string would keep on expanding forever.

Important

At this time, Fandango does not detect infinite recursions; it keeps running until manually stopped.

In order to avoid infinite recursion, we need to provide a *non-recursive alternative*, as in:

```
<expr> ::= <term> " + " <expr> | <term>
```

With this rule, we can now store the above definitions in a `.fan` file additions.fan and get Fandango running:

```
$ fandango fuzz -f additions.fan -n 10
```

```
9499625777 + 924081490437
```

```
5 + 658351544937489593
86736151822
14095149758 + 12140169 + 11638482386 + 5119 + 32302972611653
56621906
695
006896
54120 + 93828
820618644 + 6477166
927928 + 375
```

We see that the above rules yield nice (recursive) chains of additions.

Important

For each recursion in the grammar, there must be a non-recursive alternative.

15.2 More Repetitions

In our definition of `<number>`, above, we used the + operator to state that an element should be repeated:

```
<number> ::= <digit>+
```

Instead of using the + operator, though, we can also use a recursive rule. How would one do that?

We could also write

```
<number> ::= <number> <digit> | <digit>
```

However, if the recursive element is the *last* in a rule, this allows for more efficient parsing. We prefer such *tail recursion* whenever we can.

Solution

Here's an equivalent `<number>` definition that comes without +:

```
<number> ::= <digit> <number> | <digit>
```

Indeed, we can define a `<number>` as a `<digit>` that is followed by another "number".

Tip

Using shorthands such as `*`, `+`, and `?` can make Fandango parsers more efficient.

15.3 Arithmetic Expressions

Let us put all of the above together into a grammar for arithmetic expressions. expr.fan defines additional operators (`-`, `*`, `/`), unary `+` and `-`, as well as subexpressions in parentheses. It also ensures the conventional order of operations[40], giving multiplication and division a higher rank than addition and subtraction. Hence, `2 * 3 + 4` gets interpreted as `(2 * 3) + 4` (10), not `2 * (3 + 4)` (14).

```
<start>   ::= <expr>
<expr>    ::= <term> " + " <expr> | <term> " - " <expr> | <term>
<term>    ::= <term> " * " <factor> | <term> " / " <factor> | <factor>
<factor>  ::= "+" <factor> | "-" <factor> | "(" <expr> ")" | <int>
<int>     ::= <digit>+
```

These are the expressions we get from this grammar:

```
$ fandango fuzz -f expr.fan -n 10
```

```
-440472404295670 / (38432856974 / 84351467564919) - 02995874272276895
++((+242 / 51 / --+--+436920358738 / 49527855 + --52) / +45 + +40) / +40 / -67 -␣
↪334
```

```
+-(-(+1943909 - 58605)) + -+5960452070518112 + (986416 * ++--(-206)) * ++-8
371 / (((90023870647 * 8 * 34 - 991) * -4 * +29 * 21 / 51 * 42) / -6 * -07 - -1) +␣
↪-402
```

```
+(2566292124788796 / (-(+((39 * 54 / 15 + 83) * +69 + +41) + -56)) - -6909) - 035
047304343594 / ((+667 / 60 / 7 / +06 * 11 - +47) * 09 * -0 / 382 / 76 - 67) * 686 -
↪ 564
```

```
(-(+126)) / (310890563835 / 62 * 73 * 34 * 63 * 08 * 21 * 21 / 73 / 53 / 91 * 61) -
↪ +75
++63771509684988 / 99351752187917719399
+19 + 76349862 * +-((+(-+24 * 29 * 24 / 85 / 27 * 96 * 08 + +70) - -+1) + (45)) -␣
↪(+7)
```

```
00059 / 9797 * ((+(((-49) * 86 - 409) / 67 / 95 / 63 * 44)) * 542 * 14 * 66 + +89)␣
↪* 06
```

We see that the resulting expressions can become quite complex. If we had an arithmetic evaluation to test – say, from a programming language - these would all make a good start.

[40] https://en.wikipedia.org/wiki/Order_of_operations

Early Python versions interpreted zero-leading numbers as *octal* numbers, so `011` would evaluate to 9 (not 11). This was prone to errors, so in today's Python, octal numbers need a `0o` prefix (as in `0o11`), and `0` prefixes yield an error.

There are two ways we can still improve the grammar, though.

1. First, many programming languages assign a special meaning to *numbers starting with zero*, so we'd like to get rid of these.
2. Second, we only have *integers* so far. How would one go to extend the grammar with *floating-point numbers*?

Try this out for yourself by extending the above grammar.

Solution

To get rid of numbers starting with `0`, we can introduce a `<lead_digit>`:

```
<int> ::= <digit> | <lead_digit> <digits>
<lead_digit> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Note that the option of having a single `0` as an `<int>` is still there, as a `<digit>` can expand into any digit.

To include floating-point numbers, we can add a `<float>` element:

```
<float> ::= <int> "." <digits>
```

Feel free to further extend this - say, with an optional exponent, or by making the `<int>` optional (`.001`).

The resulting grammar `expr-float.fan` produces all-valid numbers:

```
$ fandango fuzz -f expr-float.fan -n 10
```

```
125984004.17253 * +0
70739005 - -5.120 / +28929474452 * (+-+-+5763071327043 + 233.63 * 2) + (1)
```

```
-4 * -++736468697696720237 * 7 * +4 / 4.324691 * 3.864 / 56754948.96 / 6 - 9.2
(4 * -+(++87 / -9.795287420600251 * -843130647.49) / 205 * -7 + ++1) / +-7 + 9
-3.394475524854100175
```

```
0 / ++-4.8684876212841172511 * --+-715394169 * 41.214705981 / 3 + 3 / -0 / -1
+9
+-46459324057167855203.0003803381 * -7 / +1 / +1 / 4 / -+57 / +6 / 4.1 * +6 + +8
```

```
+(1266976.55065975018414703 / (7 / 3 / +456 + ++8) / (6) / -+1 / 740 + -2) - 8.38
162259025 * 3.1440647789033530149 + -+(+(9) - 5 / -3) * 1.51 / 6 * 618 / 7 * 1
```

With extra constraints, we can now have Fandango produce only expressions that satisfy further properties – say, evaluate to a value above 1000 in Python:

```
$ fandango fuzz -f expr-float.fan -n 10 -c 'eval(str(<start>)) > 1000'
```

```
-7.71392194871751 * (-(-1.881 / (+1) / +-(6.03 * 7 - --4) - 7 * 76) + +3) + 270753
```

```
+-(9.497481725560011911 - 5 - (+(48623.36 - 9) * +--5 * -+6 * -2)) * ++383 + +2
-62830896140998651.2173206 / 198 * 33 * -3 * +3 * +-1 * +8 / 7 * 5.58 * 8 / --2 * -
 ↪9
```

```
+1.28062443228165012 * -+106830942416443501 * (3 / -1 * -3 * -5 - 5886) - 9
```

```
-3508214089956495976.1232068150769 * 407191257311 * +(+-7) / 5
40222248319531.1569901 / 477099751 * 82262804329796101.67 + 6 - 7 / +6
```

```
++-9704609158013 + 0.1665593012862457 - +97261574321845857342 / -8
593562049562179208 * 5 / (++((++--6 / 772 * 2) - +6.9) * 4 - 9 / +3) / +--9 / ++9 /
 ↪ -3
```

```
+34938925388333775.7635175 + 0.3537809572205701252 / 60 * 632586 - 294
```

```
124446.4422032951 / ++9 + +(2 / (4 * -+---3.81657 * (6 / -546 / -6)) / +-9) - 9.130
```

Note that some of these expressions raise divisions by zero errors:

```
ZeroDivisionError: float division by zero
```

In the next section, we'll talk about *accessing input elements* (page 87) in complex inputs, so we can impose further constraints on them.

CHAPTER

SIXTEEN

ACCESSING INPUT ELEMENTS

When dealing with *complex input formats* (page 81), attaching *constraints* (page 45) can be complex, as elements can occur *multiple times* in a generated input. Fandango offers a few mechanisms to disambiguate these, and to specify specific *contexts* in which to search for elements.

16.1 Derivation Trees

So far, we have always assumed that Fandango generates *strings* from the grammar. Behind the scenes, though, Fandango creates a far richer data structure - a so-called *derivation tree* that *maintains the structure of the string* and *allows accessing individual elements*. Every time Fandango sees a grammar rule

```
<symbol> ::= ...
```

it generates a derivation tree whose root is `<symbol>` and whose children are the elements of the right-hand side of the rule.

Let's have a look at our `persons.fan` spec:

```
<start> ::= <person_name> "," <age>
<person_name> ::= <first_name> " " <last_name>
<first_name> ::= <name>
<last_name> ::= <name>
<name> ::= <ascii_uppercase_letter><ascii_lowercase_letter>+
<age> ::= <digit>+
```

The `<start>` rule says

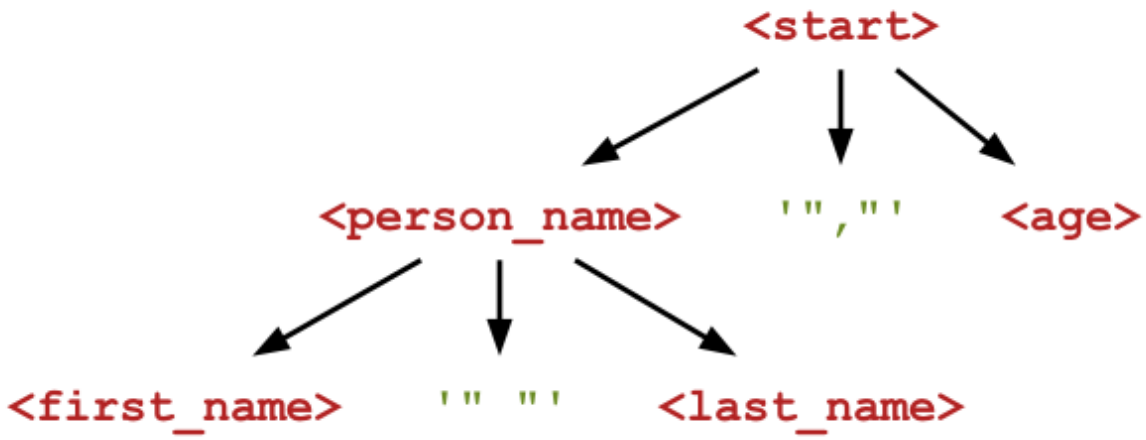
```
<start> ::= <person_name> "," <age>
```

Then, a resulting derivation tree for `<start>` looks like this:

As Fandango expands more and more symbols, it expands the derivation tree accordingly. Since the grammar definition for `<person_name>` says

```
<person_name> ::= <first_name> " " <last_name>
```

the above derivation tree would be extended to

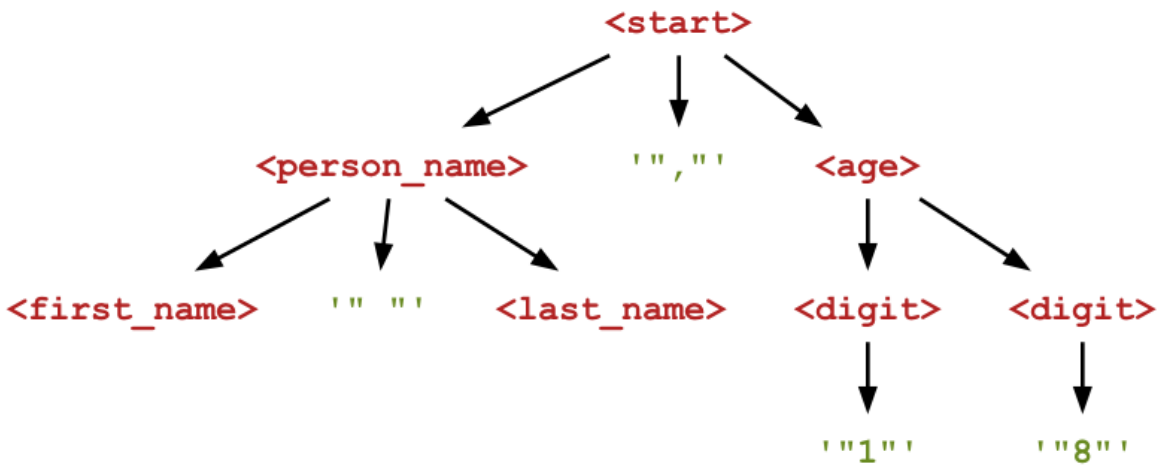


And if we next extend `<age>` and then `<digit>` based on their definitions

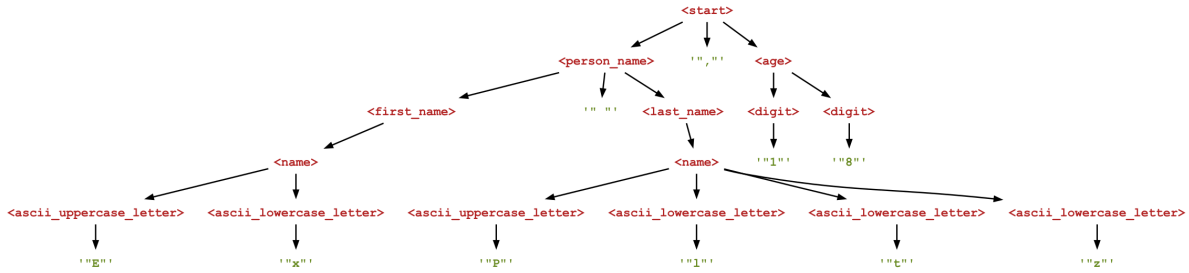
```
<age> ::= <digit>+
```

If one has a Kleene operator like `+`, `*`, or `?`, the elements operated on become children of the symbol being defined. Hence, the `<digit>` elements become children of the `<age>` symbol.

our tree gets to look like this:



Repeating the process, it thus obtains a tree like this:



Note how the tree records the entire history of how it was created - how it was *derived*, actually.

To obtain a string from the tree, we traverse its children left-to-right, ignoring all *nonterminal* symbols (in <...>) and considering only the *terminal* symbols (in quotes). This is what we get for the above tree:

```
Ex Pltz,18
```

And this is the string Fandango produces. However, viewing the Fandango results as derivation trees allows us to access *elements* of the Fandango-produced strings and to express *constraints* on them.

16.1.1 Diagnosing Derivation Trees

To examine the derivation trees that Fandango produces, use the *`--format=grammar` output format* (page 162). This produces the output in a grammar-like format, where children are indented under their respective parents. As an example, here is how to print a derivation tree from `persons.fan`:

```
$ fandango fuzz -f persons.fan -n 1 --format=grammar
```

```
<start> ::= <person_name> ',' <age>  # Position 0x0000 (0); Hqwlsxsvunuqlykmguykt␣
↪Yxvrkmdcyviqthymndxy,6053337261160
  <person_name> ::= <first_name> ' ' <last_name>  # Position 0x0001 (1);␣
↪Hqwlsxsvunuqlykmguykt Yxvrkmdcyviqthymndxy
    <first_name> ::= <name>
      <name> ::= <ascii_uppercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
```

(continues on next page)

(continued from previous page)

```
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter>  # Hqwlsxsvunuqlykmguykt
       <ascii_uppercase_letter> ::= <_ascii_uppercase_letter>
         <_ascii_uppercase_letter> ::= 'H'  # Position 0x0002 (2)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'q'  # Position 0x0003 (3)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'w'  # Position 0x0004 (4)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'l'  # Position 0x0005 (5)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 's'  # Position 0x0006 (6)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'x'  # Position 0x0007 (7)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 's'  # Position 0x0008 (8)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'v'  # Position 0x0009 (9)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'u'  # Position 0x000a (10)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'n'  # Position 0x000b (11)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'u'  # Position 0x000c (12)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'q'  # Position 0x000d (13)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'l'  # Position 0x000e (14)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'y'  # Position 0x000f (15)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'k'  # Position 0x0010 (16)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'm'  # Position 0x0011 (17)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'g'  # Position 0x0012 (18)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'u'  # Position 0x0013 (19)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'y'  # Position 0x0014 (20)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 'k'  # Position 0x0015 (21)
       <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
         <_ascii_lowercase_letter> ::= 't'  # Position 0x0016 (22)
   <last_name> ::= <name>
     <name> ::= <ascii_uppercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter> <ascii_
↪lowercase_letter> <ascii_lowercase_letter> <ascii_lowercase_letter>  #␣
↪Yxvrkmdcyviqthymndxy
       <ascii_uppercase_letter> ::= <_ascii_uppercase_letter>
```

(continues on next page)

(continued from previous page)

```
        <_ascii_uppercase_letter> ::= 'Y'  # Position 0x0017 (23)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'x'  # Position 0x0018 (24)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'v'  # Position 0x0019 (25)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'r'  # Position 0x001a (26)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'k'  # Position 0x001b (27)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'm'  # Position 0x001c (28)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'd'  # Position 0x001d (29)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'c'  # Position 0x001e (30)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'y'  # Position 0x001f (31)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'v'  # Position 0x0020 (32)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'i'  # Position 0x0021 (33)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'q'  # Position 0x0022 (34)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 't'  # Position 0x0023 (35)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'h'  # Position 0x0024 (36)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'y'  # Position 0x0025 (37)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'm'  # Position 0x0026 (38)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'n'  # Position 0x0027 (39)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'd'  # Position 0x0028 (40)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'x'  # Position 0x0029 (41)
      <ascii_lowercase_letter> ::= <_ascii_lowercase_letter>
        <_ascii_lowercase_letter> ::= 'y'  # Position 0x002a (42)
 <age> ::= <digit> <digit> <digit> <digit> <digit> <digit> <digit> <digit> <digit>
↪ <digit> <digit> <digit> <digit>  # 6053337261160
   <digit> ::= <_digit>
     <_digit> ::= '6'  # Position 0x002b (43)
   <digit> ::= <_digit>
     <_digit> ::= '0'  # Position 0x002c (44)
   <digit> ::= <_digit>
     <_digit> ::= '5'  # Position 0x002d (45)
   <digit> ::= <_digit>
     <_digit> ::= '3'  # Position 0x002e (46)
   <digit> ::= <_digit>
     <_digit> ::= '3'  # Position 0x002f (47)
   <digit> ::= <_digit>
     <_digit> ::= '3'  # Position 0x0030 (48)
   <digit> ::= <_digit>
     <_digit> ::= '7'  # Position 0x0031 (49)
   <digit> ::= <_digit>
     <_digit> ::= '2'  # Position 0x0032 (50)
```

(continues on next page)

(continued from previous page)

```
    <digit> ::= <_digit>
      <_digit> ::= '6'  # Position 0x0033 (51)
    <digit> ::= <_digit>
      <_digit> ::= '1'  # Position 0x0034 (52)
    <digit> ::= <_digit>
      <_digit> ::= '1'  # Position 0x0035 (53)
    <digit> ::= <_digit>
      <_digit> ::= '6'  # Position 0x0036 (54)
    <digit> ::= <_digit>
      <_digit> ::= '0'  # Position 0x0037 (55)
```

We see how the produced derivation tree consists of a `<start>` symbol, whose `<first_name>` and `<last_name>` children expand into `<name>` and letters; the `<age>` symbol expands into `<digit>` symbols.

The comments (after `#`) show the individual positions into the input, as well as the values of compound symbols.

What is the full string represented by the above derivation tree?

Solution

You can find on the right-hand side of the first line.

Tip

The `--format=grammar` option is great for debugging, especially *binary formats* (page 129).

16.2 Specifying Paths

One effect of Fandango producing derivation trees rather than "just" strings is that we can define special *operators* that allow us to access *subtrees* (or sub-elements) of the produced strings - and express constraints on them. This is especially useful if we want constraints to apply only in specific *contexts* - say, as part of some element `<a>`, but not as part of an element `<b>`.

16.2.1 Accessing Children

These selectors are similar to XPath, but better aligned with Python. In XPath, the first child has the index 1, in Fandango, it has the index 0.

The expression `<foo>[N]` accesses the `N`-th child of `<foo>`, starting with zero.

If `<foo>` is defined in the grammar as

```
<foo> ::= <bar> ":" <baz>
```

then `<foo>[0]` returns the `<bar>` element, `<foo>[1]` returns `":"`, and `<foo>[2]` returns the `<baz>` element.

In our *persons.fan derivation tree for Ex Pltz* (page 87), for instance, `<start>[0]` would return the `<person_name>` element (`"Ex Pltz"`), and `<start>[2]` would return the `<age>` element (`18`).

We can use this to access elements in specific contexts. For instance, if we want to refer to a `<name>` element, but only if it is the child of a `<first_name>` element, we can refer to it as `<first_name>[0]` - the first child of a `<first_name>` element:

```
<first_name> ::= <name>
```

Here is a constraint that makes Fandango produce first names that end with `x`:

Since a `<first_name>` is defined to be a `<name>`, we could also write `<first_name>.endswith("x")`

```
$ fandango fuzz -f persons.fan -n 10 -c '<first_name>[0].endswith("x")'
```

```
Qcmqrtciqokqvwigx Aqtuenh,61844108062622802793
```

```
Xkx Iyr,980868069931160483
```

```
Qcmqrtciqokqvbigx Aqtuenh,61844108062622802793
Xkx Ivr,980868069931160483
```

```
Qcmqrtciqokqvwigx Bmooyhtf,30826958
```

```
Dkx Iyr,980868069931160483
```

```
Qcmqrtciqokqvwigx Bztpq,61844108062622802793
Xkx Iyr,980868069971160483
```

```
Xkx Iwr,980868069931160483
```

```
Xkx Ixr,980868069931160483
```

Tip

As in Python, you can use *negative* indexes to refer to the last elements. `<age>[-1]`, for instance, gives you the *last* child of an `<age>` subtree.

Important

While symbols act as strings in many contexts, this is where they differ. To access the first *character* of a symbol `<foo>`, you need to explicitly convert it to a string first, as in `str(<foo>)[0]`.

16.2.2 Slices

The Fandango slices maintain the property that `<name>[n:m]` = `<name>[n]` + `<name>[n + 1]` + ... + `<name>[m - 1]`, which holds for all sequences in Python.

Fandango also allows you to use Python *slice* syntax to access *multiple children at once*. `<name>[n:m]` returns a new (unnamed) root which has `<name>[n]`, `<name>[n + 1]`, ..., `<name>[m - 1]` as children. This is useful, for instance, if you want to compare several children against a string:

```
$ fandango fuzz -f persons-faker.fan -n 10 -c '<name>[0:2] == "Ch"'
```

```
Christine Chang,4081447068693
```

```
Christopher Chang,9580558591168114622
Chad Chang,0360690955
```

```
Chad Chapman,0398965045592420
Chad Chang,18566885937
```

```
Christopher Chang,447484190500
Christopher Chapman,447484190500
Chase Chapman,11282220055
```

```
Chad Chapman,0398966025592420
Chad Chang,607083694747
```

Would one also be able to use `<start>[0:2] == "Ch"` to obtain inputs that all start with `"Ch"`?

Solution

No, this would not work. Remember that in derivation trees, indexes refer to *children*, not characters. So, according to the rule

```
<start> ::= <person_name> "," <age>
```

`<start>[0]` is a `<person_name>`, `<start>[1]` is a `","`, and `<start>[2]` is an `<age>`. Hence, `<start>[0:2]` refers to `<start>` itself, which cannot be `"Ch"`.

Indeed, to have the *string* start with `"Ch"`, you (again) need to convert `<start>` into a string first, and then access its individual characters:

```
$ fandango fuzz -f persons-faker.fan -n 10 -c 'str(<start>)[0:2] == "Ch"'
```

```
Charles Lowe,4487045549245965555
```

```
Chad Miles,141873140
Christopher Graham,49
```

```
Christopher Jones,370799
```

```
Christopher Dunn,370799
```

```
Chad Miles,141875140
Chad Miles,141873149
Christopher Martin,077690
Christopher Hudson,016032106
```

```
Chad Dunn,141873140
```

The Fandango slices maintain the property that `<name>[i:] + <name>[:i] = <name>`

Fandango supports the full Python slice semantics:

- An omitted first index defaults to zero, so `<foo>[:2]` returns the first two children.
- An omitted second index defaults to the size of the string being sliced, so `<foo>[2:]` returns all children starting with `<foo>[2]`.
- Both the first and the second index can be negative again.

16.2.3 Selecting Children

Referring to children by *number*, as in `<foo>[0]`, can be a bit cumbersome. This is why in Fandango, you can also refer to elements by *name*.

In XPath, the corresponding operator is `/`.

The expression `<foo>.<bar>` allows accessing elements `<bar>` when they are a *direct child* of a symbol `<foo>`. This requires that `<bar>` occurs in the grammar rule defining `<foo>`:

```
<foo> ::= ...some expansion that has <bar>...
```

To refer to the `<name>` element as a direct child of a `<first_name>` element, you thus write `<first_name>.<name>`. This allows you to express the earlier constraint in a possibly more readable form:

```
$ fandango fuzz -f persons.fan -n 10 -c '<first_name>.<name>.endswith("x")'
```

```
Cmx Diatihxdzteqtbk,008
```

```
Rwx Ukpokjfipkthnlj,41656844096130933088
```

```
Lyjx Dctxahrodon,4171809947
```

```
Dzpbx Kpwn,5749397107716243836
```

```
Kjcxkqjaenzx Mltrmqsdzrnyuchup,67678564671813323326
```

```
Rwx Ukpokjfipkthnlj,41556844096130933088
```

```
Kjcxkqjaenzx Vejnl,67678564671813323326
```

```
Kjcxkqjaenzx Caripayodkakwtob,67678564671813323326
```

```
Dzpbx Mltrmqsdzrnyuchup,5749397107716243836
Kjcxkqjaenzx Kpwn,67678564671813323326
```

Note

You can only access *nonterminal* children this way; `<person_name>." "` (the space in the `<person_name>`) gives an error.

16.2.4 Selecting Descendants

Often, you want to refer to elements in a particular *context* set by the enclosing element. This is why in Fandango, you can also refer to *descendants*.

In XPath, the corresponding operator is `//`.

The expression `<foo>..<bar>` allows accessing elements `<bar>` when they are a *descendant* of a symbol `<foo>`. `<bar>` is a descendant of `<foo>` if

`<foo>..<bar>` includes `<foo>.<bar>`.

- `<bar>` is a child of `<foo>`; or
- one of `<foo>`'s children has `<bar>` as a descendant.

If that sounds like a recursive definition, that is because it is. A simpler way to think about `<foo>..<bar>` may be "All `<bar>`s that occur within `<foo>`".

Let us take a look at some rules in our `persons.fan` example:

```
<first_name> ::= <name>
<last_name> ::= <name>
<name> ::= <ascii_uppercase_letter><ascii_lowercase_letter>+
<ascii_uppercase_letter> ::= "A" | "B" | "C" | ... | "Z"
```

To refer to all <ascii_uppercase_letter> element as descendant of a <first_name> element, you thus write <first_name>..<ascii_uppercase_letter>.

Hence, to make all uppercase letters X, but only as part of a first name, you may write

```
$ fandango fuzz -f persons.fan -n 10 -c '<first_name>..<ascii_uppercase_letter> == "X"
 ↪'
```

```
Xzcjinmkmxkmwahsq Nzasfwdcv,8
Xgrjt Clamwbyayak,47793767945933
Xdepsoocgf Nstungefkphiqmneiipb,84079
```

```
Xnyusqxphkkuhm Csofummvzjeynr,9198074563640612
Xgsvcrntn Cgnjjxagddfldzr,0292886977
Xbsutbufsefokc Fugolsixv,6546
Xsoddufc Cpxmlzeecomjcmnf,4609440868846
```

```
Xuqatnsduaky Phkjkllklhjqjzluklad,12722185932001327
Xwhnwcb Upyghidvuynqusesanq,9243
Xfowwzohcevucq Lgduegelbzhn,7505
```

16.2.5 Chains

You can freely combine [], ., and .. into *chains* that select subtrees. What would the expression <start>[0].<last_name>..<ascii_lowercase_letter> refer to, for instance?

Solution

This is easy:

- <start>[0] is the first element of start, hence a <person_name>.
- .<last_name> refers to the child of type <last_name>
- ..<ascii_lowercase_letter> refers to all descendants of type <ascii_lowercase_letter>

Let's use this in a constraint:

```
$ fandango fuzz -f persons.fan -n 10 -c '<start>[0].<last_name>..<ascii_lowercase_
 ↪letter> == "x"'
```

```
Sodgkiuxgrgxmrpbpsn Pxx,9
Kvypdipqzlrapldyl Yxxxxxxxxxxxxxx,6
```

```
Trjltkofausrwoicoxwvq Cxxxxxxxxxxxxx,2856332196
Hbnkxgsqjxj Xxxxxxxxxxxxxx,31420984
Qqumkhalpymqc Mxxxxxxxx,137284799925
```

```
Ikflhylveafpdqnhbb Vxxxxx,297896639329815377
Yudj Dxxxxxxx,478782
Yfxosihf Wxxxxxxxxxxxxxxxxx,370239791166
```

```
Zbbhc Hxxxxxxxxxxxxxxxxxx,848483366342186
Pdhypffffueyxwp Wxxxxxxxxxxxx,69374
```

16.3 Quantifiers

By default, whenever you use a symbol `<foo>` in a constraint, this constraint applies to *all* occurrences of `<foo>` in the produced output string. For your purposes, though, one such instance already may suffice. This is why Fandango allows expressing *quantification* in constraints.

16.3.1 Star Expressions

In Fandango, you can prefix an element with `*` to obtain a collection of *all* these elements within an individual string. Hence, `*<name>` is a collection of *all* `<name>` elements within the generated string. This syntax can be used in a variety of ways. For instance, we can have a constraint check whether a particular element is in the collection:

Not every constraint that can be *expressed* also can be *solved* by Fandango.

```
"Pablo" in *<name>
```

This constraint evaluates to True if any of the values in `*<name>` (= one of the two `<name>` elements) is equal to `"Pablo"`. `*`-expressions are mostly useful in quantifiers, which we discuss below.

16.3.2 Existential Quantification

To express that within a particular scope, at least one instance of a symbol must satisfy a constraint, use a `*`-expression in combination with `any()`:

```
any(CONSTRAINT for ELEM in *SCOPE)
```

where

- `SCOPE` is a nonterminal (e.g. `<age>`);
- `ELEM` is a Python variable; and
- `CONSTRAINT` is a constraint over `ELEM`

Hence, the expression

```
any(n.startswith("A") for n in *<name>)
```

ensures there is at least *one* element `<name>` that starts with an “A”:

Let us decompose this expression for a moment:

- The expression `for n in *<name>` lets Python iterate over `*<name>` (all `<name>` objects within a person)…
- … and evaluate `n.startswith("A")` for each of them, resulting in a collection of Boolean values.
- The Python function `any(list)` returns `True` if at least one element in `list` is True.

So, what we get is existential quantification:

```
$ fandango fuzz -f persons.fan -n 10 -c 'any(n.startswith("A") for n in *<name>)'
```

```
Vopvltzfnxbseibqo Atplmpayctsq,9667951565671
Ncslqwhwgwpojqsjzu Aurljacaadtlh,629036841238338
Lwafpvszpoyydg Agdazhqpqexfoscyvu,57626889196
Ajlnpmaqmpzohpnzidzvn Frhzmu,073
Aekptk Fzvpgscz,323530892327453048
Fyavd Ayytdfcnlvkjlpnnr,85526377732882269
Ar Uhpgej,46125065202833140000
Ad Zqwizfammmxbhyfp,84565127
Uvbhiymoyaut Ag,5816764
Ad Zqwizfammmxbhyfp,323530892327453048
```

16.3.3 Universal Quantification

Where there are existential quantifiers, there also are *universal* quantifiers. Here we use the Python `all()` function; `all(list)` evaluates to True only if *all* elements in `list` are True.

We use a `*`-expression in combination with `all()`:

```
all(CONSTRAINT for ELEM in *SCOPE)
```

Hence, the expression

```
all(c == "a" for c in *<first_name>..<ascii_lowercase_letter>)
```

ensures that *all* sub-elements `<ascii_lowercase_letter>` in `<first_name>` have the value “a”.

Again, let us decompose this expression:

- The expression `for c in *<first_name>..<ascii_lowercase_letter>` lets Python iterate over all `<ascii_lowercase_letter>` objects within `<first_name>`…
- … and evaluate `c == "a"` for each of them, resulting in a collection of Boolean values.
- The Python function `all(list)` returns `True` if all elements in `list` are True.

So, what we get is universal quantification:

```
$ fandango fuzz -f persons.fan -n 10 -c 'all(c == "a" for c in *<first_name>..<ascii_
↪lowercase_letter>)'
```

```
Uaaaaaaaaaaa Bbsszwnsoaremyuodr,88546
```

```
Faaaaaa Alqcyibgmceqbnngay,67627214275666506
Zaaaaaaaaaaa Ebu,6177420507
Gaaaaaaaaaaaaaaaaa Cjwhkqmj,0173261297
```

```
Kaaaaaaaaaaaaaaaaa Fonejuwbwyblt,31359
Qaaaaaaaaaaa Acfw,30803021836
Faaaaaa Nnk,9
```

```
Gaaaaaaaaaaaaaaaaa Oldzmel,3841929650
Laaaaaaaaaaa Lkpzprxe,61630587
Laa Uupseuicbvonxjm,812
```

By default, all symbols are universally quantified within `<start>`, so a dedicated universal quantifier is only needed if you want to *limit the scope* to some sub-element. This is what we do here with `<first_name>..<ascii_lowercase_letter>`, limiting the scope to `<first_name>`.

Given the default universal quantification, you can actually achieve the same effect as above without `all()` and without a `*`. How?

Solution

You can access all `<ascii_lowercase_letter>` elements within `<first_name>` directly:

```
$ fandango fuzz -f persons.fan -n 10 -c '<first_name>..<ascii_lowercase_letter> ==
 ↪"a"'
```

16.3.4 Old-Style Quantifiers

Prior to version 1.0, Fandango supported another style of quantifiers:

- The expression `forall SYMBOL in EXPRESSION: CONSTRAINT` is equivalent to `all(CONSTRAINT for ELEM in *EXPRESSION: )`
- The expression `exists SYMBOL in EXPRESSION: CONSTRAINT` is equivalent to `any(CONSTRAINT for ELEM in *EXPRESSION: )`

with `SYMBOL` being a symbol (in `<...>`, so not a Python object) which can be referred to in `CONSTRAINT`.

To express that at least one letter in `<first_name>` should be `'a'`, write

```
exists <c> in <first_name>: str(<c>) == 'a'
```

```
$ fandango fuzz -f persons.fan -n 5 -c 'exists <c> in <ascii_lowercase_letter>: str(
 ↪<c>) == "a"'
```

```
Jrravt Sswejzwoikila,50108909348
Jaraaa Saaaaawaaaiaa,50108909348
Azrsjdccnuscj Wmadlhqsiybix,97003417348677
Aaaaaaacaascj Waadaaasaaaia,97003417348677
```

```
Dogaoqjrinnho Axburir,7345201
```

```
$ fandango fuzz -f persons.fan -n 5 -c 'forall <c> in <ascii_lowercase_letter>: str(
↪<c>) == "a"'
```

```
Ta Iaaaaaaaaaa,8260400713333793
Oaaa Raa,4856801198
```

```
Baa La,11128252
Iaaaaaa Taa,8
```

```
Eaaaa Uaa,00193169827297205722
```

Deprecated since version 1.0: Old-style quantifiers will be deprecated in a future Fandango version. Use *all() and any()* (page 98) instead.

CHAPTER

SEVENTEEN

CONVERTING ANTLR AND OTHER INPUT SPECS

Often, you may already have an input format specification available, but not (yet) in Fandango `.fan` format. Fandango's `convert` command allows you automatically translate common input specifications into `.fan` files - at least most of it.

For instance, *executable code* in ANTLR and 010 specs can only be partially converted at best.

Important

All these converters are *lossy* - that is, some features of the original specifications may not be converted into Fandango. Hence, the idea is that you use converted formats as a base for further manual editing and checking.

Note

All these formats define the *syntax* of input files, typically for the purpose of *parsing*. To *produce* inputs that are also *semantically* valid, you will often have to augment the `.fan` files with *constraints* (page 45) to make them semantically valid, too.

Under Construction

All these converters are experimental at this point.

17.1 Converting ANTLR Specs

Fandango allows you to automatically convert ANTLR grammar specifications (`.g4`, `.antlr`) into Fandango `.fan` files. ANTLR is a very popular parser generator; a wide large collections of ANTLR grammars[41] is available.

Simply use the command `fandango convert`, followed by the ANTLR file to be converted.

As an example, consider this simple `Calculator.g4` ANTLR file:

[41] https://github.com/antlr/grammars-v4

```
// https://www.inovex.de/de/blog/building-a-simple-calculator-with-antlr-in-python/

grammar Calculator;

expression
    : NUMBER                          # Number
    | '(' expression ')'              # Parentheses
    | expression TIMES expression     # Multiplication
    | expression DIV expression       # Division
    | expression PLUS expression      # Addition
    | expression MINUS expression     # Subtraction
;

PLUS : '+';
MINUS: '-';
TIMES: '*';
DIV  : '/';
NUMBER : [0-9]+;
WS : [ \r\n\t]+ -> skip;
```

Invoking `fandango convert` produces an (almost) equivalent Fandango `.fan` file:

The `fandango convert` command determines the spec language from the file extension:

- `.antlr` and `.g4` - ANTLR
- `.bt` and `.010` - 010 Binary Templates
- `.dtd` - DTDs
- `.fan` - `.fan` files

If your file has another file name, you can use the `--from` option (say, `--from=antlr`) to specify a specific format.

```
$ fandango convert Calculator.g4
```

```
# Automatically generated from '../src/fandango/converters/antlr/Calculator.g4'.
#
# Calculator
<expression> ::= <NUMBER> | '(' <expression> ')' | <expression> <TIMES>
 ↪<expression> | <expression> <DIV> <expression> | <expression> <PLUS> <expression>
 ↪ | <expression> <MINUS> <expression>
<PLUS> ::= '+'
<MINUS> ::= '-'
<TIMES> ::= '*'
<DIV> ::= '/'
<NUMBER> ::= r'[0-9]'+
<WS> ::= r'[ \r\n\t]'+  # NOTE: was '-> skip'
```

Note the `NOTE` comment at the bottom: The ANTLR lexer action `skip` has no equivalent in Fandango; hence `WS` elements will neither be skipped nor generated.

Still, we can use this grammar to produce expressions. Note the usage of the `-o` option to specify an output file and the `--start` option to specify a start symbol.

```
$ fandango convert -o Calculator.fan Calculator.g4
$ fandango fuzz -f Calculator.fan --start='<expression>' -n 10
```

```
fandango:WARNING: Symbol <WS> defined, but not used
```

```
88501523312718*(0142127872104-(2064554742803)*08106307820343+37-3/1-285/7104/258-
↪1*(360)-240-11703)+4508/0*6
```

```
848944382328882-(((444991+5156783161/64702770219236+9371905+373-3+(960)+92982/
↪239*4)*(0098)-4043/92911/3965))*601849
4390191989/(379080033-((((018072*7-0*12*2182*0319*0653)-61219)*89881)+05423*5726-
↪9302/5018/6824+3417))
```

```
(29906172229-679774669*033334715600+(2652)+805/6-8+77/(08353247)*1071*8*2557967-
↪5626/7498+((090))*6-(2954))-(374)
9
```

```
(((679874441501368)/39933462-377697301-7-(8368)*741555-1450+9499*2223+13)-643248-
↪5+4490*3510561-486)*320680+0996-6731
((776072+((((42382-79668108479258*9160169-98828815059139*838672+4173))))-(609-0)/
↪366+9365*702304996+730*1/(535)-67589))
```

```
(99588115008542715*(272218255556*(0985-7/9-06+4957)*41411/3)*(753)-5-(7435)+74583/
↪5483/5715-(3))/6519/0+(865)
```

```
404241*3613126911888195175/7257977006510357287/
↪34*814282934603060 94*1666045384764405+650-4-1871/5-0/70*60/6*(8373)/(666078438)
(385800726372111+15275047894+8830793598-3657093+(4273/0)*3-(517242*4)-(1278)-
↪((45225))*521489*623*5)*91079956-9414*2436
```

Note

Most features of ANTLR that cannot be represented in Fandango will be marked by `NOTE` comments. These include

- Actions
- Modifiers
- Clauses such as `return` or `throws`
- Exceptions
- Predicate options
- Element options
- Negations (~) over complex expressions

17.2 Converting 010 Binary Templates

Fandango provides some basic support for converting Binary Templates (`.bt`, `.010`) for the 010 Editor. A large collection of binary templates for various binary formats[42] is available.

Again, simply use the command `fandango convert`, followed by the binary template file to be converted.

Our *GIF example* (page 151) is automatically created from a GIF binary template.

> **Note**
>
> 010 Binary Templates can contain *arbitrary code* that will be executed during parsing. Fandango will recognize a number of common patterns; features that will require manual work include
>
> - Checksums
> - Complex length encodings

> **Note**
>
> The `fandango convert` command provides two options to specify *bit orderings*, should the `.bt` file not already do so.
>
> - `--endianness=(little|big)` and
> - `--bitfield-order=(left-to-right|right-to-left)`

17.3 Converting DTDs

A Document Type Definition (DTD, `.dtd`) specifies the format of an XML file. Fandango can convert these into `.fan` files, enabling the production of XML files that conform to the DTD.

Again, simply use the command `fandango convert`, followed by the binary template file to be converted.

> **Note**
>
> As with Binary Templates, Fandango will recognize a number of common patterns, but not all.

In the generated `.fan` file, you can customize every single element in its context. As an example, consider this `svg11.fan` file which specializes individual elements of a `svg.fan` file generated from an SVG DTD. The DTD by itself does not specify types of individual fields, so we do this here:

If you find that this is long, consider that SVG is actually a very complex format.

[42] https://www.sweetscape.com/010editor/repository/templates/

```
include('svg.fan')

# Add standard blurb at top
<start> ::= ('<?xml version="1.0" standalone="no"?>'
'<!DOCTYPE svg>' <svg>)

<svg> ::= ('<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/
 ↪1999/xlink"'
' width=' <svg_width_value>
' height=' <svg_height_value>
' baseProfile="full" viewBox=' <svg_viewBox_value> '>'
(<desc> | <title> | <metadata> | <animate> | <set> | <animateMotion> |
 ↪<animateColor> | <animateTransform> | <svg> | <g> | <defs> | <symbol> | <use> |
 ↪<switch> | <image> | <style> | <path> | <rect> | <circle> | <line> | <ellipse> |
 ↪<polyline> | <polygon> | <text> | <altGlyphDef> | <marker> | <color_profile> |
 ↪<linearGradient> | <radialGradient> | <pattern> | <clipPath> | <mask> | <filter>␣
 ↪| <cursor> | <a> | <view> | <script> | <font> | <font_face> | <foreignObject>)
 ↪{10} '</svg>')

# Standard data types
<cdata> ::= <qint> | <string>
<qnat> ::= <q> <nat> <q>
<nat> ::= r'[1-9]' <digit>* | '0'
<qint> ::= <q> <int> <q>
<int> ::= r'[1-9]' <digit>* | '-' r'[1-9]' <digit>* | '0'
<string> ::= '"' <char>* '"' | "'" <char>* "'"
<char> ::= r'[0-9a-zA-Z_-]+'
<id> ::= <q> <ascii_letter> (<ascii_letter> | <digit> | '_')* <q>
<nmtoken> ::= <id>
<pcdata> ::= <cdata>
<url> ::= <q> 'https://cispa.de' <q>
<qpercentage> ::= <q> <percentage> <q>
<percentage> ::= ("0" | r"[1-9][0-9]?" | "100")

# SVG-specific data types
<Coordinate_datatype> ::= <qint> := "'100'"
<Length_datatype> ::= <qnat>
<FontFamilyValue_datatype> ::= <string> := '"sans-serif"'
<FontSizeValue_datatype> ::= <qnat> := "'12'"
<FontSizeAdjustValue_datatype> ::= <qnat> := "'0'"
<GlyphOrientationHorizontalValue_datatype> ::= <qint> := "'0'"
<GlyphOrientationVerticalValue_datatype> ::= <qint> := "'0'"
<Number_datatype> ::= <qint>
<NumberOptionalNumber_datatype> ::= <qint>
<OpacityValue_datatype> ::= <qpercentage> := "'100'"
<PathData_datatype> ::= <q> (<int> <ws>)+ <q>
<Text_datatype> ::= <string>
<Script_datatype> ::= <string>
<SVGColor_datatype> ::= <q> '#' (<hexdigit>{3} | <hexdigit>{6}) <q>

# Mappings of attributes to data types
<accent_height_value> ::= <Number_datatype>
<alphabetic_value> ::= <Number_datatype>
<amplitude_value> ::= <Number_datatype>
<arabic_form_value> ::= <cdata>
<arcrole_value> ::= <cdata>
<ascent_value> ::= <Number_datatype>
```

(continues on next page)

(continued from previous page)

```
<attributeName_value> ::= <cdata>
<attributeType_value> ::= <cdata>
<azimuth_value> ::= <Number_datatype>
<baseFrequency_value> ::= <NumberOptionalNumber_datatype>
<baseProfile_value> ::= <Text_datatype>
<base_value> ::= <cdata>
<baseline_shift_value> ::= <cdata>
<bbox_value> ::= <cdata>
<begin_value> ::= <cdata>
<bias_value> ::= <Number_datatype>
<by_value> ::= <cdata>
<cap_height_value> ::= <Number_datatype>
<class_value> ::= <cdata>
<clip_path_value> ::= <cdata>
<clip_value> ::= <cdata>
<color_profile_value> ::= <cdata>
<color_value> ::= <cdata>
<contentScriptType_value> ::= <cdata>
<contentStyleType_value> ::= <cdata>
<cursor_value> ::= <cdata>
<cx_value> ::= <Coordinate_datatype>
<cy_value> ::= <Coordinate_datatype>
<d_value> ::= <PathData_datatype>
<descent_value> ::= <Number_datatype>
<diffuseConstant_value> ::= <Number_datatype>
<divisor_value> ::= <Number_datatype>
<dur_value> ::= <cdata>
<dx_value> ::= <Number_datatype>
<dy_value> ::= <Number_datatype>
<elevation_value> ::= <Number_datatype>
<enable_background_value> ::= <cdata>
<end_value> ::= <cdata>
<exponent_value> ::= <Number_datatype>
<fePointLight_z_value> ::= <Number_datatype>
<fePointLight_y_value> ::= <Number_datatype>
<fePointLight_x_value> ::= <Number_datatype>
<feSpotLight_z_value> ::= <Number_datatype>
<feSpotLight_y_value> ::= <Number_datatype>
<feSpotLight_x_value> ::= <Number_datatype>
<fill_opacity_value> ::= <OpacityValue_datatype>
<fill_value> ::= <cdata>
<filterRes_value> ::= <NumberOptionalNumber_datatype>
<filter_value> ::= <cdata>
<flood_color_value> ::= <SVGColor_datatype>
<flood_opacity_value> ::= <OpacityValue_datatype>
<font_family_value> ::= <FontFamilyValue_datatype>
<font_size_adjust_value> ::= <FontSizeAdjustValue_datatype>
<font_size_value> ::= <FontSizeValue_datatype>
<font_stretch_value> ::= <cdata>
<font_style_value> ::= <cdata>
<font_variant_value> ::= <cdata>
<font_weight_value> ::= <cdata>
<format_value> ::= <cdata>
<from_value> ::= <cdata>
<fx_value> ::= <Coordinate_datatype>
<fy_value> ::= <Coordinate_datatype>
<g1_value> ::= <cdata>
```

(continues on next page)

(continued from previous page)

```
<g2_value> ::= <cdata>
<glyphRef_value> ::= <cdata>
<glyph_name_value> ::= <cdata>
<glyph_orientation_horizontal_value> ::= <GlyphOrientationHorizontalValue_datatype>
<glyph_orientation_vertical_value> ::= <GlyphOrientationVerticalValue_datatype>
<gradientTransform_value> ::= <cdata>
<hanging_value> ::= <Number_datatype>
<height_value> ::= <Number_datatype>
<horiz_adv_x_value> ::= <Number_datatype>
<horiz_origin_x_value> ::= <Number_datatype>
<horiz_origin_y_value> ::= <Number_datatype>
<href_value> ::= <url>
<id_value> ::= <id>
<ideographic_value> ::= <Number_datatype>
<in2_value> ::= <cdata>
<in_value> ::= <cdata>
<intercept_value> ::= <Number_datatype>
<k1_value> ::= <Number_datatype>
<k2_value> ::= <Number_datatype>
<k3_value> ::= <Number_datatype>
<k4_value> ::= <Number_datatype>
<k_value> ::= <Number_datatype>
<kernelMatrix_value> ::= <cdata>
<kernelUnitLength_value> ::= <NumberOptionalNumber_datatype>
<kerning_value> ::= <cdata>
<keyPoints_value> ::= <cdata>
<keySplines_value> ::= <cdata>
<keyTimes_value> ::= <cdata>
<lang_value> ::= <nmtoken>
<letter_spacing_value> ::= <cdata>
<lighting_color_value> ::= <SVGColor_datatype>
<limitingConeAngle_value> ::= <Number_datatype>
<local_value> ::= <cdata>
<markerHeight_value> ::= <Length_datatype>
<markerWidth_value> ::= <Length_datatype>
<marker_end_value> ::= <cdata>
<marker_mid_value> ::= <cdata>
<marker_start_value> ::= <cdata>
<mask_value> ::= <cdata>
<mathematical_value> ::= <Number_datatype>
<max_value> ::= <cdata>
<media_value> ::= <cdata>
<min_value> ::= <cdata>
<name_value> ::= <cdata>
<numOctaves_value> ::= <cdata>
<offset_value> ::= <Number_datatype>
<onabort_value> ::= <Script_datatype>
<onactivate_value> ::= <Script_datatype>
<onbegin_value> ::= <Script_datatype>
<onclick_value> ::= <Script_datatype>
<onend_value> ::= <Script_datatype>
<onerror_value> ::= <Script_datatype>
<onfocusin_value> ::= <Script_datatype>
<onfocusout_value> ::= <Script_datatype>
<onload_value> ::= <Script_datatype>
<onmousedown_value> ::= <Script_datatype>
<onmousemove_value> ::= <Script_datatype>
```

(continues on next page)

(continued from previous page)

```
<onmouseout_value> ::= <Script_datatype>
<onmouseover_value> ::= <Script_datatype>
<onmouseup_value> ::= <Script_datatype>
<onrepeat_value> ::= <Script_datatype>
<onresize_value> ::= <Script_datatype>
<onscroll_value> ::= <Script_datatype>
<onunload_value> ::= <Script_datatype>
<onzoom_value> ::= <Script_datatype>
<opacity_value> ::= <OpacityValue_datatype>
<order_value> ::= <Number_datatype>
<orient_value> ::= <cdata>
<orientation_value> ::= <cdata>
<origin_value> ::= <cdata>
<overline_position_value> ::= <Number_datatype>
<overline_thickness_value> ::= <Number_datatype>
<panose_1_value> ::= <cdata>
<pathLength_value> ::= <Number_datatype>
<path_value> ::= <cdata>
<patternTransform_value> ::= <cdata>
<pointsAtX_value> ::= <Number_datatype>
<pointsAtY_value> ::= <Number_datatype>
<pointsAtZ_value> ::= <Number_datatype>
<points_value> ::= <cdata>
<preserveAspectRatio_value> ::= <cdata>
<r_value> ::= <Length_datatype>
<radius_value> ::= <Number_datatype>
<refX_value> ::= <cdata>
<refY_value> ::= <cdata>
<repeatCount_value> ::= <cdata>
<repeatDur_value> ::= <cdata>
<requiredExtensions_value> ::= <cdata>
<requiredFeatures_value> ::= <cdata>
<result_value> ::= <cdata>
<role_value> ::= <cdata>
<rotate_value> ::= <cdata>
<rx_value> ::= <Length_datatype>
<ry_value> ::= <Length_datatype>
<scale_value> ::= <Number_datatype>
<seed_value> ::= <Number_datatype>
<slope_value> ::= <Number_datatype>
<specularConstant_value> ::= <Number_datatype>
<specularExponent_value> ::= <Number_datatype>
<startOffset_value> ::= <Length_datatype>
<stdDeviation_value> ::= <NumberOptionalNumber_datatype>
<stemh_value> ::= <Number_datatype>
<stemv_value> ::= <Number_datatype>
<stop_color_value> ::= <SVGColor_datatype>
<stop_opacity_value> ::= <OpacityValue_datatype>
<strikethrough_position_value> ::= <Number_datatype>
<strikethrough_thickness_value> ::= <Number_datatype>
<string_value> ::= <cdata>
<stroke_dasharray_value> ::= <cdata>
<stroke_dashoffset_value> ::= <cdata>
<stroke_miterlimit_value> ::= <cdata>
<stroke_opacity_value> ::= <OpacityValue_datatype>
<stroke_value> ::= <cdata>
<stroke_width_value> ::= <Number_datatype>
```

(continues on next page)

(continued from previous page)

```
<style_value> ::= <cdata>
<surfaceScale_value> ::= <Number_datatype>
<systemLanguage_value> ::= <cdata>
<tableValues_value> ::= <cdata>
<targetX_value> ::= <cdata>
<targetY_value> ::= <cdata>
<target_value> ::= <nmtoken>
<textLength_value> ::= <Length_datatype>
<text_decoration_value> ::= <cdata>
<title_value> ::= <Text_datatype>
<to_value> ::= <cdata>
<transform_value> ::= <cdata>
<type_value> ::= <cdata>
<u1_value> ::= <cdata>
<u2_value> ::= <cdata>
<underline_position_value> ::= <Number_datatype>
<underline_thickness_value> ::= <Number_datatype>
<unicode_range_value> ::= <cdata>
<unicode_value> ::= <cdata>
<units_per_em_value> ::= <Number_datatype>
<v_alphabetic_value> ::= <Number_datatype>
<v_hanging_value> ::= <Number_datatype>
<v_ideographic_value> ::= <Number_datatype>
<v_mathematical_value> ::= <Number_datatype>
<values_value> ::= <cdata>
<vert_adv_y_value> ::= <Number_datatype>
<vert_origin_x_value> ::= <Number_datatype>
<vert_origin_y_value> ::= <Number_datatype>
<viewBox_value> ::= <q> <int> <ws> <int> <ws> <int> <ws> <int> <q>
<viewTarget_value> ::= <cdata>
<width_value> ::= <Number_datatype>
<widths_value> ::= <cdata>
<word_spacing_value> ::= <cdata>
<x1_value> ::= <Coordinate_datatype>
<x2_value> ::= <Coordinate_datatype>
<x_height_value> ::= <Number_datatype>
<x_value> ::= <Coordinate_datatype>
<y1_value> ::= <Coordinate_datatype>
<y2_value> ::= <Coordinate_datatype>
<y_value> ::= <Coordinate_datatype>
<z_value> ::= <Coordinate_datatype>
```

Once this is all set, we can use this to test SVGs with extreme values, as in this `svgextreme.fan` example:

```
include('svg11.fan')

# where (int(<width_value>) > 1e8 or int(<height_value>) > 1e8)

# Check with extreme number values
where <Number_datatype> == "'1000000'"

# Ensure we have a minimum of children
where len(<svg>) > 20
```

17.4 Converting `.fan` files

With `fandango convert`, you can also "convert" `.fan` files. This results in a "normalized" format, where all comments and blank lines have been removed. If we send this input to `fandango convert`:

```
# A fine file to produce person names
from faker import Faker
fake = Faker()

include('persons.fan')

<first_name> ::= <name> := fake.first_name()
<last_name> ::= <name> := fake.last_name()
```

then we get

```
$ fandango convert persons-faker.fan
```

```
# Automatically generated from 'persons-faker.fan'.
#
from faker import Faker
fake = Faker()
include('persons.fan')

<first_name> ::= <name> := fake.first_name()
<last_name> ::= <name> := fake.last_name()
```

Note

This feature can be useful to detect semantic changes in `.fan` files.

CHAPTER

EIGHTEEN

COVERING SPECS AND CODE

During input generation, Fandango attempts to *cover* as many behaviors as possible. There are three ways to do so:

- **Grammar coverage** - that is, covering *alternatives in the grammar*;
- **Code coverage** - that is, covering *alternatives in the program under test*; and
- **Constraint coverage** - that is, covering *alternatives in constraints*.

Fandango is set to satisfy all these goals in unison.

18.1 Grammar Coverage

During *input generation*, Fandango favors individual inputs that have a high *grammar coverage* - that is, they cover as many alternatives as possible. It does so after construction, as part of evaluating the fitness of individual inputs.

During protocol testing (page 233), Fandango already produces such interactions *by construction* – that is, with every new iteration, it tries to cover parts of the grammar (= states, messages, and transitions) that have not been seen yet.

Grammar coverage is determined and achieved using the k-path metric by Havrikov and Zeller [HZ20].

Added in version 1.2: For version 1.2, during input generation, Fandango will also achieve grammar coverage by construction.

18.2 Code Coverage

As of version 1.2, Fandango will provide experimental support for guidance by code coverage. Details will be added here.

Added in version 1.2: This feature is planned for Fandango 1.2.

18.3 Constraint Coverage

Future versions of Fandango will aim to achieve diversity by fulfilling different alternatives in constraints. Once this is complete, details will be added here.

Added in version 1.x: This feature is planned for a future Fandango version.

CHAPTER

NINETEEN

CASE STUDY: ISO 8601 DATE + TIME

Let us make use of what we have explored so far in a more elaborate case study. ISO 8601[43] is an international standard for exchange and communication of *date and time-related data*, providing an unambiguous method of representing calendar dates and times.

In this chapter, we will define a full Fandango spec for ISO 8601 date and time formats, ensuring both syntactic and semantic validity. To make matters more interesting, we will create the spec *programmatically* - that is, by having a number of Python functions that generate the spec for us.

19.1 Creating Grammars Programmatically

We start with a number of functions that help us create fragments of the grammar.

`make_rule()` creates a grammar rule from `symbol` and its possible `expansions`:

```
import sys

def print_s(s: str) -> str:
    print(s, end="")
    return s

def make_rule(symbol: str, expansions: list[str], sep: str = '') -> str:
    return print_s(f"{sep}<{symbol}> ::= " + " | ".join(expansions) + "\n")

make_rule("start", ["<iso8601datetime>"]);  # a final ";" suppresses result
```

```
<start> ::= <iso8601datetime>
```

We can also use `make_rule()` to quickly create a list of expansions:

```
make_rule("digit", [f"'{digit}'" for digit in range(0, 10)]);
```

```
<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

These additional functions help in adding non-grammar elements such as headers, code, and constraints:

```
def make_header(title: str) -> str:
    return print_s(f"\n# {title}\n")

def make_comment(comment: str, sep: str = '') -> str:
```

(continues on next page)

[43] https://en.wikipedia.org/wiki/ISO_8601

(continued from previous page)

```
    return print_s(f"{sep}# {comment}\n")

def make_constraint(constraint: str) -> str:
    return print_s(f"where {constraint}\n")

def make_code(code: str) -> str:
    return print_s(f"{code}\n")

make_header("ISO 8601 grammar");
```

```
# ISO 8601 grammar
```

19.2 Spec Header

Let us create a Fandango spec. For now, we store the final spec in the `iso8601lib` string variable, to be saved in a file at the end. Every time we add a new fragment, the new fragment will be output here, so we can see the actual content of `iso8601lib` incrementally.

```
iso8601lib = make_header("ISO 8601 date and time")
iso8601lib += make_comment("Generated by docs/ISO8601.md. Do not edit.")
```

```
# ISO 8601 date and time
# Generated by docs/ISO8601.md. Do not edit.
```

We will need the `datetime` library below, so we import it.

```
iso8601lib += make_code("\nimport datetime\n")
```

```
import datetime
```

19.3 Date

We start with a `<start>` symbol. An ISO 8601 date/time spec starts with a date, and an optional time, separated by `T`:

```
iso8601lib += make_rule("start", ["<iso8601datetime>"])

iso8601lib += make_rule("iso8601datetime",
                    ["<iso8601date> ('T' <iso8601time>)?"])
```

```
<start> ::= <iso8601datetime>
<iso8601datetime> ::= <iso8601date> ('T' <iso8601time>)?
```

A date can either be a *calendar date*, but also a *week date* or an *ordinal date*.

```
iso8601lib += make_rule("iso8601date",
                        ["<iso8601calendardate>",
                         "<iso8601weekdate>",
                         "<iso8601ordinaldate>"], sep='\n')
```

```
<iso8601date> ::= <iso8601calendardate> | <iso8601weekdate> | <iso8601ordinaldate>
```

19.3.1 Calendar Dates

An ISO 8601 calendar date has the format YYYY-MM-DD, but can also be YYYY-MM or YYYYMMDD. The year can be prefixed with + or − to indicate AC/BC[44] (or CE/BCE[45]) designators.

```
iso8601lib += make_rule("iso8601calendardate",
                    ["<iso8601year> '-' <iso8601month> ('-' <iso8601day>)?",
                     "<iso8601year> <iso8601month> <iso8601day>"], sep='\n')
iso8601lib += make_rule("iso8601year", ["('+'|'-')? <digit>{4}"])
```

```
<iso8601calendardate> ::= <iso8601year> '-' <iso8601month> ('-' <iso8601day>)? |
 ↪<iso8601year> <iso8601month> <iso8601day>
<iso8601year> ::= ('+'|'-')? <digit>{4}
```

19.3.2 Months

Months are easy:

```
iso8601lib += make_rule("iso8601month",
                    [f"'{month:02d}'" for month in range(1, 13)])
```

```
<iso8601month> ::= '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' | '09' |
 ↪'10' | '11' | '12'
```

19.3.3 Days

As with months, we define each day individually to avoid biasing the grammar. The fact that some months have only 28, 29, or 30 days will be handled later on in a constraint.

```
iso8601lib += make_rule("iso8601day",
                    [f"'{day:02d}'" for day in range(1, 32)])
```

```
<iso8601day> ::= '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' | '09' | '10
 ↪' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20' | '21' |
 ↪'22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31'
```

Tip

For testing purposes, it makes sense to test with *extreme* values - in our case, January 1, February 28 or 29, and December 31.

[44] https://en.wikipedia.org/wiki/Anno_Domini
[45] https://en.wikipedia.org/wiki/Common_Era

19.3.4 Week Dates

ISO 8601 also allows specifying dates by week numbers (1 - 53) and optional days of the week (1 - 7). `2025-W12-1` is the Monday of the 12th week in 2025.

```
iso8601lib += make_rule("iso8601weekdate",
                    ["<iso8601year> '-'? 'W' <iso8601week> ('-' <iso8601weekday>)?"],␣
 ↪sep='\n')
iso8601lib += make_rule("iso8601week",
                    [f"'{week:02d}'" for week in range(1, 54)])
iso8601lib += make_rule("iso8601weekday",
                    [f"'{weekday:1d}'" for weekday in range(1, 8)])
```

```
<iso8601weekdate> ::= <iso8601year> '-'? 'W' <iso8601week> ('-' <iso8601weekday>)?
<iso8601week> ::= '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' | '09' |
 ↪'10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20' | '21
 ↪' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31' | '32' |
 ↪'33' | '34' | '35' | '36' | '37' | '38' | '39' | '40' | '41' | '42' | '43' | '44
 ↪' | '45' | '46' | '47' | '48' | '49' | '50' | '51' | '52' | '53'
<iso8601weekday> ::= '1' | '2' | '3' | '4' | '5' | '6' | '7'
```

19.3.5 Ordinal Dates

ISO 8601 also allows specifying dates by day numbers (1 - 366). `2000-366`, for instance, specifies the last day in 2000 (which was a leap year). Again, we specify each day programmatically (and individually).

```
iso8601lib += make_rule("iso8601ordinaldate",
                    ["<iso8601year> ('-'? <iso8601ordinalday>)?"], sep='\n')
iso8601lib += make_rule("iso8601ordinalday",
                    [f"'{day:03d}'" for day in range(1, 367)])
```

```
<iso8601ordinaldate> ::= <iso8601year> ('-'? <iso8601ordinalday>)?
<iso8601ordinalday> ::= '001' | '002' | '003' | '004' | '005' | '006' | '007' |
 ↪'008' | '009' | '010' | '011' | '012' | '013' | '014' | '015' | '016' | '017' |
 ↪'018' | '019' | '020' | '021' | '022' | '023' | '024' | '025' | '026' | '027' |
 ↪'028' | '029' | '030' | '031' | '032' | '033' | '034' | '035' | '036' | '037' |
 ↪'038' | '039' | '040' | '041' | '042' | '043' | '044' | '045' | '046' | '047' |
 ↪'048' | '049' | '050' | '051' | '052' | '053' | '054' | '055' | '056' | '057' |
 ↪'058' | '059' | '060' | '061' | '062' | '063' | '064' | '065' | '066' | '067' |
 ↪'068' | '069' | '070' | '071' | '072' | '073' | '074' | '075' | '076' | '077' |
 ↪'078' | '079' | '080' | '081' | '082' | '083' | '084' | '085' | '086' | '087' |
 ↪'088' | '089' | '090' | '091' | '092' | '093' | '094' | '095' | '096' | '097' |
 ↪'098' | '099' | '100' | '101' | '102' | '103' | '104' | '105' | '106' | '107' |
 ↪'108' | '109' | '110' | '111' | '112' | '113' | '114' | '115' | '116' | '117' |
 ↪'118' | '119' | '120' | '121' | '122' | '123' | '124' | '125' | '126' | '127' |
 ↪'128' | '129' | '130' | '131' | '132' | '133' | '134' | '135' | '136' | '137' |
 ↪'138' | '139' | '140' | '141' | '142' | '143' | '144' | '145' | '146' | '147' |
 ↪'148' | '149' | '150' | '151' | '152' | '153' | '154' | '155' | '156' | '157' |
 ↪'158' | '159' | '160' | '161' | '162' | '163' | '164' | '165' | '166' | '167' |
 ↪'168' | '169' | '170' | '171' | '172' | '173' | '174' | '175' | '176' | '177' |
 ↪'178' | '179' | '180' | '181' | '182' | '183' | '184' | '185' | '186' | '187' |
 ↪'188' | '189' | '190' | '191' | '192' | '193' | '194' | '195' | '196' | '197' |
 ↪'198' | '199' | '200' | '201' | '202' | '203' | '204' | '205' | '206' | '207' |
 ↪'208' | '209' | '210' | '211' | '212' | '213' | '214' | '215' | '216' | '217' |
 ↪'218' | '219' | '220' | '221' | '222' | '223' | '224' | '225' | '226' | '227' |
```

(continues on next page)

(continued from previous page)

```
↪'228' | '229' | '230' | '231' | '232' | '233' | '234' | '235' | '236' | '237' |
↪'238' | '239' | '240' | '241' | '242' | '243' | '244' | '245' | '246' | '247' |
↪'248' | '249' | '250' | '251' | '252' | '253' | '254' | '255' | '256' | '257' |
↪'258' | '259' | '260' | '261' | '262' | '263' | '264' | '265' | '266' | '267' |
↪'268' | '269' | '270' | '271' | '272' | '273' | '274' | '275' | '276' | '277' |
↪'278' | '279' | '280' | '281' | '282' | '283' | '284' | '285' | '286' | '287' |
↪'288' | '289' | '290' | '291' | '292' | '293' | '294' | '295' | '296' | '297' |
↪'298' | '299' | '300' | '301' | '302' | '303' | '304' | '305' | '306' | '307' |
↪'308' | '309' | '310' | '311' | '312' | '313' | '314' | '315' | '316' | '317' |
↪'318' | '319' | '320' | '321' | '322' | '323' | '324' | '325' | '326' | '327' |
↪'328' | '329' | '330' | '331' | '332' | '333' | '334' | '335' | '336' | '337' |
↪'338' | '339' | '340' | '341' | '342' | '343' | '344' | '345' | '346' | '347' |
↪'348' | '349' | '350' | '351' | '352' | '353' | '354' | '355' | '356' | '357' |
↪'358' | '359' | '360' | '361' | '362' | '363' | '364' | '365' | '366'
```

Tip

For testing purposes, it makes sense to test with *extreme* values - in our case, `1`, `365` and `366`.

19.4 Time

Now for *time* specifications. In ISO 8601, time is specified as `HH:MM:SS`, where `HH` comes in 24-hour format. `MM` and `SS` are optional, as is the colon separator in an "unambiguous" context.

```
iso8601lib += make_rule("iso8601time",
                    ["'T'? <iso8601hour> (':'? <iso8601minute> (':'? <iso8601second>␣
↪(('.' | ',') <iso8601fraction>)? )? )? <iso8601timezone>?"], sep='\n')
```

```
<iso8601time> ::= 'T'? <iso8601hour> (':'? <iso8601minute> (':'? <iso8601second> ((
↪'.' | ',') <iso8601fraction>)? )? )? <iso8601timezone>?
```

19.4.1 Hours

Hours normally go from `00` to `23`, but `24:00:00` is allowed to represent midnight at the end of a day.

```
iso8601lib += make_comment("24:00:00 is allowed to represent midnight at the end of a␣
↪day", sep='\n')
iso8601lib += make_rule("iso8601hour",
                    [f"'{hour:02d}'" for hour in range(0, 25)])
```

```
# 24:00:00 is allowed to represent midnight at the end of a day
<iso8601hour> ::= '00' | '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' |
↪'09' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20
↪' | '21' | '22' | '23' | '24'
```

19.4.2 Minutes

Minutes go from `00` to `59`. Nothing special here.

```
iso8601lib += make_rule("iso8601minute",
                        [f"'{minute:02d}'" for minute in range(0, 60)])
```

```
<iso8601minute> ::= '00' | '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' |
 ↪'09' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20
 ↪' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31' |
 ↪'32' | '33' | '34' | '35' | '36' | '37' | '38' | '39' | '40' | '41' | '42' | '43
 ↪' | '44' | '45' | '46' | '47' | '48' | '49' | '50' | '51' | '52' | '53' | '54' |
 ↪'55' | '56' | '57' | '58' | '59'
```

19.4.3 Seconds

Seconds normally go from `00` to `59`, but `60` can be used to represent leap seconds.

```
iso8601lib += make_comment("xx:yy:60 is allowed to represent leap seconds", sep='\n')
iso8601lib += make_rule("iso8601second",
                        [f"'{second:02d}'" for second in range(0, 61)])
```

```
# xx:yy:60 is allowed to represent leap seconds
<iso8601second> ::= '00' | '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' |
 ↪'09' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20
 ↪' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31' |
 ↪'32' | '33' | '34' | '35' | '36' | '37' | '38' | '39' | '40' | '41' | '42' | '43
 ↪' | '44' | '45' | '46' | '47' | '48' | '49' | '50' | '51' | '52' | '53' | '54' |
 ↪'55' | '56' | '57' | '58' | '59' | '60'
```

Seconds can be followed by `,` or a `.` (see `<iso8601time>`, above) and a fraction - an arbitrary number of digits.

```
iso8601lib += make_rule("iso8601fraction", ["<digit>+"])
```

```
<iso8601fraction> ::= <digit>+
```

> **Tip**
>
> Testing with *extreme* values would mandate times such as `20:15:00,99999999999999999999` to test for possible buffer overflows.

19.4.4 Time Zones

Any time can be followed by a time zone designator. This is either `Z`, indicating Coordinated Universal Time (UTC), or an offset designating the time zone.

```
iso8601lib += make_rule("iso8601timezone", ["'Z'",
                                            "'+' <iso8601hour> (':'? <iso8601minute>)?
 ↪",
                                            "'-' <iso8601hour> (':'? <iso8601minute>)?
 ↪"], sep='\n')
```

```
<iso8601timezone> ::= 'Z' | '+' <iso8601hour> (':'? <iso8601minute>)? | '-'
 ↪<iso8601hour> (':'? <iso8601minute>)?
```

19.5 Ensuring Validity

So far, our dates and times are *syntactically* valid, but not necessarily *semantically*. We can easily create invalid dates such as `2025-02-31` (February 31) or invalid times such as `24:10:00` (10 minutes past midnight - only `24:00:00` is allowed). We could now add a number of additional *rules* to check for all these properties, and/or extend the grammar accordingly. However, we can also simply be lazy and have the existing Python `datetime` module check all this for us:

```
import datetime
```

```
def is_valid_iso8601datetime(iso8601datetime: str) -> bool:
    """Return True iff `iso8601datetime` is valid."""
    try:
        datetime.datetime.fromisoformat(iso8601datetime)
        return True
    except ValueError:
        return False
```

```
is_valid_iso8601datetime("2025-01-01T14:00")
```

```
True
```

```
is_valid_iso8601datetime("2025-02-31")
```

```
False
```

```
is_valid_iso8601datetime("2000-02-29T00:00:00")
```

```
True
```

With this, we can now add a *constraint* that limits our generator to only valid dates and times:

```
iso8601lib += make_constraint("is_valid_iso8601datetime(str(<iso8601datetime>))")
```

```
where is_valid_iso8601datetime(str(<iso8601datetime>))
```

```
iso8601lib += '''
def is_valid_iso8601datetime(iso8601datetime: str) -> bool:
    """Return True iff `iso8601datetime` is valid."""
    try:
        datetime.datetime.fromisoformat(iso8601datetime)
        return True
    except ValueError:
        return False
'''
```

19.6 Even Better Validity

The Python `datetime` module has a some limitations, which extend to our spec. For instance, `datetime` does not support `24:00:00` as a valid time:

```
is_valid_iso8601datetime("2024-12-31T24:00:00")
```

```
False
```

Hence, our `iso8601.fan` spec may miss out a number of ISO 8601 features.

The `dateutil` alternative[46] provides an ISO 8601 parser without these deficiencies.

Let us redefine `is_valid_iso8601datetime()` to make use of the `dateutil` parser:

```
def is_valid_iso8601datetime(iso8601datetime: str) -> bool:
    """Return True iff `iso8601datetime` is valid."""
    try:
        dateutil.parser.isoparse(iso8601datetime)
        return True
    except ValueError:
        return False
```

Let us see if the above example now works:

```
is_valid_iso8601datetime("2024-12-31T24:00:00")
```

```
True
```

This now works! How about the earlier examples?

```
is_valid_iso8601datetime("2025-01-01T14:00")
```

```
True
```

```
is_valid_iso8601datetime("2025-02-31")
```

```
False
```

```
is_valid_iso8601datetime("2000-02-29T00:00:00")
```

```
True
```

These also work. Let us fix the spec to use `dateutil` instead:

```
iso8601lib = iso8601lib.replace('datetime.datetime.fromisoformat', 'dateutil.parser.
↪isoparse')
iso8601lib = iso8601lib.replace('import datetime', 'import dateutil  # See https://
↪dateutil.readthedocs.io');
```

[46] https://dateutil.readthedocs.io/en/stable/index.html

19.7 Fuzzing Dates and Times

Our ISO 8601 spec is now complete. Let us write it into a `.fan` file, so we can use it for fuzzing:

```
open('ISO8601.fan', 'w').write(iso8601lib);
```

Here comes `iso8601.fan` in all its glory:

```
# ISO 8601 date and time
# Generated by docs/ISO8601.md. Do not edit.

import dateutil  # See https://dateutil.readthedocs.io

<start> ::= <iso8601datetime>
<iso8601datetime> ::= <iso8601date> ('T' <iso8601time>)?

<iso8601date> ::= <iso8601calendardate> | <iso8601weekdate> |
<iso8601ordinaldate>

<iso8601calendardate> ::= <iso8601year> '-' <iso8601month> ('-' <iso8601day>)?
| <iso8601year> <iso8601month> <iso8601day>
<iso8601year> ::= ('+'|'-')? <digit>{4}
<iso8601month> ::= '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' | '09'
| '10' | '11' | '12'
<iso8601day> ::= '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' | '09' |
'10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20' |
'21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31'

<iso8601weekdate> ::= <iso8601year> '-'? 'W' <iso8601week> ('-'
<iso8601weekday>)?
<iso8601week> ::= '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' | '09'
| '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20' |
'21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31' |
'32' | '33' | '34' | '35' | '36' | '37' | '38' | '39' | '40' | '41' | '42' |
'43' | '44' | '45' | '46' | '47' | '48' | '49' | '50' | '51' | '52' | '53'
<iso8601weekday> ::= '1' | '2' | '3' | '4' | '5' | '6' | '7'

<iso8601ordinaldate> ::= <iso8601year> ('-'? <iso8601ordinalday>)?
<iso8601ordinalday> ::= '001' | '002' | '003' | '004' | '005' | '006' | '007' |
'008' | '009' | '010' | '011' | '012' | '013' | '014' | '015' | '016' | '017' |
'018' | '019' | '020' | '021' | '022' | '023' | '024' | '025' | '026' | '027' |
'028' | '029' | '030' | '031' | '032' | '033' | '034' | '035' | '036' | '037' |
'038' | '039' | '040' | '041' | '042' | '043' | '044' | '045' | '046' | '047' |
'048' | '049' | '050' | '051' | '052' | '053' | '054' | '055' | '056' | '057' |
'058' | '059' | '060' | '061' | '062' | '063' | '064' | '065' | '066' | '067' |
'068' | '069' | '070' | '071' | '072' | '073' | '074' | '075' | '076' | '077' |
'078' | '079' | '080' | '081' | '082' | '083' | '084' | '085' | '086' | '087' |
'088' | '089' | '090' | '091' | '092' | '093' | '094' | '095' | '096' | '097' |
'098' | '099' | '100' | '101' | '102' | '103' | '104' | '105' | '106' | '107' |
'108' | '109' | '110' | '111' | '112' | '113' | '114' | '115' | '116' | '117' |
'118' | '119' | '120' | '121' | '122' | '123' | '124' | '125' | '126' | '127' |
'128' | '129' | '130' | '131' | '132' | '133' | '134' | '135' | '136' | '137' |
'138' | '139' | '140' | '141' | '142' | '143' | '144' | '145' | '146' | '147' |
'148' | '149' | '150' | '151' | '152' | '153' | '154' | '155' | '156' | '157' |
'158' | '159' | '160' | '161' | '162' | '163' | '164' | '165' | '166' | '167' |
'168' | '169' | '170' | '171' | '172' | '173' | '174' | '175' | '176' | '177' |
'178' | '179' | '180' | '181' | '182' | '183' | '184' | '185' | '186' | '187' |
```

(continues on next page)

(continued from previous page)

```
'188' | '189' | '190' | '191' | '192' | '193' | '194' | '195' | '196' | '197' |
'198' | '199' | '200' | '201' | '202' | '203' | '204' | '205' | '206' | '207' |
'208' | '209' | '210' | '211' | '212' | '213' | '214' | '215' | '216' | '217' |
'218' | '219' | '220' | '221' | '222' | '223' | '224' | '225' | '226' | '227' |
'228' | '229' | '230' | '231' | '232' | '233' | '234' | '235' | '236' | '237' |
'238' | '239' | '240' | '241' | '242' | '243' | '244' | '245' | '246' | '247' |
'248' | '249' | '250' | '251' | '252' | '253' | '254' | '255' | '256' | '257' |
'258' | '259' | '260' | '261' | '262' | '263' | '264' | '265' | '266' | '267' |
'268' | '269' | '270' | '271' | '272' | '273' | '274' | '275' | '276' | '277' |
'278' | '279' | '280' | '281' | '282' | '283' | '284' | '285' | '286' | '287' |
'288' | '289' | '290' | '291' | '292' | '293' | '294' | '295' | '296' | '297' |
'298' | '299' | '300' | '301' | '302' | '303' | '304' | '305' | '306' | '307' |
'308' | '309' | '310' | '311' | '312' | '313' | '314' | '315' | '316' | '317' |
'318' | '319' | '320' | '321' | '322' | '323' | '324' | '325' | '326' | '327' |
'328' | '329' | '330' | '331' | '332' | '333' | '334' | '335' | '336' | '337' |
'338' | '339' | '340' | '341' | '342' | '343' | '344' | '345' | '346' | '347' |
'348' | '349' | '350' | '351' | '352' | '353' | '354' | '355' | '356' | '357' |
'358' | '359' | '360' | '361' | '362' | '363' | '364' | '365' | '366'

<iso8601time> ::= 'T'? <iso8601hour> (':'? <iso8601minute> (':'?
<iso8601second> (('.' | ',') <iso8601fraction>)? )? )? <iso8601timezone>?

# 24:00:00 is allowed to represent midnight at the end of a day
<iso8601hour> ::= '00' | '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08'
| '09' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' |
'20' | '21' | '22' | '23' | '24'
<iso8601minute> ::= '00' | '01' | '02' | '03' | '04' | '05' | '06' | '07' |
'08' | '09' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' |
'19' | '20' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' |
'30' | '31' | '32' | '33' | '34' | '35' | '36' | '37' | '38' | '39' | '40' |
'41' | '42' | '43' | '44' | '45' | '46' | '47' | '48' | '49' | '50' | '51' |
'52' | '53' | '54' | '55' | '56' | '57' | '58' | '59'

# xx:yy:60 is allowed to represent leap seconds
<iso8601second> ::= '00' | '01' | '02' | '03' | '04' | '05' | '06' | '07' |
'08' | '09' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' |
'19' | '20' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' |
'30' | '31' | '32' | '33' | '34' | '35' | '36' | '37' | '38' | '39' | '40' |
'41' | '42' | '43' | '44' | '45' | '46' | '47' | '48' | '49' | '50' | '51' |
'52' | '53' | '54' | '55' | '56' | '57' | '58' | '59' | '60'
<iso8601fraction> ::= <digit>+

<iso8601timezone> ::= 'Z' | '+' <iso8601hour> (':'? <iso8601minute>)? | '-'
<iso8601hour> (':'? <iso8601minute>)?
where is_valid_iso8601datetime(str(<iso8601datetime>))

def is_valid_iso8601datetime(iso8601datetime: str) -> bool:
    """Return True iff `iso8601datetime` is valid."""
    try:
        dateutil.parser.isoparse(iso8601datetime)
        return True
    except ValueError:
        return False
```

With this, we can now create a bunch of random date/time elements:

```
$ fandango fuzz -f iso8601.fan -n 10
```

```
8166-W42
4021-W11-5
4906-04
3973-358
8841-04
```

```
2993051
2791-11-06T16:28
96341118
6112-12-27
7875W45
```

And we can use additional constraints to further narrow down date intervals:

```
$ fandango fuzz -f ISO8601.fan -n 10 -c 'int(<iso8601year>) > 1950 and int(
↪<iso8601year>) < 2000'
```

```
1968-175
```

```
1964-247T07
```

```
1964-W45-5T24+16:44
```

```
1964-247T10
19680601
1964-247T03:24:42
```

```
1968-175T04Z
1998-175
```

```
1964-W41-7T10+16
```

```
1960-247T10
```

Or produce individual elements, again with individual constraints:

```
$ fandango fuzz -f ISO8601.fan -n 10 --start-symbol='<iso8601time>' -c '<iso8601hour>␣
↪== "00"'
```

```
00
0055
00:02
```

```
T00:0250,40499784166447424
T00
T00:2549
T000336-00
T00:13:12,7568604487170323473
T0015:41,368708432130
0008
```

Try out more constraints for yourself! The generated `ISO8601.fan` file is available for download.

Part IV

Generating Binary Inputs

CHAPTER

TWENTY

GENERATING BINARY INPUTS

Creating *binary* inputs with Fandango is a bit more challenging than creating human-readable inputs. This is because they have a few special features, such as *checksums* and *length encodings*. Fortunately, we can address all of them with dedicated constraints.

20.1 Checksums

Strictly speaking, this only holds for context-free grammars Fandango uses. *Context-sensitive* and *universal* grammars can perform arithmetic computations, but someone would have to implement them all.

Our first challenge is *checksums*. Binary input formats frequently use checksums to ensure integrity. The problem is that checksums cannot be expressed in a grammar alone, as grammars lack the arithmetic functions required to compute and check checksums. In Fandango, though, we can express the computation of a checksum in a dedicated function, which is then used in a dedicated constraint.

As an example for checksums, let's have a look at *credit card numbers*. These are definitely very human-readable and not binary at all, but for an example, they will do fine. A credit card number consists of a series of digits, where the last one is a *check digit*. Here is a grammar that expresses the structure for 16-digit credit card numbers:

```
<start>              ::= <credit_card_number>
<credit_card_number> ::= <number> <check_digit>
<number>             ::= <digit>{15}  # for 16-digit numbers
<check_digit>        ::= <digit>
```

All credit cards use Luhn's algorithm[47] to compute the check digit. Here is an implementation, adapted from the Faker library[48]. The function `credit_card_check_digit()` gets all numbers of a credit card (except the last digit) and returns the computed check digit.

```
def credit_card_check_digit(number: str) -> str:
    """Create a check digit for the credit card number `number`."""
    luhn_lookup = {
        "0": 0,
        "1": 2,
        "2": 4,
        "3": 6,
```

(continues on next page)

[47] https://en.wikipedia.org/wiki/Luhn_algorithm
[48] https://github.com/joke2k/faker/blob/master/faker/providers/credit_card/__init__.py#L99

(continued from previous page)

```
        "4": 8,
        "5": 1,
        "6": 3,
        "7": 5,
        "8": 7,
        "9": 9,
    }

    # Calculate sum
    length = len(number) + 1
    reverse = number[::-1]
    tot = 0
    pos = 0
    while pos < length - 1:
        tot += luhn_lookup[reverse[pos]]
        if pos != (length - 2):
            tot += int(reverse[pos + 1])
        pos += 2

    # Calculate check digit
    check_digit = (10 - (tot % 10)) % 10
    return str(check_digit)
```

We can easily make use of `credit_card_check_digit()` in a constraint that ties `<check_digit>` and `<number>`:

```
where <check_digit> == credit_card_check_digit(str(<number>))
```

All of this can go into a single `.fan` file: `credit_card.fan` joins the above grammar, the `credit_card_check_digit()` definition, and the above constraint into a single file.

Do not use such numbers to test third-party systems.

We can now use `credit-card.fan` to produce valid credit card numbers:

```
$ fandango fuzz -f credit_card.fan -n 10
```

```
0221593856605222
3642812364885885
```

```
6202463767783312
0951122983064610
8846981668830591
0999437570139524
4328184496013326
7310538693726126
```

```
3741107377536595
3184990543901973
```

We can also use the grammar to *parse* and *check* numbers. This credit card number should be valid:

```
$ echo -n 4931633575526870 | fandango parse -f credit_card.fan
$ echo $?  # print exit code
0
```

Adding a `1` digit to this number should make it *invalid*:

```
$ echo -n 4931633575526871 | fandango parse -f credit_card.fan
```

```
FandangoParseError: Did not match the following constraints: <check_digit> ==␣
↪credit_card_check_digit(str(<number>))
FandangoParseError: Did not match the following constraints: <check_digit> ==␣
↪credit_card_check_digit(str(<number>))
Traceback (most recent call last):
  File "/Users/zeller/.virtualenvs/python3.13/bin/fandango", line 7, in <module>
    sys.exit(main())
             ~~~~^^
  File "<@beartype(fandango.cli.main) at 0x109e77060>", line 68, in main
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/__init__.py", line 47,␣
↪in main
    last_status = run(command, args)
  File "<@beartype(fandango.cli.commands.run) at 0x10a4bd1c0>", line 51, in run
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 487,␣
↪in run
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
  File "<@beartype(fandango.logger.print_exception) at 0x108e228e0>", line 54, in␣
↪print_exception
  File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
↪print_exception
    raise e
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 485,␣
↪in run
    command(args)
    ~~~~~~~^^^^^^
  File "<@beartype(fandango.cli.commands.parse_command) at 0x10a4bc900>", line 33,␣
↪in parse_command
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 270,␣
↪in parse_command
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
  File "<@beartype(fandango.logger.print_exception) at 0x108e228e0>", line 54, in␣
↪print_exception
  File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
↪print_exception
    raise e
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 267,␣
↪in parse_command
    tree = parse_file(fd, args, grammar, constraints, settings)
  File "<@beartype(fandango.cli.utils.parse_file) at 0x109ec0400>", line 136, in␣
↪parse_file
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/utils.py", line 324, in␣
↪parse_file
    raise FandangoParseError(
        f"Did not match the following constraints: {', '.join(failed_constraints)}"
    )
fandango.errors.FandangoParseError: Did not match the following constraints:
↪<check_digit> == credit_card_check_digit(str(<number>))
```

```
$ echo $?  # print exit code
1
```

You can also simply do an Internet search for a Python implementation of the respective algorithm. Or ask your favorite AI assistant.

Similarly, you can define any kind of checksum function and then use it in a constraint. In Python, it is likely that someone has already implemented the specific checksum function, so you can also *import* it:

- The `hashlib` module[49] provides hash functions such as MD5 or SHA-256.
- The `binascii` module[50] offers CRC checks.
- The `zlib` module[51] provides CRC32 and ADLER32 checks used in zip files.

20.2 Characters and Bytes

The second set of features one frequently encounters in binary formats is, well, *bytes*. So far, we have seen Fandango operates on strings of Unicode *characters*, which use UTF-8 encoding. This clashes with a byte interpretation as soon as the produced string contains a UTF-8 prefix byte, such as `\xc2` or `\xe0`, which mark the beginning of a two- and three-byte UTF-8 sequence, respectively.

To ensure bytes will be interpreted as bytes (and as bytes only), place a `b` (binary) prefix in front of them. This ensures that a byte `b'\xc2'` will always be interpreted as a single byte, whereas □ will be interpreted as a single character (despite occupying multiple bytes).

Tip

Fandango provides a `<byte>` symbol by default, which expands into all bytes `b'\x00'..b'\xff'`.

20.2.1 Text Files and Binary Files

By default, Fandango will read and write files in `text` mode, meaning that characters will be read in using UTF-8 encoding. However, if a grammar can produce bytes (or *bits* (page 137)), the associated files will be read and written in `binary` mode, reading and writing *bytes* instead of (encoded) characters. If your grammar contains bytes *and* strings, then the strings will be written in UTF-8 encoding into the binary file.

You can enforce a specific behavior using the Fandango `--file-mode` flag for the `fuzz` and `parse` commands:

- `fuzz --file-mode=text` opens all files in `text` mode. Strings and bytes will be written in UTF-8 encoding.
- `fuzz --file-mode=binary` opens all files in `binary` mode. Strings will be written in UTF-8 encoding; bytes will be written as is.

The default is `fuzz --file=mode=auto` (default), which will use `binary` or `text` mode as described above.

[49] https://docs.python.org/3/library/hashlib.html
[50] https://docs.python.org/3/library/binascii.html
[51] https://docs.python.org/3/library/zlib.html

> **Tip**
>
> Avoid mixing non-ASCII strings with bits and bytes in a single grammar.

20.2.2 Bytes and Regular Expressions

Fandango also supports *regular expressions* (page 75) over bytes. To obtain a regular expression over a byte string, use both `r` and `b` prefixes. This is especially useful for character classes.

Here is an example: `binfinity.fan` produces strings of five bytes *outside* the range `\x80-\xff`:

```
<start> ::= rb"[^\x80-\xff]{5}"
```

This is what we get:

```
$ fandango fuzz -f binfinity.fan -n 10
```

```
o.BrF
tgEoA
!I#uI
I>NEy
x8xBW
B69a4
b:YDq
```

```
@6n8V
u^4;0
0%vcK
```

20.3 Length Encodings

The third set of features one frequently encounters in binary formats is *length encodings* - that is, a particular field holds a value that represents the length of one or more fields that follow. Here is a simple grammar that expresses this characteristic: A `<field>` has a two-byte length, followed by the actual (byte) content (of length `<length>`).

```
<start>    ::= <field>
<field>    ::= <length> <content>
<length>   ::= <byte> <byte>
<content>  ::= <byte>+
```

20.3.1 Encoding Lengths with Constraints

The relationship between `<length>` and `<content>` can again be expressed using a constraint. Let us assume that `<length>` comes as a two-byte (16-bit) unsigned integer with *little-endian* encoding - that is, the low byte comes first, and the high byte follows. The value 258 (hexadecimal `0x0102`) would thus be represented as the two bytes `\x02` and `\x01`.

We can define a function `uint16()` that takes an integer and converts it to a two-byte string according to these rules. The Python method `N.to_bytes(LENGTH, ENDIANNESS)` converts the integer `N` into a bytes string of length `LENGTH`. `ENDIANNESS` is either `'big'` (default) or `'little'`.

```
def uint16(n: int) -> bytes:
    return n.to_bytes(2, 'little')
```

Using `uint16()`, we can now define how the value of `<length>` is related to the length of `<content>`:

```
where <length> == uint16(len(bytes(<content>)))
```

> **Tip**
>
> Having a derived value (like `<length>`) isolated on the left-hand side of an equality equation makes it easy for Fandango to first compute the content and then compute and assign the derived value.

Again, all of this goes into a single `.fan` file: `binary.fan` holds the grammar, the `uint16()` definition, and the constraint. Let us produce a single output using `binary.fan` and view its (binary) contents, using `od -c`:

```
$ fandango fuzz -n 1 -f binary.fan | hexdump -C
```

```
00000000  0c 00 3b c2 95 31 10 c2  92 41 54 47 c2 86 c3 83  |..;..1...ATG....|
```

```
00000010  63 44 0a                                          |cD.|
00000013
```

The hexadecimal dump shows that the first two bytes encode the length of the string of digits that follows. The format is correct - we have successfully produced a length encoding.

20.3.2 Encoding Lengths with Repetitions

Another way to implement length constraints is by using *repetitions*. In Fandango, repetitions `{}` can also contain *expressions*, and like constraints, these can also refer to nonterminals that have already been parsed or produced. Hence, we can specify a rule

```
<content> ::= <byte>{f(<length>)}
```

where `f()` is a function that computes the number of `<byte>` repetitions based on `<length>`.

Let us define a variant `binary-rep.fan` that makes use of this. Here, we specify that `<content>` consists of `N` bytes, where `N` is given as follows:

We use a generator expression `:= VALUE` to prevent generated values from getting too large.

```
<start>    ::= <field>
<field>    ::= <length> <content>
<length>   ::= <byte> <byte> := b'\x00\x04'
<content>  ::= <byte>{from_uint16(bytes(<length>))}
```

The method `<length>.value()` returns the bytes string value of the `<length>` element. The function `from_uint16()` is defined as follows:

```
def from_uint16(n: bytes) -> int:
    return n[0] << 8 | n[1]
```

With this, we can easily produce length-encoded inputs:

```
$ fandango fuzz -n 1 -f binary-rep.fan | hexdump -C
```

```
00000000  00 04 c2 95 3a c3 a6 57  0a                       |....:..W.|
00000009
```

> **Tip**
>
> When *parsing* (page 159) inputs, computed repetitions are much more efficient than constraints.

20.4 Converting Values to Binary Formats

Instead of implementing `uint16()` manually, we can also use the Python `struct` module[52], which offers several functions to convert data into binary formats. Using `struct`, we can redefine `uint16()` as

```
from struct import pack

def uint16(n: int) -> str:
    return pack('<H', n).decode('iso8859-1')  # convert to string
```

and obtain the same result:

```
0000000   \r  \0   3   7   3   2   6   2   4   7   2   7   7   7   4  \n
```

```
0000020
```

Note that the return value of `struct.pack()` has the type `bytes` (byte string), which is different from the `str` Unicode strings that Fandango uses:

```
pack('<H', 20)
```

```
b'\x14\x00'
```

```
type(pack('<H', 20))
```

```
bytes
```

In Python, comparisons of different types always return `False`:

```
# Left hand is byte string, right hand is Unicode string
b'\x14\x00' == '\x14\x00'
```

```
False
```

[52] https://docs.python.org/3/library/struct.html

Hence, a constraint that compares a Fandango symbol against a byte string *will always fail*.

Important

When comparing symbols against values, always be sure to convert the values to the appropriate type first.

Tip

Using the `'iso8859-1'` encoding (also known as `'latin-1'`) allows a 1:1 conversion of byte strings with values `'\x00'..'\xff'` into Unicode `str` strings without further interpretation.

Tip

Adding *type annotations* to functions in `.fan` files allows for future static type checking and further optimizations.

Check out the `struct` module[53] for additional encodings, including float types, long and short integers, and many more.

[53] https://docs.python.org/3/library/struct.html

CHAPTER

TWENTYONE

BITS AND BIT FIELDS

Some binary inputs use individual *bits* to specify contents. For instance, you might have a `flag` byte that holds multiple (bit) flags:

```
struct {
  unsigned int italic: 1;  // 1 bit
  unsigned int bold: 1;
  unsigned int underlined: 1;
  unsigned int strikethrough: 1;
  unsigned int brightness: 4;  // 4 bits
} format_flags;
```

How does one represent such *bit fields* in a Fandango spec?

21.1 Representing Bits

In Fandango, bits can be represented in Fandango using the special values `0` (for a zero bit) and `1` (for a non-zero bit). Hence, you can define a `<bit>` value as

```
<bit> ::= 0 | 1
```

With this, the above `format_flag` byte would be specified as

```
<start>         ::= <format_flag>
<format_flag>   ::= <italic> <bold> <underlined> <strikethrough> <brightness>
<italic>        ::= <bit>
<bold>          ::= <bit>
<underlined>    ::= <bit>
<strikethrough> ::= <bit>
<brightness>    ::= <bit>{4}
<bit>           ::= 0 | 1
```

A `<format_flag>` symbol would thus always consist of these eight bits. We can use the special option `--format=bits` to view the output as a bit stream:

```
$ fandango fuzz --format=bits -f bits.fan -n 1 --start-symbol='<format_flag>'
```

```
11001011
```

> **Tip**
>
> The combination of `--format=bits` and `--start-symbol` is particularly useful to debug bit fields.

Internally, Fandango treats individual flags as integers, too. Hence, we can also apply *constraints* to the individual flags. For instance, we can profit from the fact that Python treats `0` as False and `1` as True:

```
$ fandango fuzz --format=bits -f bits.fan -n 10 -c '<italic> and <bold>'
```

```
11111111
10100000
10101001
11010001
01001011
10011011
01000000
00001111
10010010
01100100
```

Fandango strictly follows a "left-to-right" order - that is, the order in which bits and bytes are specified in the grammar, the most significant bit is stored first.

Hence, we can also easily set the value of the entire `brightness` field using a constraint:

```
$ fandango fuzz --format=bits -f bits.fan -n 1 -c '<brightness> == 0b1111'
```

```
00001111
```

> **Note**
>
> Fandango always strictly follows a "left-to-right" order - that is, the order in which bits and bytes are specified in the grammar.

Of course, we can also give the number in decimal format:

```
$ fandango fuzz --format=bits -f bits.fan -n 1 -c '<brightness> == 15'
```

```
00001111
```

```
00111111
00101111
```

```
10011111
```

```
10111111
```

```
00011111
10101111
11101111
```

```
01101111
```

```
01111111
```

Note how the last four bits (the `<brightness>` field) are always set to `1111` - the number 15.

Important

When implementing a format, be sure to follow its conventions regarding

- *bit ordering* (most or least significant bit first)
- *byte ordering* (most or least significant byte first)

21.2 Parsing Bits

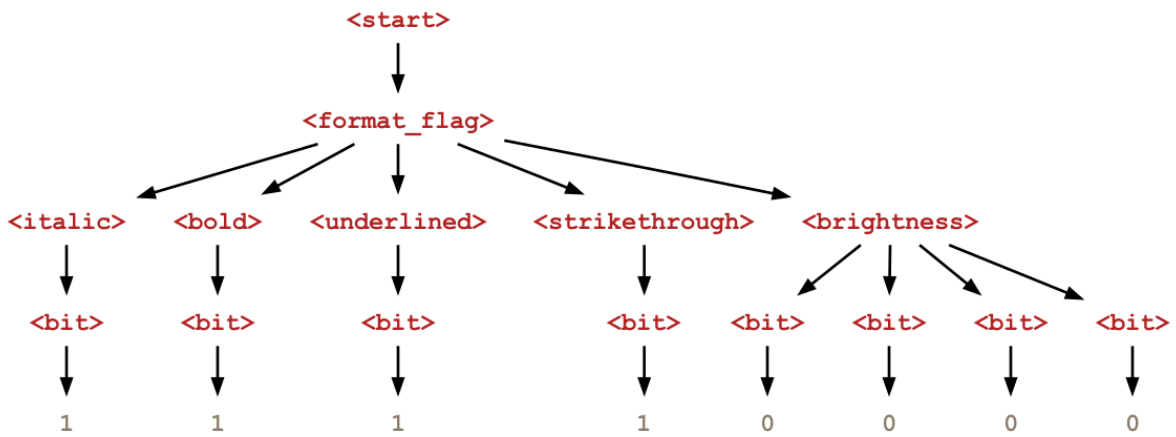
Fandango also supports *parsing* (page 159) inputs with bits. This is what happens if we send a byte `\xf0` (the upper four bits set) to the parser:

```
$ printf "\xf0" | fandango parse -f bits.fan -o - --format=bits
```

```
11110000
```

We see that the input was properly parsed and decomposed into individual bits.

This is the resulting parse tree:



The `grammar` format shows us that the values are properly assigned:

```
$ printf '\xf0' | fandango parse -f bits.fan -o - --format=grammar
```

```
<start> ::= <format_flag>
  <format_flag> ::= <italic> <bold> <underlined> <strikethrough> <brightness>  #␣
↪0b11110000
    <italic> ::= <bit>
      <bit> ::= 1  # Position 0x0000 (0), bit 7
    <bold> ::= <bit>
```

(continues on next page)

(continued from previous page)

```
    <bit> ::= 1  # Position 0x0000 (0), bit 6
<underlined> ::= <bit>
    <bit> ::= 1  # Position 0x0000 (0), bit 5
<strikethrough> ::= <bit>
    <bit> ::= 1  # Position 0x0000 (0), bit 4
<brightness> ::= <bit> <bit> <bit> <bit>  # 0b0000
    <bit> ::= 0  # Position 0x0000 (0), bit 3
    <bit> ::= 0  # Position 0x0000 (0), bit 2
    <bit> ::= 0  # Position 0x0000 (0), bit 1
    <bit> ::= 0  # Position 0x0000 (0), bit 0
```

Note

To parse bits properly, they must come in multiples of eight.

21.3 Bits and Padding

When generating binary inputs, you may need to adhere to specific *lengths*. Such lengths are often enforced by *padding* – that is, adding bits until the required length is achieved. For instance, let us assume you have a field consisting of some bits. However, the overall length of the field must be a multiple of eight to have it byte-aligned. For such *padding*, define the field as

```
<field> ::= <bits> <padding>
<padding> ::= 0*
```

combined with a constraint

```
where len(<field>) % 8 == 0
```

Note that applied on derivation trees, `len()` always returns the number of child elements, not the string length; here, we use this to access the number of elements in `<field>`.

CHAPTER

TWENTYTWO

DATA CONVERTERS

When defining a complex input format, some parts may be the result of applying an *operation* on another, more structured part. Most importantly, content may be *encoded*, *compressed*, or *converted*.

Fandango uses a special form of *generators* (page 67) to handle these, called *converters*. These are generator expressions with *symbols*, mostly functions that take symbols as arguments. Let's have a look at how these work.

22.1 Encoding Data During Fuzzing

In Fandango, a *generator* (page 67) expression can contain *symbols* (enclosed in `<...>`) as elements. Such generators are called *converters*. When fuzzing, converters have the effect of Fandango using the grammar to

- instantiate each symbol from the grammar,
- evaluate the resulting expression, and
- return the resulting value.

Here is a simple example to get you started. The Python `base64` module[54] provides methods to encode arbitrary binary data into printable ASCII characters:

```
import base64

encoded = base64.b64encode(b'Fandango\x01')
encoded
```

```
b'RmFuZGFuZ28B'
```

Of course, these can be decoded again:

```
base64.b64decode(encoded)
```

```
b'Fandango\x01'
```

Let us make use of these functions. Assume we have a `<data>` field that contains a number of bytes:

```
<data> ::= b'Fandango' <byte>+
```

To encode such a `<data>` field into an `<item>`, we can write

```
<item> ::= rb'.*' := base64.b64encode(bytes(<data>))
```

[54] https://docs.python.org/3/library/base64.html

This rule brings multiple things together:

- First, we convert `<data>` into a suitable type (in our case, `bytes`).
- Then, we invoke `base64.b64encode()` on it as a generator to obtain a string of bytes.
- We parse the string into an `<item>`, whose definition is `rb'.*'` (any sequence of bytes except newline).

In a third step, we embed the `<item>` into a (binary) string:

```
<start> ::= b'Data: ' <item>
```

The full resulting `encode.fan` spec looks like this:

```
import base64

<start> ::= b'Data: ' <item>
<item> ::= rb'.*' := base64.b64encode(bytes(<data>))
<data> ::= b'Fandango' <byte>+
```

With this, we can encode and embed binary data:

```
$ fandango fuzz -f encode.fan -n 1
```

```
Data: RmFuZGFuZ28lISlKIu678SvulQIg4nI1
```

In the same vein, one can use functions for compressing data or any other kind of conversion.

22.2 Sources, Encoders, and Constraints

When Fandango produces an input using a generator, it *saves* the generated arguments as a *source* in the produced derivation tree. Sources become visible as soon as the input is shown as a grammar:

```
$ fandango fuzz -f encode.fan -n 1 --format=grammar
```

```
<start> ::= b'Data: ' <item>  # Position 0x0000 (0); b'Data:␣
↪RmFuZGFuZ28lISlKIu678SvulQIg4nI1'
 <item> ::= b'RmFuZGFuZ28lISlKIu678SvulQIg4nI1' := f(<data>)  # Position 0x0006␣
↪(6)
   <data> ::= b'Fandango' <byte> <byte> <byte> <byte> <byte> <byte> <byte> <byte>
↪<byte> <byte> <byte> <byte> <byte> <byte> <byte> <byte>  # Position 0x0000 (0); b
↪'Fandango%!)J"\xee\xbb\xf1+\xee\x95\x02 \xe2r5'
     <byte> ::= <_byte>
       <_byte> ::= b'%'  # Position 0x0008 (8)
     <byte> ::= <_byte>
       <_byte> ::= b'!'  # Position 0x0009 (9)
     <byte> ::= <_byte>
       <_byte> ::= b')'  # Position 0x000a (10)
     <byte> ::= <_byte>
       <_byte> ::= b'J'  # Position 0x000b (11)
     <byte> ::= <_byte>
       <_byte> ::= b'"'  # Position 0x000c (12)
     <byte> ::= <_byte>
       <_byte> ::= b'\xee'  # Position 0x000d (13)
     <byte> ::= <_byte>
       <_byte> ::= b'\xbb'  # Position 0x000e (14)
```

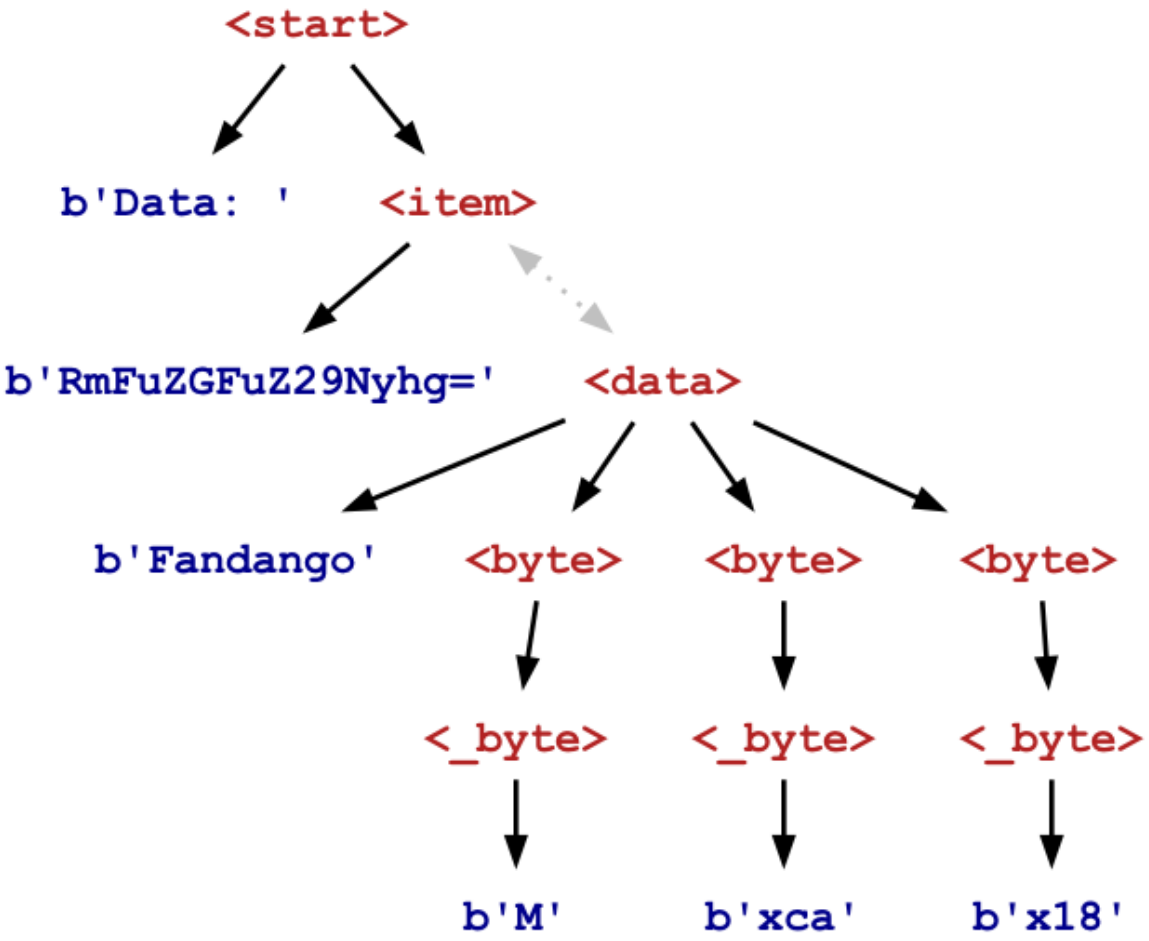
(continues on next page)

(continued from previous page)

```
<byte> ::= <_byte>
  <_byte> ::= b'\xf1'  # Position 0x000f (15)
<byte> ::= <_byte>
  <_byte> ::= b'+'  # Position 0x0010 (16)
<byte> ::= <_byte>
  <_byte> ::= b'\xee'  # Position 0x0011 (17)
<byte> ::= <_byte>
  <_byte> ::= b'\x95'  # Position 0x0012 (18)
<byte> ::= <_byte>
  <_byte> ::= b'\x02'  # Position 0x0013 (19)
<byte> ::= <_byte>
  <_byte> ::= b' '  # Position 0x0014 (20)
<byte> ::= <_byte>
  <_byte> ::= b'\xe2'  # Position 0x0015 (21)
<byte> ::= <_byte>
  <_byte> ::= b'r'  # Position 0x0016 (22)
<byte> ::= <_byte>
  <_byte> ::= b'5'  # Position 0x0017 (23)
```

In the definition of `<item>`, we see a generic converter `f(<data>)` as well as the definition of `<data>` that went into the generator. (The actual generator code, `base64.b64encode(bytes(<data>))`, is not saved in the derivation tree.)

We can visualize the resulting tree, using a double arrow between `<item>` and its source `<data>`, indicating that their values depend on each other:



Since sources like `<data>` are preserved, we can use them in *constraints* (page 45). For instance, we can produce a string with specific values for `<data>`:

```
$ fandango fuzz -f encode.fan -n 1 -c '<data> == b"Fandango author"'
```

```
Data: RmFuZGFuZ28gYXV0aG9y
```

Is this string a correct encoding of a correct string? We will see in the next section.

22.3 Decoding Parsed Data

So far, we can only *encode* data during fuzzing. But what if we also want to *decode* data, say during *parsing* (page 159)? Our `encode.fan` will help us *parse* the data, but not decode it:

```
$ echo -n 'Data: RmFuZGFuZ28gYXV0aG9y' | fandango parse -f encode.fan
```

```
FandangoValueError: <data>: Missing converter from <item> (<data> ::= ... := f(
↪<item>))
FandangoValueError: <data>: Missing converter from <item> (<data> ::= ... := f(
↪<item>))
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
  File "/Users/zeller/.virtualenvs/python3.13/bin/fandango", line 7, in <module>
    sys.exit(main())
             ~~~~^^
  File "<@beartype(fandango.cli.main) at 0x107d47060>", line 68, in main
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/__init__.py", line 47,␣
↪in main
    last_status = run(command, args)
  File "<@beartype(fandango.cli.commands.run) at 0x10838d1c0>", line 51, in run
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 487,␣
↪in run
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
  File "<@beartype(fandango.logger.print_exception) at 0x106cf28e0>", line 54, in␣
↪print_exception
  File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
↪print_exception
    raise e
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 485,␣
↪in run
    command(args)
    ~~~~~~~^^^^^^
  File "<@beartype(fandango.cli.commands.parse_command) at 0x10838c900>", line 33,␣
↪in parse_command
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 270,␣
↪in parse_command
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
  File "<@beartype(fandango.logger.print_exception) at 0x106cf28e0>", line 54, in␣
↪print_exception
  File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
↪print_exception
    raise e
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 267,␣
↪in parse_command
    tree = parse_file(fd, args, grammar, constraints, settings)
  File "<@beartype(fandango.cli.utils.parse_file) at 0x107d90400>", line 136, in␣
↪parse_file
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/utils.py", line 311, in␣
↪parse_file
    for tree in gen:
                ^^^
  File "/Users/zeller/Projects/Fandango!/src/fandango/evolution/__init__.py", line␣
↪22, in __iter__
    self._return_value = yield from self.generator
                         ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/zeller/Projects/Fandango!/src/fandango/api.py", line 463, in parse
    self.grammar.populate_sources(tree)
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^
  File "/Users/zeller/Projects/Fandango!/src/fandango/language/grammar/grammar.py",
↪ line 165, in populate_sources
    self._populate_sources(tree)
    ~~~~~~~~~~~~~~~~~~~~~~^^^^^^
  File "/Users/zeller/Projects/Fandango!/src/fandango/language/grammar/grammar.py",
↪ line 174, in _populate_sources
    self._populate_sources(child)
    ~~~~~~~~~~~~~~~~~~~~~~^^^^^^^
```

(continues on next page)

(continued from previous page)

```
  File "/Users/zeller/Projects/Fandango!/src/fandango/language/grammar/grammar.py",
↪ line 169, in _populate_sources
    tree.sources = self.derive_sources(tree)
                   ~~~~~~~~~~~~~~~~~~~^^^^^^
  File "/Users/zeller/Projects/Fandango!/src/fandango/language/grammar/grammar.py",
↪ line 137, in derive_sources
    raise FandangoValueError(
        f"{val.symbol.format_as_spec()}: Missing converter from {gen_symbol.format_
↪as_spec()} ({val.symbol.format_as_spec()} ::= ... := f({gen_symbol.format_as_
↪spec()}))"
    )
fandango.errors.FandangoValueError: <data>: Missing converter from <item> (<data>␣
↪::= ... := f(<item>))
```

The fact that parsing fails is not a big surprise, as we only have specified an *encoder*, but not a *decoder*. As the error message suggests, we need to add a generator for `<data>` - a decoder that converts `<item>` elements into `<data>`.

We can achieve this by providing a generator for `<data>` that builds on `<item>`:

```
<data> ::= b'Fandango' <byte>+ := base64.b64decode(bytes(<item>))
```

Here, `base64.b64decode(bytes(<item>))` takes an `<item>` (which is previously parsed) and decodes it. The decoded result is parsed and placed in `<data>`.

The resulting `encode-decode.fan` file now looks like this:

```
import base64

<start> ::= b'Data: ' <item>
<item> ::= rb'.*' := base64.b64encode(bytes(<data>))
<data> ::= b'Fandango' <byte>+ := base64.b64decode(bytes(<item>))
```

Fandango allows generators in both directions so one `.fan` file can be used for fuzzing and parsing.

If this looks like a mutual recursive definition, that is because it is. During fuzzing and parsing, Fandango tracks the *dependencies* between generators and uses them to decide which generators to use first:

- When fuzzing, Fandango operates *top-down*, starting with the topmost generator encountered; their arguments are *produced*. In our case, this is the `<item>` generator, generating a value for `<data>`.
- When parsing, Fandango operates *bottom-up*, starting with the lowest generators encountered; their arguments are *parsed*. In our case, this is the `<data>` generator, parsing a value for `<item>`.

In both case, when Fandango encounters a recursion, *it stops evaluating the generator*:

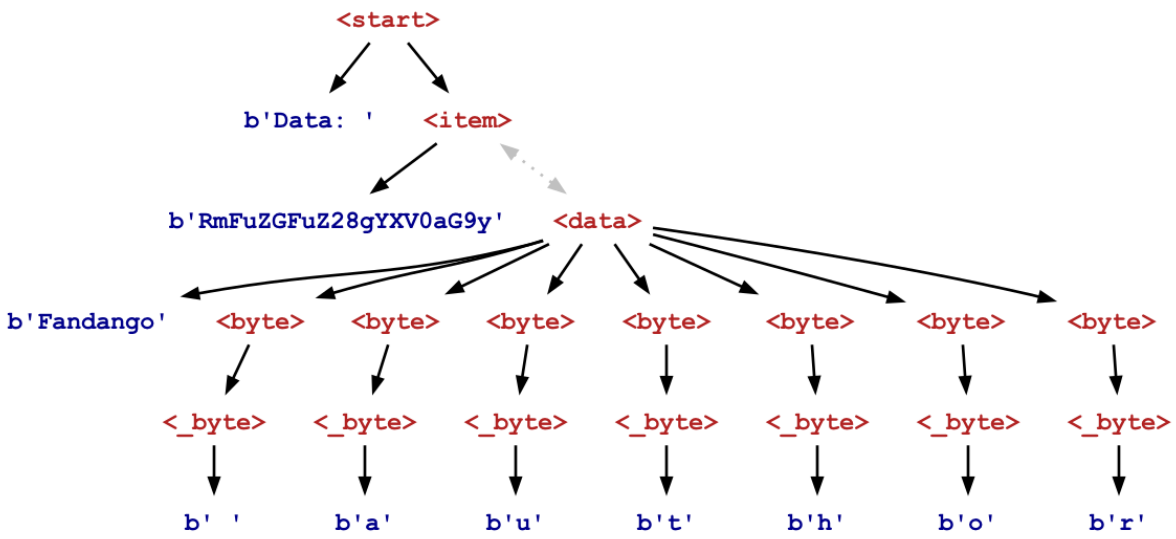
- When parsing an `<item>`, Fandango does not invoke the generator for `<data>` because `<data>` is being processed already.
- Likewise, when producing `<data>`, Fandango does not invoke the generator for `<item>` because `<item>` is being processed already.

Let us see if all of this works and if this input is indeed properly parsed and decoded.

```
$ echo -n 'Data: RmFuZGFuZ28gYXV0aG9y' | fandango parse -f encode-decode.fan -o - --
↪format=grammar
```

```
<start> ::= b'Data: ' <item>  # Position 0x0000 (0); b'Data: RmFuZGFuZ28gYXV0aG9y'
  <item> ::= b'RmFuZGFuZ28gYXV0aG9y' := f(<data>)  # Position 0x0006 (6)
    <data> ::= b'Fandango' <byte> <byte> <byte> <byte> <byte> <byte> <byte>  #␣
 ↪Position 0x0000 (0); b'Fandango author'
      <byte> ::= <_byte>
        <_byte> ::= b' '  # Position 0x0008 (8)
      <byte> ::= <_byte>
        <_byte> ::= b'a'  # Position 0x0009 (9)
      <byte> ::= <_byte>
        <_byte> ::= b'u'  # Position 0x000a (10)
      <byte> ::= <_byte>
        <_byte> ::= b't'  # Position 0x000b (11)
      <byte> ::= <_byte>
        <_byte> ::= b'h'  # Position 0x000c (12)
      <byte> ::= <_byte>
        <_byte> ::= b'o'  # Position 0x000d (13)
      <byte> ::= <_byte>
        <_byte> ::= b'r'  # Position 0x000e (14)
```

We see that the `<data>` element contains the `"Fandango author"` string we provided as a constraint during generation. This is what the parsed derivation tree looks like:



With a constraint, we can check that the decoded string is correct:

```
$ echo -n 'Data: RmFuZGFuZ28gYXV0aG9y' | fandango parse -f encode-decode.fan -c '
 ↪<data> == b"Fandango author"'
```

We get no error - so the parse was successful, and that all constraints hold.

22.4 Applications

The above scheme can be used for all kinds of encodings and compressions - and thus allow *translations between abstraction layers*. Typical applications include:

- *Compressed* data (e.g. pixels in a GIF or PNG file)
- *Encoded* data (e.g. binary input as ASCII chars in MIME encodings)
- *Converted* data (e.g. ASCII to UTF-8 to UTF-16 and back)

Even though parts of the input are encoded (or compressed), you can still use *constraints* to shape them. And if the encoding or compression can be inverted, you can also use it to *parse* inputs again.

22.5 Converters vs. Constraints

Since converters (and generally, generators) can do anything, they can be used for any purpose, including producing solutions that normally would come from *constraints* (page 45).

As an example, consider the credit card grammar from the *chapter on binary inputs* (page 129):

```
<start>              ::= <credit_card_number>
<credit_card_number> ::= <number> <check_digit>
<number>             ::= <digit>{15}  # for 16-digit numbers
<check_digit>        ::= <digit>
where <check_digit> == credit_card_check_digit(str(<number>))
```

Instead of having a constraint (`where`) that expresses the relationship between `<number>` and `<check_digit>`, we can easily enhance the grammar with converters between `<number>` and `<credit_card_number>`:

```
<credit_card_number> ::= <number> <check_digit> := add_check_digit(str(<number>))
<number>             ::= <digit>{15} := strip_check_digit(str(<credit_card_number>))
```

with

```
def add_check_digit(number: str) -> str:
    """Add a check digit to the credit card number `number`."""
    check_digit = credit_card_check_digit(number)
    return number + check_digit
```

and

```
def strip_check_digit(number: str) -> str:
    """Strip the check digit from the credit card number `number`."""
    return number[:-1]
```

The resulting `.fan` spec `credit_card-gen.fan` has the same effect as the original `credit_card.fan` from the *chapter on binary inputs* (page 129):

```
$ fandango fuzz -f credit_card-gen.fan -n 10
```

```
3318978269345441
```

```
8543462045542932
1397100388954833
2591302579883076
6595225503872443
```

```
9387529895741934
0334320254279242
4684415490253726
```

```
5573281287717609
4658514578534064
```

Now, these two functions `add_check_digit()` and `strip_check_digit()` are definitely longer than our original constraint

```
where <check_digit> == credit_card_check_digit(str(<number>))
```

However, they are not necessarily more complex. And they are more efficient, as they provide a solution right away. So when should one use constraints, and when converters?

Tip

In general:

- If you have a simple, *operational* way to solve a problem, consider a *converter*.
- If you want a simple, *declarative* way to specify your needs, use a *constraint*.

CHAPTER

TWENTYTHREE

CASE STUDY: THE GIF FORMAT

Under Construction

To be added later.

The GIF format[55] is widely used to encode image sequences.

We start with a very short GIF to keep things simple (source[56]): tinytrans.gif.

We can parse this file using Fandango:

```
!fandango parse -f gif89a.fan tinytrans.gif -o - --format=grammar --validate
assert _exit_code == 0
```

```
<start> ::= <GifHeader> <LogicalScreenDescriptor> <GlobalColorTable> <Data_1>
 ↪<Trailer>  # b'GIF89a\x01\x00\x01\x00\x80\x01\x00\xff\xff\xff\x00\x00\x00!\xf9\
 ↪x04\x01\n\x00\x01\x00,\x00\x00\x00\x00\x01\x00\x01\x00\x00\x02\x02L\x01\x00;'
  <GifHeader> ::= <GIFHEADER>
    <GIFHEADER> ::= <Signature> <Version>  # b'GIF89a'
      <Signature> ::= <char> <char> <char>  # b'GIF'
        <char> ::= <byte>
          <byte> ::= <_byte>
            <_byte> ::= b'G'  # Position 0x0000 (0)
        <char> ::= <byte>
          <byte> ::= <_byte>
            <_byte> ::= b'I'  # Position 0x0001 (1)
        <char> ::= <byte>
          <byte> ::= <_byte>
            <_byte> ::= b'F'  # Position 0x0002 (2)
      <Version> ::= <char> <char> <char>  # b'89a'
        <char> ::= <byte>
          <byte> ::= <_byte>
            <_byte> ::= b'8'  # Position 0x0003 (3)
        <char> ::= <byte>
          <byte> ::= <_byte>
            <_byte> ::= b'9'  # Position 0x0004 (4)
        <char> ::= <byte>
          <byte> :
```

[55] https://www.fileformat.info/format/gif/egff.htm
[56] http://probablyprogramming.com/2009/03/15/the-tiniest-gif-ever

```
:= <_byte>
            <_byte> ::= b'a'  # Position 0x0005 (5)
  <LogicalScreenDescriptor> ::= <LOGICALSCREENDESCRIPTOR>
    <LOGICALSCREENDESCRIPTOR> ::= <Width> <Height> <PackedFields>
↪<BackgroundColorIndex> <PixelAspectRatio>  # b'\x01\x00\x01\x00\x80\x01\x00'
      <Width> ::= b'\x01' b'\x00'  # Position 0x0006 (6); b'\x01\x00'
      <Height> ::= b'\x01' b'\x00'  # Position 0x0008 (8); b'\x01\x00'
      <PackedFields> ::= <LOGICALSCREENDESCRIPTOR_PACKEDFIELDS>
        <LOGICALSCREENDESCRIPTOR_PACKEDFIELDS> ::= <GlobalColorTableFlag>
↪<ColorResolution> <SortFlag> <SizeOfGlobalColorTable>  # 0b10000000
          <GlobalColorTableFlag> ::= 1  # Position 0x000a (10), bit 7
          <ColorResolution> ::= 0 0 0  # Position 0x000a (10), bits 6-4; 0b000
          <SortFlag> ::= <bit>
            <bit> ::= <_bit>
              <_bit> ::= 0  # Position 0x000a (10), bit 3
          <SizeOfGlobalColorTable> ::= 0 0 0  # Position 0x000a (10), bits 2-0;␣
↪0b000
      <BackgroundColorIndex> ::= b'\x01'  # Position 0x000b (11)
      <PixelAspectRatio> ::= b'\x00'  # Position 0x000c (12)
  <GlobalColorTable> ::= <RGB> b'\x00' b'\x00' b'\x00'  # Position 0x000d (13); b'\
↪xff\xff\xff\x00\x00\x00'
    <RGB> ::= <R> <G> <B>  # b'\xff\xff\xff'
      <R> ::= <UBYTE>
        <UBYTE> ::= <ubyte>
          <ubyte> ::= <uchar>
            <uchar> ::= <unsigned_char>
              <unsigned_char> ::= <byte>
                <byte> ::= <_byte>
                  <_byte> ::= b'\xff'  # Position 0x0010 (16)
      <G> ::= <UBYTE>
        <UBYTE> ::= <ubyte>
          <ubyte> ::= <uchar>
            <uchar> ::= <unsigned_char>
              <unsigned_char> ::= <byte>
                <byte> ::= <_byte>
                  <_byte> ::= b'\xff'  # Position 0x0011 (17)
      <B> ::= <UBYTE>
        <UBYTE> ::= <ubyte>
          <ubyte> ::= <uchar>
            <uchar> ::= <unsigned_char>
              <unsigned_char> ::= <byte>
                <byte> ::= <_byte>
                  <_byte> ::= b'\xff'  # Position 0x0012 (18)
  <Data_1> ::= <DATA>
    <DATA> ::= <GraphicControlExtension> <ImageDescriptor> <LocalColorTable>
↪<ImageData>  # b'!\xf9\x04\x01\n\x00\x01\x00,\x00\x00\x00\x00\x01\x00\x01\x00\
↪x00\x02\x02L\x01\x00'
      <GraphicControlExtension> ::= <GRAPHICCONTROLEXTENSION>
        <GRAPHICCONTROLEXTENSION> ::= <ExtensionIntroducer_1> <GraphicControlLabel_
↪1> <GraphicControlSubBlock> <BlockTerminator_2>  # b'!\xf9\x04\x01\n\x00\x01\x00'
          <ExtensionIntroducer_1> ::= b'!'  # Position 0x0013 (19)
          <GraphicControlLabel_1> ::= b'\xf9'  # Position 0x0014 (20)
          <GraphicControlSubBlock> ::= <GRAPHICCONTROLSUBBLOCK>
            <GRAPHICCONTROLSUBBLOCK> ::= <BlockSize> <PackedFields_2> <DelayTime>
↪<TransparentColorIndex>  # b'\x04\x01\n\x00\x01'
              <BlockSize> ::= b'\x04'  # Position 0x0015 (21)
              <PackedFields_2> ::= <GRAPHICCONTROLEXTENSION_DATASUBBLOCK_
```

(continues on next page)

(continued from previous page)

```
↪PACKEDFIELDS>
                <GRAPHICCONTROLEXTENSION_DATASUBBLOCK_PACKEDFIELDS> ::= 0 0 0 0 0␣
↪0 0 1  # Position 0x0016 (22), bits 7-0; 0b00000001
             <DelayTime> ::= b'\n' b'\x00'  # Position 0x0017 (23); b'\n\x00'
             <TransparentColorIndex> ::= b'\x01'  # Position 0x0019 (25)
          <BlockTerminator_2> ::= b'\x00'  # Position 0x001a (26)
      <ImageDescriptor> ::= <IMAGEDESCRIPTOR>
        <IMAGEDESCRIPTOR> ::= <ImageSeparator_1> <ImageLeftPosition>
↪<ImageTopPosition> <ImageWidth> <ImageHeight> <PackedFields_1>  # b',\x00\x00\
↪x00\x00\x01\x00\x01\x00\x00'
          <ImageSeparator_1> ::= b','  # Position 0x001b (27)
          <ImageLeftPosition> ::= b'\x00' b'\x00'  # Position 0x001c (28); b'\x00\
↪x00'
          <ImageTopPosition> ::= b'\x00' b'\x00'  # Position 0x001e (30); b'\x00\
↪x00'
          <ImageWidth> ::= b'\x01' b'\x00'  # Position 0x0020 (32); b'\x01\x00'
          <ImageHeight> ::= b'\x01' b'\x00'  # Position 0x0022 (34); b'\x01\x00'
          <PackedFields_1> ::= 0 0 0 0 0 0 0 0  # Position 0x0024 (36), bits 7-0;␣
↪0b00000000

      <LocalColorTable> ::= b'\x02' b'\x02' b'L'  # Position 0x0025 (37); b'\x02\
↪x02L'
      <ImageData> ::= <IMAGEDATA>
        <IMAGEDATA> ::= <LZWMinimumCodeSize> <DataSubBlocks>  # b'\x01\x00'
          <LZWMinimumCodeSize> ::= b'\x01'  # Position 0x0028 (40)
          <DataSubBlocks> ::= b'\x00'  # Position 0x0029 (41)
 <Trailer> ::= <TRAILER>
   <TRAILER> ::= <GIFTrailer_1>
     <GIFTrailer_1> ::= b';'  # Position 0x002a (42)
```

CHAPTER

TWENTYFOUR

HATCHING SPECS

Fandango provides an `include()` function that you can use to *include* existing Fandango content. This allows you to distribute specifications over multiple files, defining *base* specs whose definitions can be further *refined* in specs that use them.

24.1 Including Specs with `include()`

Specifically, in a `.fan` file, a call to `include(FILE)`

1. Finds and loads `FILE` (typically *in the same location as the including file* (page 341))
2. Executes the *code* in `FILE`
3. Parses and adds the *grammar* in `FILE`
4. Parses and adds the *constraints* in `FILE`.

The `include()` function allows for *incremental refinement* of Fandango specifications - you can create some base `base.fan` spec, and then have more *specialized* specifications that alter grammar rules, add more constraints, or refine the code.

24.2 Incremental Refinement

Let us assume you have a *base* spec for a particular format, say, `base.fan`. Then, in a *refined* spec (say, `refined.fan`) that *includes* `base.fan`, you can

- override *grammar definitions*, by redefining rules;
- override *function and constant definitions*, by redefining them; and
- add additional *constraints*.

As an example, consider our `persons.fan` *definition of a name database* (page 31). We can create a more specialized version `persons50.fan` by including `persons.fan` and adding a *constraint* (page 45):

```
include('persons.fan')
where int(<age>) < 50
```

Likewise, we can create a specialized version `persons-faker.fan` that uses *fakers* (page 67) by overriding the `<first_name>` and `<last_name>` definitions:

```
from faker import Faker
fake = Faker()

include('persons.fan')

<first_name> ::= <name> := fake.first_name()
<last_name> ::= <name> := fake.last_name()
```

The *include* mechanism thus allows us to split responsibilities across multiple files:

- We can have one spec #1 with basic definitions of individual elements
- We can have a spec #2 that uses (includes) these basic definitions from spec #1 to define a *syntax*
- We can have a spec #3 that refines spec #2 to define a specific format for a particular program or device
- We can have a spec #4 that refines spec #3 towards a particular testing goal.

These mechanisms are akin to *inheritance* and *specialization* in object-oriented programming.

Tip

Generally, Fandango will warn about unused symbols, but not in an included `.fan` file.

24.3 Crafting a Library

If you create multiple specifications, you may wonder where best to store them. The *rules for where Fandango searches for included files* (page 341) are complex, but they boil down to two simple rules:

Tip

Store your included Fandango specs either

- in the directory where the *including* specs are, or
- in `$HOME/.local/share/fandango` (or `$HOME/Library/Fandango` on a Mac).

24.4 `include()` vs. `import`

Python provides its own import mechanism for referring to existing features. In general, you should use

- `import` whenever you want to make use of Python functions; and
- `include()` only if you want to make use of Fandango features.

Important

Using `include` for *pure Python code*, as in `include('code.py')` is not recommended. Most importantly, the current Fandango implementation will process "included" Python code only *after* all code in the "including" spec has been run. In contrast, the effects of `import` are immediate.

Part V

Checking Responses

CHAPTER

TWENTYFIVE

PARSING STRINGS

Fandango can also use its specifications to *parse* given strings and to *check* if they conform to the specification - both

- *syntactically* (according to the grammar); and
- *semantically* (according to the constraints).

Fandango uses this ability extensively for *checking outputs* (page 165) and *protocol testing* (page 233). In this section, we cover the basics of parsing.

25.1 The `parse` command

To parse an existing input, Fandango provides a `parse` command. Its arguments are any *files* to be parsed; if no files are given, `parse` reads from standard input. As with the `fuzz` command, providing a specification (with `-f FILE.fan`) is mandatory.

Let us use `parse` to check some dates against the *ISO 8601 format* (page 115) we have written a Fandango spec for. The command `echo -n` outputs the string given as argument (`-n` suppresses the newline it would normally produce); the pipe symbol `|` feeds this as input into Fandango:

```
$ echo -n '2025-01-27' | fandango parse -f iso8601.fan
```

If we do this, nothing happens. That is actually a good sign: it means that Fandango has successfully parsed the input.

If we pass an *invalid* input, however, Fandango will report this. This holds for *syntactically* invalid inputs:

```
$ echo -n '01/27/2025' | fandango parse -f iso8601.fan
```

```
FandangoParseError: '<stdin>': missing input at end of file
FandangoParseError: '<stdin>': missing input at end of file
Traceback (most recent call last):
  File "/Users/zeller/.virtualenvs/python3.13/bin/fandango", line 7, in <module>
    sys.exit(main())
             ~~~~^^
  File "<@beartype(fandango.cli.main) at 0x1097db060>", line 68, in main
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/__init__.py", line 47,␣
↪in main
    last_status = run(command, args)
  File "<@beartype(fandango.cli.commands.run) at 0x109e211c0>", line 51, in run
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 487,␣
↪in run
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
  File "<@beartype(fandango.logger.print_exception) at 0x1087868e0>", line 54, in␣
```

(continues on next page)

(continued from previous page)

```
↪print_exception
 File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
↪print_exception
    raise e
 File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 485,␣
↪in run
    command(args)
    ~~~~~~~^^^^^^
 File "<@beartype(fandango.cli.commands.parse_command) at 0x109e20900>", line 33,␣
↪in parse_command
 File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 270,␣
↪in parse_command
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
 File "<@beartype(fandango.logger.print_exception) at 0x1087868e0>", line 54, in␣
↪print_exception
 File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
↪print_exception
    raise e
 File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 267,␣
↪in parse_command
    tree = parse_file(fd, args, grammar, constraints, settings)
 File "<@beartype(fandango.cli.utils.parse_file) at 0x109824400>", line 136, in␣
↪parse_file
 File "/Users/zeller/Projects/Fandango!/src/fandango/cli/utils.py", line 329, in␣
↪parse_file
    raise FandangoParseError(
    ...<3 lines>...
    )
fandango.errors.FandangoParseError: '<stdin>': missing input at end of file
```

And also for *semantically* invalid inputs:

```
$ echo -n '2025-02-29' | fandango parse -f iso8601.fan
```

```
FandangoParseError: Did not match the following constraints: is_valid_
 ↪iso8601datetime(str(<iso8601datetime>))
FandangoParseError: Did not match the following constraints: is_valid_
 ↪iso8601datetime(str(<iso8601datetime>))
Traceback (most recent call last):
  File "/Users/zeller/.virtualenvs/python3.13/bin/fandango", line 7, in <module>
    sys.exit(main())
             ~~~~^^
  File "<@beartype(fandango.cli.main) at 0x109eab060>", line 68, in main
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/__init__.py", line 47,␣
↪in main
    last_status = run(command, args)
  File "<@beartype(fandango.cli.commands.run) at 0x10a4f11c0>", line 51, in run
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 487,␣
↪in run
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
  File "<@beartype(fandango.logger.print_exception) at 0x108e568e0>", line 54, in␣
↪print_exception
  File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
↪print_exception
```

(continues on next page)

(continued from previous page)

```
    raise e
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 485,␣
↪in run
    command(args)
    ~~~~~~~^^^^^^
  File "<@beartype(fandango.cli.commands.parse_command) at 0x10a4f0900>", line 33,␣
↪in parse_command
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 270,␣
↪in parse_command
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
  File "<@beartype(fandango.logger.print_exception) at 0x108e568e0>", line 54, in␣
↪print_exception
  File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
↪print_exception
    raise e
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 267,␣
↪in parse_command
    tree = parse_file(fd, args, grammar, constraints, settings)
  File "<@beartype(fandango.cli.utils.parse_file) at 0x109ef4400>", line 136, in␣
↪parse_file
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/utils.py", line 324, in␣
↪parse_file
    raise FandangoParseError(
        f"Did not match the following constraints: {', '.join(failed_constraints)}"
    )
fandango.errors.FandangoParseError: Did not match the following constraints: is_
↪valid_iso8601datetime(str(<iso8601datetime>))
```

In both cases, the return code will be non-zero:

```
$ echo $?
1
```

25.2 Validating Parse Results

By default, the `parse` command produces no output. However, to inspect the parse results, you can output the parsed string again. The `-o FILE` option writes the parsed string to `FILE`, with `-` being the standard output.

```
$ echo -n '2025-01-27' | fandango parse -f iso8601.fan -o -
```

```
2025-10-27
```

We see that input and output are identical (as should always be with parsing and unparsing).

> **Tip**
>
> As it comes to producing and storing outputs, the `parse` command has the same options as the `fuzz` command.

Since parsing and unparsing should always be symmetrical to each other, Fandango provides a `--validate` option to run this check automatically:

```
$ echo -n '2025-01-27' | fandango parse -f iso8601.fan --validate
```

Again, if nothing happens, then the (internal) check was successful.

The `--validate` option can also be passed to the `fuzz` command; here, it ensures that the produced string can be parsed by the same grammar (again, as should be).

Important

If you find that `--validate` fails, please report this as a Fandango bug.

25.3 Alternate Output Formats

In order to debug grammars, Fandango provides a number of *alternate* formats in which to output the parsed tree, controlled by the `--format` flag.

25.3.1 String

The option `--format=string` outputs the parsed tree as a string. This is the default.

```
$ echo -n '2025-01-27' | fandango parse -f iso8601.fan -o - --format=string
```

```
2025-10-27
```

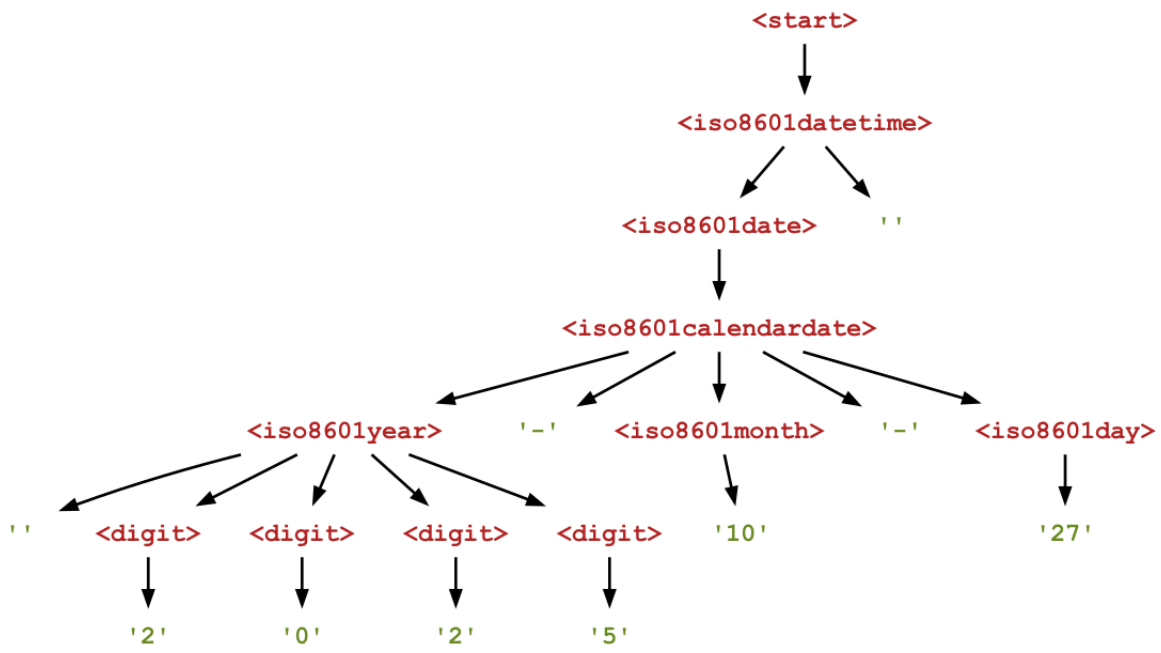
25.3.2 Tree

The option `--format=string` outputs the parsed tree as a Python `Tree()` expression. This is useful for evaluating and visualizing the tree.

```
$ echo -n '2025-01-27' | fandango parse -f iso8601.fan -o - --format=tree
```

```
Tree(<start>, Tree(<iso8601datetime>, Tree(<iso8601date>, Tree(
 ↪<iso8601calendardate>,
  Tree(<iso8601year>,
    Tree(<digit>, Tree(<_digit>, Tree('2'))),
    Tree(<digit>, Tree(<_digit>, Tree('0'))),
    Tree(<digit>, Tree(<_digit>, Tree('2'))),
    Tree(<digit>, Tree(<_digit>, Tree('5')))
  ),
  Tree('-'),
  Tree(<iso8601month>, Tree('10')),
  Tree('-'),
  Tree(<iso8601day>, Tree('27'))
))))
```

Here comes this tree, visualized:



25.3.3 Grammar

The option `--format=grammar` outputs the parsed tree as a (highly specialized) grammar, in which children are indented under their respective parents. This is useful for debugging, but also for creating a grammar from a sample file and then generalizing it.

```
$ echo -n '2025-01-27' | fandango parse -f iso8601.fan -o - --format=grammar
```

```
<start> ::= <iso8601datetime>
  <iso8601datetime> ::= <iso8601date>
    <iso8601date> ::= <iso8601calendardate>
      <iso8601calendardate> ::= <iso8601year> '-' <iso8601month> '-' <iso8601day>
↪# Position 0x0000 (0); 2025-10-27
       <iso8601year> ::= <digit> <digit> <digit> <digit>  # 2025
         <digit> ::= <_digit>
           <_digit> ::= '2'  # Position 0x0002 (2)
         <digit> ::= <_digit>
           <_digit> ::= '0'  # Position 0x0003 (3)
         <digit> ::= <_digit>
           <_digit> ::= '2'  # Position 0x0004 (4)
         <digit> ::= <_digit>
           <_digit> ::= '5'  # Position 0x0005 (5)
       <iso8601month> ::= '10'  # Position 0x0006 (6)
       <iso8601day> ::= '27'  # Position 0x0008 (8)
```

25.3.4 Bits

The option `--format=bits` outputs the parsed tree as a bit sequence.

```
$ echo -n '2025-01-27' | fandango parse -f iso8601.fan -o - --format=bits
```

```
00110010000110000001100100011010100101110100110001001100000010110100110010001101111
```

This is useful for debugging *binary formats* (page 129) that contain *bits* (page 137).

CHAPTER

TWENTYSIX

CHECKING OUTPUTS

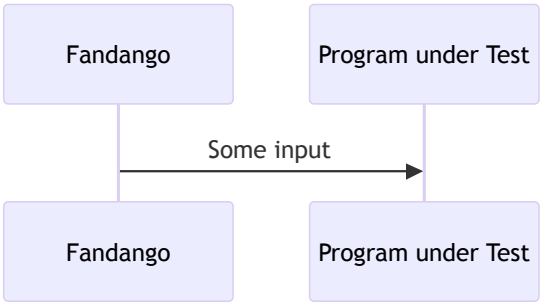
Since Fandango makes use of specifications both to produce (page 31) and to parse (page 159) strings, it can actually *combine both* to

1. first send an input to a program under test; and
2. then parse its output to check if it produced the correct result.

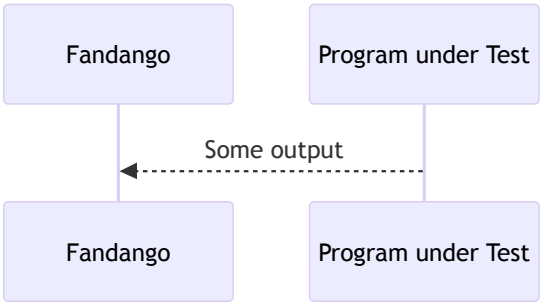
For this purpose, Fandango provides a means to combine both input and output in a *single specification*, used by the Fandango `talk` command. Let us see how this works.

26.1 Interaction Testing

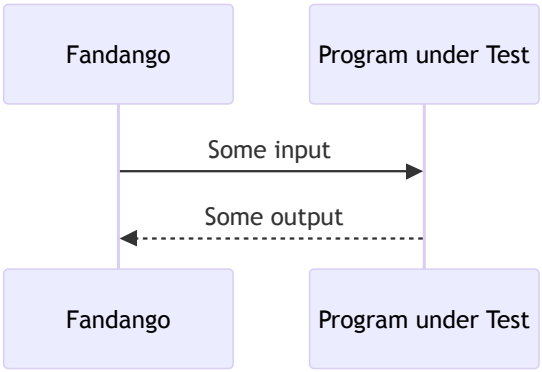
So far, we have only considered two settings. In *fuzzing*, Fandango sends a synthesized input to the program under test:



During *parsing*, Fandango accepts and processes *outputs* from the program under test:



What we want, though, is an *interaction* - a means to *first* send an input to the program, and then parse its output to check it:



For this, we need to specify which parts of the interaction are supposed to be *sent* or *received* by which *party* in the interaction.

26.2 Simple Input/Output Testing

Fandango allows a simple means to combine inputs and outputs in a single `.fan` specification. The key idea is to *identify individual nonterminals with the party that is supposed to produce it*. In Fandango, this is done by prefixing the nonterminal with a party name (an identifier), followed by a colon (`:`). Hence, a nonterminal `<fandango:string>` refers to a `<string>` element that would be produced by a party named `fandango` (whoever that might be).

Fandango conveniently defines two standard parties:

- `In` refers to the standard input of the program under test; and
- `Out` refers to the standard output of the program under test.

Hence, in a Fandango spec, `<In:id>` refers to an `<id>` element that is *received* (or input) by the program, and `<Out:result>` is a `<result>` element that is *sent* (or output) by the program.

Important

Remember that `In` and `Out` describe the interaction from the *perspective of the program under test*.

With this, we can already write a first specification.

`cat` is an abbreviation for "concatenate". `cat` can take multiple inputs, and concatenates them as one.

The UNIX `cat` command accepts some input, and outputs this very input unchanged. In Fandango, this interaction can be described in a file `cat.fan` as follows:

```
<start> ::= <In:input> <Out:output>
<input> ::= <line>
<output> ::= <line>
<line> ::= r'.*\n'
```

In this specification,

- `<input>` and `<output>` define the inputs and outputs of `cat`, respectively, as a `<string>`; and
- `<string>` defines a regular expression standing for any sequence of characters, including newlines.

Let us use Fandango with this spec to test the `cat` program.

26.3 The Fandango `talk` command

Fandango provides a `talk` command that allows testing interactions. Like `fuzz` and `parse`, it takes as argument a `-f` option, followed by a `.fan` file; however, this one must contain party specifications. The remainder of the command line is the program to be tested (possibly with arguments).

In our case, this is how the invocation of Fandango looks like:

```
$ fandango talk -f cat.fan cat
```

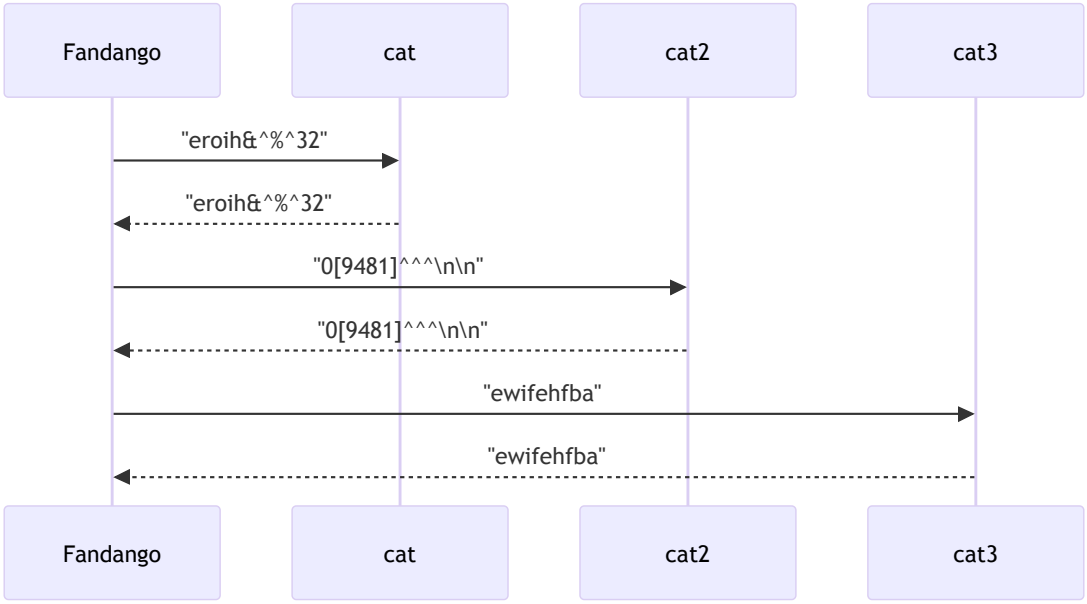
This command does not issue any outputs (all of them are being sent to `cat`), but here is what is happening behind the scenes:

- The `cat` command sends back the input via its output;
- Fandango receives and parses the `cat` output.

We can also specify multiple interactions, as in

```
$ fandango talk -f cat.fan -n 10 cat
```

Now, each time, `cat` is started anew, as shown in this diagram:


Fandango
cat
cat2
cat3
"eroih&^%^32"
"eroih&^%^32"
"0[9481]^^^\n\n"
"0[9481]^^^\n\n"
"ewifehfba"
"ewifehfba"
Fandango
cat
cat2
cat3

Note

Once a communication party is set for a nonterminal, it need not be repeated for its constituents. In the above example, we can define `<input>` as `<string>` without restating the `In:` prefix; from the first line, it is clear that `<input>` comes from `In`. Also, this allows multiple parties to share the same elements (such as `<string>`).

26.4 Oracles

So far, our `.fan` specification has not really checked whether `cat` operates correctly. It does check the `cat` output against the `<string>` regular expression - but that is a "match-all" expression, meaning that anything is valid.

To check whether the `cat` output is correct, we must compare it against the input we sent and ensure that input and output are identical. For this, *constraints* (page 45) are the ideal tool, as they allow us to reference arbitrary elements in the entire interaction. In our case, this simple constraint would suffice:

```
str(<input>) == str(<output>)
```

This constraint defines the full behavior of `cat`; it acts as an *oracle* that determines whether the behavior of the program under test is correct or not.

Let us add this constraint using a `where` clause to `cat.fan`, resulting in `cat-oracle.fan`:

```
<start> ::= <In:input> <Out:output>
<input> ::= <lines>
<output> ::= <lines>
<lines> ::= <line>+
<line> ::= r'.*\n'
where str(<input>) == str(<output>)
```

Again, we can test, and normally, nothing should happen.

```
$ fandango talk -f cat-oracle.fan cat
```

The first C implementation of `cat`[57] had 80 lines of code. A detailed history of `cat`[58] is available.

So far, we have mostly seen constraints as a *precondition* - that is, a condition that makes inputs valid in the first place. Our constraint here acts as a *postcondition* – that is, a condition that checks the output, possibly based on earlier input features.

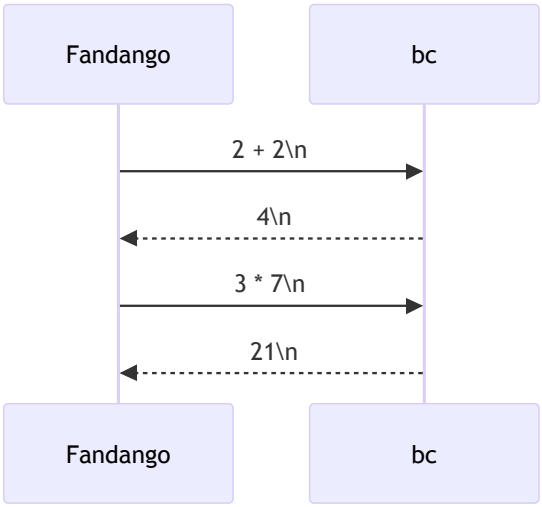
[57] https://gist.github.com/sinclairtarget/47143ba52b9d9e360d8db3762ee0cbf5#file-3-cat-v7-c
[58] https://twobithistory.org/2018/11/12/cat.html

26.5 More Complex Interactions

Let us now test a program whose interaction scheme is a bit more complex. The UNIX `bc` command accepts a line with an arithmetic expression, and then produces the result. It keeps on doing so until the input ends. To compute `2 + 2`, we can enter

```
$ bc
>>> 2 + 2
4
>>> (Ctrl-D)
```

Here, `>>>` is the prompt of the `bc` program; it goes to `stderr` and is only produced in interactive settings, so we can ignore it. A typical interaction between Fandango and `bc` would thus look like this:



Let us define a sequence of 10 interactions with `bc`. Using our earlier *expression grammar* (page 81) `expr.fan`, we can define such interactions in a `.fan` spec `bc.fan`:

```
include('expr.fan')

<start> ::= <interaction>
<interaction> ::= <In:input> <Out:output>
<input> ::= <expr> '\n'
<output> ::= <int> '\n'
```

We see that the `<input>` now is an expression; and the expected `<output>` is an integer. This is how we can test `bc`:

```
$ fandango talk -f bc.fan bc
```

Our `.fan` spec checks that the `bc` indeed produces integers, but it does not check whether the result is correct, too. How would one do this? (Hint: use the Python `eval()` function.)

Solution

Add a constraint that *evaluates* the expression (in Python) and compares it against the `bc` result.

```
where eval(str(<input>)) == int(<output>)
```

If we actually do this, we will find that there are a few differences between the way that Python and `bc` interpret expressions:

```
$ fandango talk -f bc.fan -n 1 -c 'eval(str(<input>)) == int(<output>)' bc
```

```
(FandangoFailedError(...), 'Timed out while waiting for message from remote party.␣
↪Expected message from party: Out')
```

To ensure complete testing, we need to

- avoid + and − prefixes; these are not understood by `bc`;
- avoid leading zeros in numbers; these are not permitted in Python;
- allow small differences between floating point numbers, or restrict ourselves to integer operations.

Right now, we leave this as an exercise to the reader :-)

26.6 Testing Strategies

If you find that checking results is complicated, welcome to the world of testing! Specifically, you have just encountered the *oracle problem* - the effort in specifying what a correct result should be. While Fandango makes it easy to *produce* inputs and to *decompose* outputs, the burden of specification is still on you.

Here are some established techniques to ease the oracle problem:

Compare against a different implementation.
: By using Python `eval()`, above, we already make our lives much easier. However, we could also compare against, say, a different `bc` implementation. This is called *differential testing*.

Compare against a different version.
: After having made a change to a program, we can check it against an older version to make sure there are no unexpected changes in behavior. This is called *regression testing*.

Compare the result of equivalent inputs.
: Send two inputs to a program that should produce the same result and check for differences. In the case of `bc`, for instance, any term `<a> + <b>` should yield the same result as `<b> + <a>`. This is called *metamorphic testing*.

Under Construction

Future versions of this tutorial will further detail these strategies and how to integrate them into Fandango.

26.7 Troubleshooting Interactions

Since interactions are always being sent to some party, and since the party outputs are being processed by Fandango, it may not always be easy to track which data is being sent, and where.

However, you can also make use of interaction specs in the regular `fuzz` and `parse` commands. The special `--party=PARTY` option allows you to produce outputs or parse inputs for just one given party `PARTY` in the interaction. The effect of `--party` is that it *excludes* all other parties from the interaction, allowing to produce or parse strings for just one party.

As an example, consider again our `bc.fan` example:

```
include('expr.fan')

<start> ::= <interaction>
<interaction> ::= <In:input> <Out:output>
<input> ::= <expr> '\n'
<output> ::= <int> '\n'
```

This is the effect of `--party=In`. See how the `Out:` part of the interaction has been excluded, also excluding `<output>` from production:

These are actually produced using *`fandango convert`* (page 112) with the `--party` option.

```
BeartypeCallHintParamViolation: Function fandango.language.parse.slice_parties()␣
 ↪parameter parties=['In'] violates type hint set[str], as list ['In'] not␣
 ↪instance of set.
Traceback (most recent call last):
  File "/Users/zeller/.virtualenvs/python3.13/bin/fandango", line 7, in <module>
    sys.exit(main())
             ~~~~^^
  File "<@beartype(fandango.cli.main) at 0x10974f060>", line 68, in main
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/__init__.py", line 47,␣
 ↪in main
    last_status = run(command, args)
  File "<@beartype(fandango.cli.commands.run) at 0x109d951c0>", line 51, in run
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 487,␣
 ↪in run
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
  File "<@beartype(fandango.logger.print_exception) at 0x1086fa8e0>", line 54, in␣
 ↪print_exception
  File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
 ↪print_exception
    raise e
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 485,␣
```

(continues on next page)

(continued from previous page)

```
↪in run
    command(args)
    ~~~~~~~^^^^^^
  File "<@beartype(fandango.cli.commands.convert_command) at 0x109d94b80>", line␣
↪33, in convert_command
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 405,␣
↪in convert_command
    spec = converter.to_fan()
  File "<@beartype(fandango.converters.fan.FandangoFandangoConverter.
↪FandangoFandangoConverter.to_fan) at 0x109d4b7e0>", line 12, in to_fan
  File "/Users/zeller/Projects/Fandango!/src/fandango/converters/fan/
↪FandangoFandangoConverter.py", line 20, in to_fan
    parsed_spec = parse_spec(
        contents, filename=self.filename, use_cache=False, parties=self.parties
    )
  File "<@beartype(fandango.language.parse.parse_spec) at 0x109630680>", line 123,␣
↪in parse_spec
  File "/Users/zeller/Projects/Fandango!/src/fandango/language/parse.py", line 430,
↪ in parse_spec
    slice_parties(spec.grammar, parties)
    ~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^
  File "<@beartype(fandango.language.parse.slice_parties) at 0x109631120>", line␣
↪57, in slice_parties
beartype.roar.BeartypeCallHintParamViolation: Function fandango.language.parse.
↪slice_parties() parameter parties=['In'] violates type hint set[str], as list [
↪'In'] not instance of set.
```

This is the effect of `--party=Out`, excluding the `In` part, and consequently, `<input>`:

```
BeartypeCallHintParamViolation: Function fandango.language.parse.slice_parties()␣
↪parameter parties=['Out'] violates type hint set[str], as list ['Out'] not␣
↪instance of set.
Traceback (most recent call last):
  File "/Users/zeller/.virtualenvs/python3.13/bin/fandango", line 7, in <module>
    sys.exit(main())
             ~~~~^^
  File "<@beartype(fandango.cli.main) at 0x1081ef060>", line 68, in main
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/__init__.py", line 47,␣
↪in main
    last_status = run(command, args)
  File "<@beartype(fandango.cli.commands.run) at 0x1088311c0>", line 51, in run
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 487,␣
↪in run
    print_exception(e)
    ~~~~~~~~~~~~~~~^^^
  File "<@beartype(fandango.logger.print_exception) at 0x1071968e0>", line 54, in␣
↪print_exception
  File "/Users/zeller/Projects/Fandango!/src/fandango/logger.py", line 48, in␣
↪print_exception
    raise e
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 485,␣
↪in run
    command(args)
    ~~~~~~~^^^^^^
  File "<@beartype(fandango.cli.commands.convert_command) at 0x108830b80>", line␣
↪33, in convert_command
  File "/Users/zeller/Projects/Fandango!/src/fandango/cli/commands.py", line 405,␣
```

(continues on next page)

(continued from previous page)

```
↪in convert_command
    spec = converter.to_fan()
  File "<@beartype(fandango.converters.fan.FandangoFandangoConverter.
↪FandangoFandangoConverter.to_fan) at 0x1087e37e0>", line 12, in to_fan
  File "/Users/zeller/Projects/Fandango!/src/fandango/converters/fan/
↪FandangoFandangoConverter.py", line 20, in to_fan
    parsed_spec = parse_spec(
        contents, filename=self.filename, use_cache=False, parties=self.parties
    )
  File "<@beartype(fandango.language.parse.parse_spec) at 0x1080cc680>", line 123,␣
↪in parse_spec
  File "/Users/zeller/Projects/Fandango!/src/fandango/language/parse.py", line 430,
↪ in parse_spec
    slice_parties(spec.grammar, parties)
    ~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^
  File "<@beartype(fandango.language.parse.slice_parties) at 0x1080cd120>", line␣
↪57, in slice_parties
beartype.roar.BeartypeCallHintParamViolation: Function fandango.language.parse.
↪slice_parties() parameter parties=['Out'] violates type hint set[str], as list [
↪'Out'] not instance of set.
```

Typically, you provide such a `--party` option directly as part of some `fuzz` or `parse` command. To see what typical inputs to `bc` look like, use:

The `--format=value` option makes the strings readable.

```
$ fandango fuzz -f bc.fan --party=In -n 10 --format=value
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

(continues on next page)

(continued from previous page)

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

(continues on next page)

(continued from previous page)

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

(continues on next page)

(continued from previous page)

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:ERROR: Population did not converge to a perfect population
fandango:ERROR: Only found 1 perfect solutions, instead of the required 10
```

Conversely, to see what typical outputs from `bc` would be expected, use:

```
$ fandango fuzz -f bc.fan --party=Out -n 10 --format=value
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals. Population size reduced to 1.
```

(continues on next page)

(continued from previous page)

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

(continues on next page)

(continued from previous page)

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
↪Population size reduced to 1.
fandango:ERROR: Population did not converge to a perfect population
fandango:ERROR: Only found 1 perfect solutions, instead of the required 10
```

We make use of such interactions extensively in the next part on *Testing protocols* (page 233).

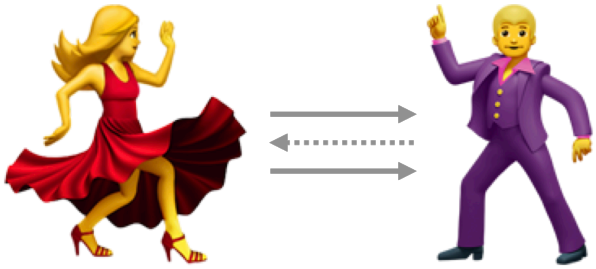
Part VI

Testing Protocols

CHAPTER

TWENTYSEVEN

TESTING PROTOCOLS



In *the chapter on checking outputs* (page 165), we already have seen how to interact with external programs. In this chapter, we will extend this concept to full *protocol testing* across networks.

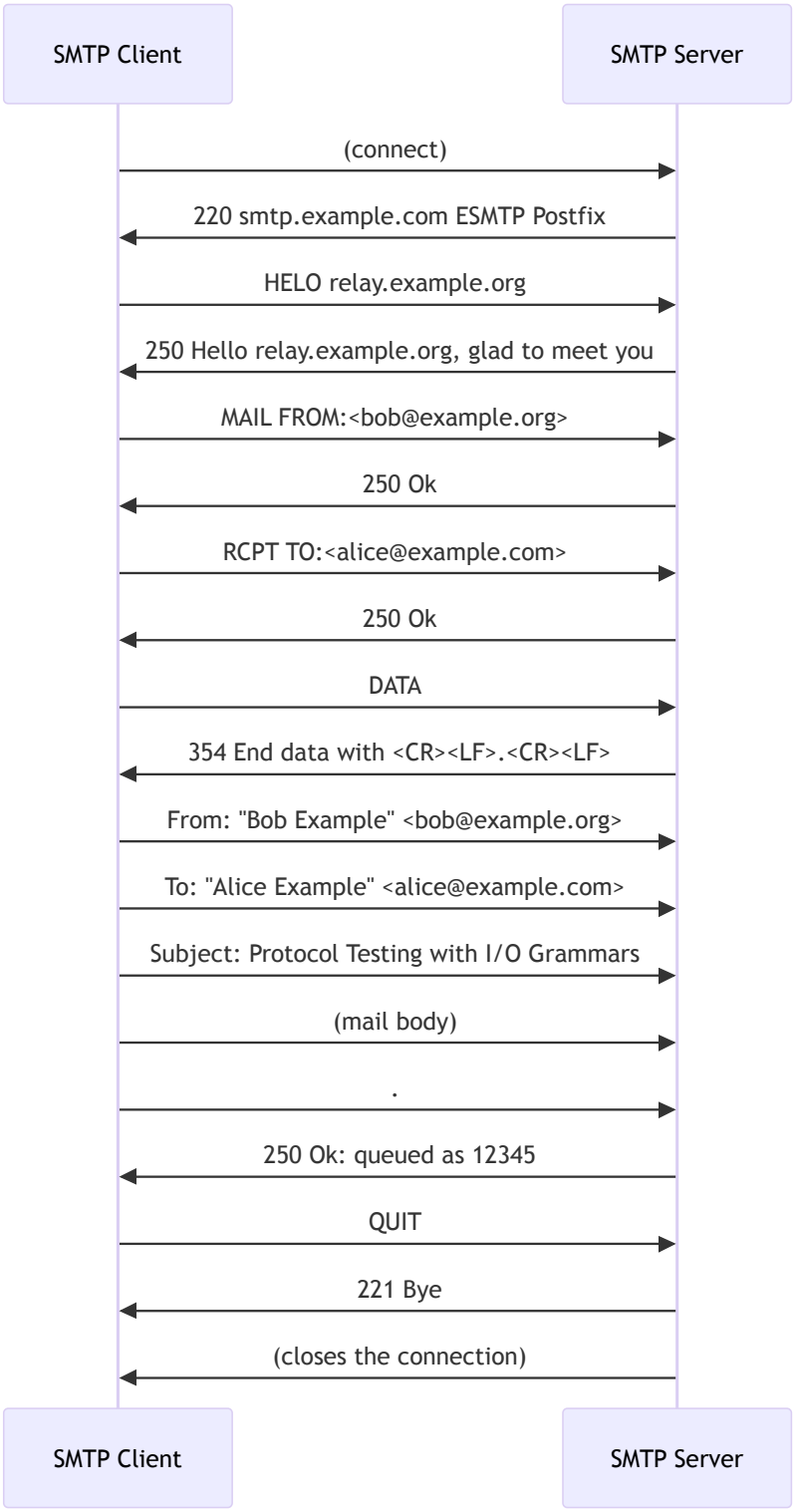
Fandango can test protocols. In particular, it can

- act as a network *client* and interact with network *servers*; and
- act as a network *server* and interact with network *clients*.

27.1 Interacting with an SMTP server

Let us start with a simple example. The Simple Mail Transfer Protocol[59] (SMTP, RFC 821[60]) is a protocol through which mail clients can connect to a server to send mail to recipients. A typical interaction with an SMTP server[61] `smtp.example.com`, sending a mail from `bob@example.org` to `alice@example.com`, is illustrated below:

[59] https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
[60] https://datatracker.ietf.org/doc/html/rfc821
[61] https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol



Our job will be to *automate this interaction* using Fandango. For this, we need two things:

1. An SMTP server to send commands to

2. A `.fan` spec that encodes this interaction.

27.2 An SMTP server for experiments

For illustrating protocol testing, we need to run an SMTP server, which we will run locally on our machine. (No worries - the local SMTP server can not actually send mails across the Internet.)

The Python `aiosmtpd` server will do the trick:

```
$ pip install aiosmtpd
```

Once installed, we can run the server locally. Normally, it runs on port 8025:

```
$ python -m aiosmtpd -d -n &
```

We can now connect to the server on the given port and send it commands. The `telnet` command is handy for this. We give it a hostname (`localhost` for our local machine) and a port (8025 for our local SMTP server.)

Once connected, anything we type into the `telnet` input will automatically be relayed to the given port, and hence to the SMTP server. For instance, entering a `QUIT` command (followed by Return) into telnet will be forwarded to the SMTP server, which will terminate the connection:

```
$ telnet localhost 8025
```

```
Trying localhost...
Connected to localhost.
Escape character is '^]'.
220 smtp.example.com Python SMTP 1.4.6
QUIT
221 Bye
Connection closed by foreign host.
```

Try this for yourself! What happens if you invoke `telnet`, introducing yourself with `HELO client.example.org`?

Solution

When sending `HELO client.example.org`, the server replies with its own hostname. This is the name of the local computer, in our example `smtp.example.com`.

```
$ telnet localhost 8025
Trying localhost...
Connected to localhost.
Escape character is '^]'.
220 smtp.example.com Python SMTP 1.4.6
HELO client.example.org
250 smtp.example.com
QUIT
221 Bye
Connection closed by foreign host.
```

> **Tip**
>
> To interrupt a telnet session, type the escape character (`CTRL + ]`). At the `telnet>` prompt, enter `help` for possible commands; `quit` exits the telnet program.

27.3 A simple SMTP grammar

We use `telnet` only for illustrative purposes here; later in this chapter, you will see how to have Fandango directly connect to servers (and clients!)

With *`fandango talk`* (page 165), we have seen a Fandango facility that allows us to connect to the standard input and output channels of a given program and interact with it. The idea would now be to use the `telnet` program for this very purpose. By invoking

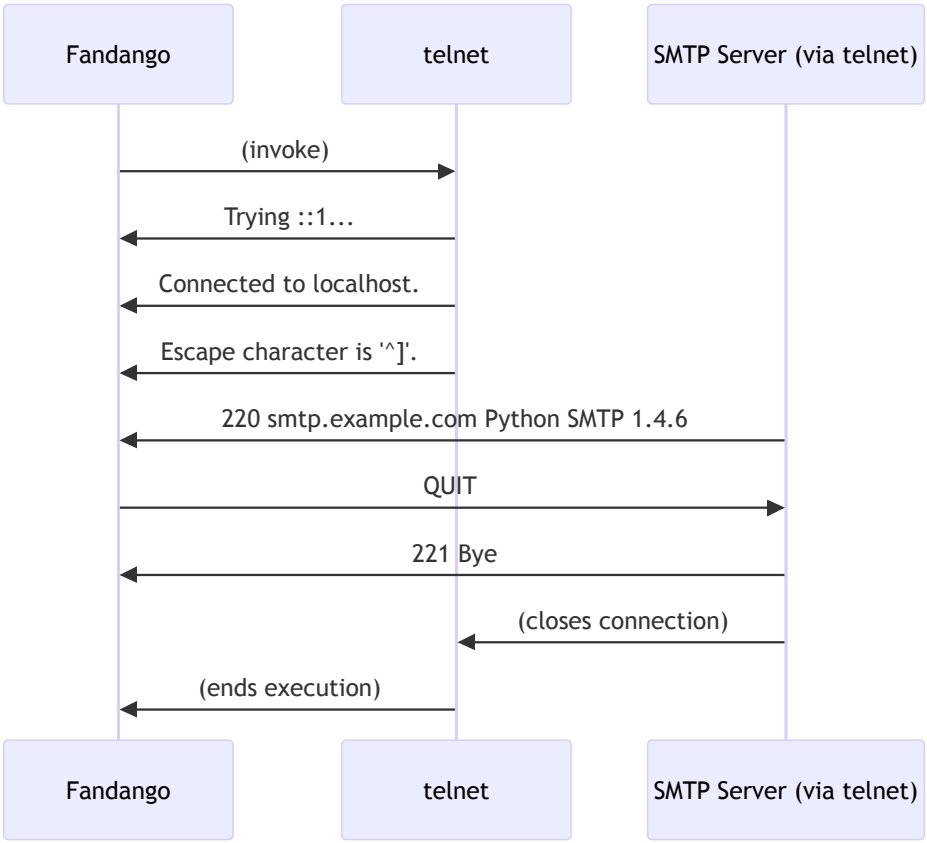
```
$ fandango talk -f smtp-telnet.fan telnet localhost 8025
```

we could interact with the `telnet` program as described above. All we now need is a grammar that describes the `telnet` interaction.

The following grammar has two parts:

1. First, we expect some output from the `telnet` program.
2. Then, we interact with the SMTP server - just sending a `QUIT` command and then exiting.

A typical interaction thus would be:



The following I/O grammar smtp-telnet.fan implements this interaction via `telnet`:

1. First, `<telnet-intro>` lets Fandango expect the `telnet` introduction;

2. Then, <smtp> takes care of the actual SMTP interaction.

```
<start> ::= <Out:telnet_intro> <smtp>
<telnet_intro> ::= \
    r"Trying.*" "\n" \
    r"Connected.*" "\n" \
    r"Escape.*" "\n"

<smtp> ::= <Out:m220> <In:quit> <Out:m221>
<m220> ::= "220 " r".*" "\n"
<quit> ::= "QUIT\n"
<m221> ::= "221 " r".*" "\n"
```

Note

Again, note that In and Out describe the interaction from the *perspective of the program under test*; hence, Out is what telnet and the SMTP server produce, whereas In is what the SMTP server (and telnet) get as input.

With this, we can now connect to our (hopefully still running) SMTP server and actually send it a QUIT command:

The -n 1 option limits fuzzing to one interaction. Without -n 1, Fandango keeps on generating inputs until *grammar coverage* (page 113) is achieved.

```
$ fandango talk -f smtp-telnet.fan -n 1 telnet localhost 8025
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

To track the data that is actually exchanged, use the verbose -v flag. The In: and Out: log messages show the data that is being exchanged.

```
$ fandango -v talk -f smtp-telnet.fan -n 1 telnet localhost 8025
```

```
fandango:INFO: ---------- Parsing FANDANGO content ----------
```

```
fandango:INFO: <stdlib>: loading cached spec from /Users/zeller/Library/Caches/
 ↪Fandango/6ce667e6ac3405c5ae9aee6e21c982a086df69bc50a44ec8794a3ccad7b79284.pickle
fandango:INFO: smtp-telnet.fan: loading cached spec from /Users/zeller/Library/
 ↪Caches/Fandango/e47138be5dd67cc443bdbd7e72e1ffc9b5c92e37c458688f5ca83769352cf692.
 ↪pickle
fandango:INFO: <string>: loading cached spec from /Users/zeller/Library/Caches/
 ↪Fandango/8f969ea485a31fc394a2d0174f66c9786c1bdcc0779f8e190b2b9c811138470e.pickle
fandango:INFO: Setting command ['telnet', 'localhost', '8025']
fandango:INFO: Starting subprocess with command ['telnet', 'localhost', '8025']
fandango:INFO: File mode: text
fandango:INFO: ---------- Initializing base population ----------
fandango:INFO: ---------- Initializing FANDANGO algorithm ----------
fandango:INFO: ---------- Done initializing base population ----------
fandango:INFO: ---------- Generating  for 500 generations----------
fandango:INFO: Generating (additional) initial population (size: 100)...
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:INFO: Initial population generated in 0.03 seconds
fandango:INFO: Current coverage: 1.96%
```

```
fandango:INFO: Out: <telnet_intro> "Trying ::1...\nConnected to localhost.\nEscape␣
 ↪character is '^]'.\n"
fandango:INFO: Current coverage: 29.41%
fandango:INFO: Out: <m220> '220 dyn-010017130244.sci.c0.cispa.net Python SMTP 1.4.
 ↪6\n'
```

```
fandango:INFO: Current coverage: 60.78%
fandango:INFO: In: <quit> 'QUIT\n'
fandango:INFO: Current coverage: 74.51%
fandango:INFO: Out: <m221> '221 Bye\n'
```

```
fandango:INFO: Current coverage: 100.00%
```

Changed in version 1.1: As of version 1.1, Fandango by default

- keeps on generating interactions until stopped (or limited by the `-n` option)
- aims for *grammar coverage* (page 113), thus covering states and message alternatives

27.4 Interacting as Network Client

Using `telnet` to communicate with servers generally works, but it has a number of drawbacks. Most importantly, `telnet` is meant for *human* interaction. Hence, our I/O grammars have to reflect the `telnet` output (which actually might change depending on operating system and configuration); also, `telnet` is not suited for transmitting binary data.

Fortunately, Fandango offers a means to be invoked *directly as a network client*, not requiring external programs such as `telnet`. The `fandango talk` option `--client` allows Fandango to be used as a network client. The argument to `--client` is a network address to connect to. In the simplest form, it is just a port number on the local machine. Hence, to have Fandango act as an SMTP client for the local server, we would use the option `--client 8025`.

Since Fandango directly talks to the SMTP server now, we can also simplify the grammar by removing the `<telnet_intro>` part. Also, there is no more `In` and `Out` parties, since we do not interact with the standard input and output of an invoked program. Instead,

- `Client` is the party representing the *client*, connecting to an external server on the network.
- `Server` is the party representing a *server* on the network, accepting connections from clients.

Consequently,

- all outputs produced by the *client* (and processed by the server) are prefixed with `Client:` in the respective nonterminals; and
- all outputs produced by the *server* (and processed by the client) are prefixed with `Server:`.

With this, we can reframe and simplify our SMTP grammar, using `Client` and `Server` to describe the respective interactions. The spec `smtp-simple.fan` reads as follows:

```
<start> ::= <smtp>
<smtp> ::= <Server:m220> <Client:quit> <Server:m221>
<m220> ::= "220 " <hostname> " " r".*" "\r\n"
<quit> ::= "QUIT\r\n"
```

(continues on next page)

(continued from previous page)

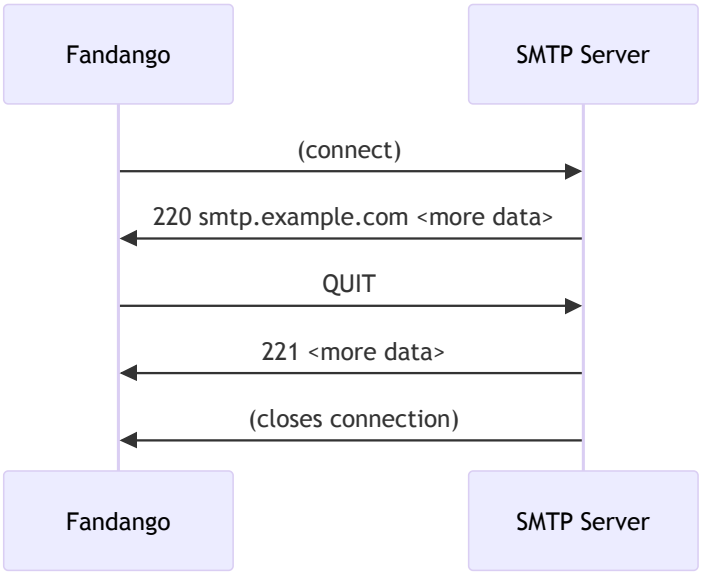
```
<m221> ::= "221 " r".*" "\r\n"

<hostname> ::= r"[-a-zA-Z0-9.:]*" := "host.example.com"
```

Note how we added <hostname> as additional specification of the hostname that is typically part of the initial server message.

With this, we have Fandango act as client and connect to the (hopefully still running) server on port 8025:

```
$ fandango talk -f smtp-simple.fan -n 1 --client 8025
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```



From here on, we can have Fandango directly "talk" to network components such as servers.

27.5 Interacting as Network Server

Obviously, our SMTP specification is still very limited. Before we go and extend it, let us first highlight a particular Fandango feature. From the same specification, Fandango can act as a *client* and as a *server*. When invoked with the `--server` option, Fandango will *create* a server at the given port and accept client connections. So if we invoke

The option `-n 100` ensures the server will run for 100 interactions.

```
$ fandango talk -f smtp-simple.fan -n 100 --server 8125
```

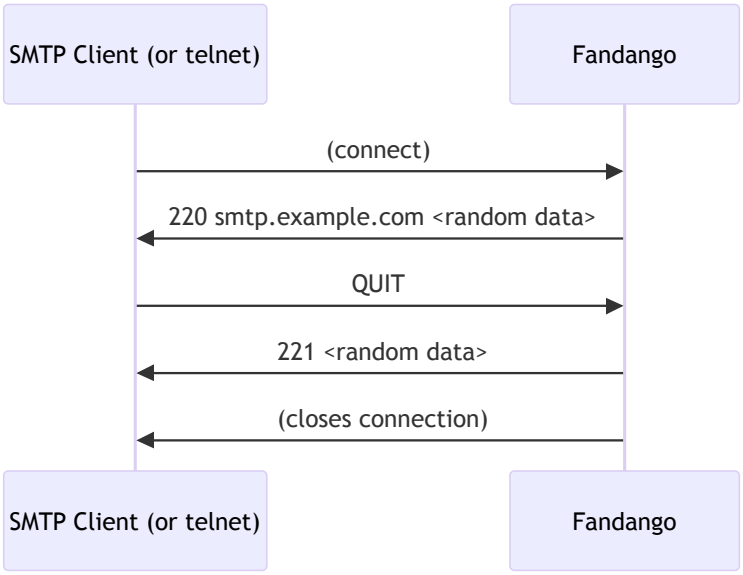
```
fandango:WARNING: Fandango IO only supports desired-solution values of 1 for now,␣
 ↪overriding value
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

we can then connect to our running Fandango "SMTP Server" and interact with it according to the `smtp-simple.fan` spec:

```
$ telnet localhost 8125
```

```
Trying localhost...
Connected to localhost.
Escape character is '^]'.
220 host.example.com 9Ms1iQ
QUIT
221 H^D$
Connection closed by foreign host.
```

Note that as server, Fandango produces its own `220` and `221` messages, effectively *fuzzing the client*. Note how the interaction diagram reflects how Fandango is now taking the role of the client:


SMTP Client (or telnet)
Fandango
(connect)
220 smtp.example.com <random data>
QUIT
221 <random data>
(closes connection)
SMTP Client (or telnet)
Fandango

27.6 A Bigger Protocol Spec

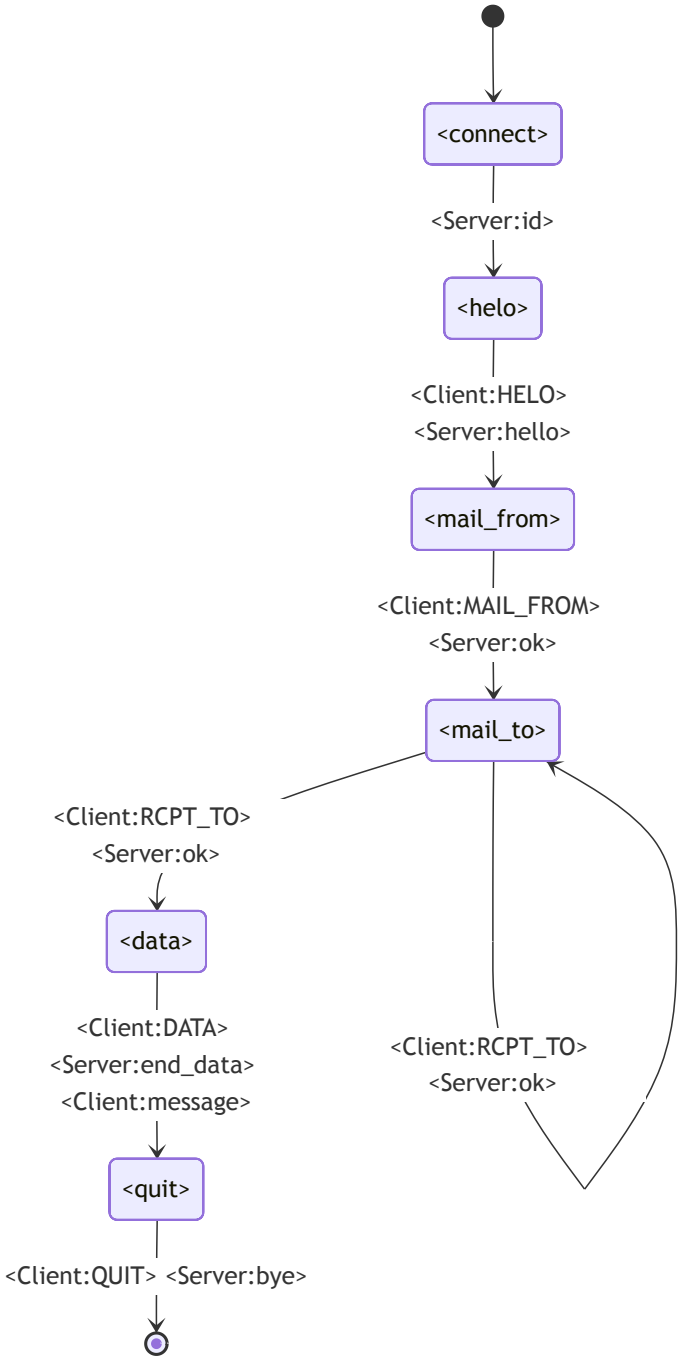
So far, our SMTP server is not great at testing SMTP clients – all it can handle is a single `QUIT` command. Let us extend it a bit with a few more commands, reflecting the interaction in the introduction:

```
<start> ::= <connect>
<connect> ::= <Server:id> <helo>

<id> ::= '220 ' <hostname> ' ESMTP Postfix\r\n'
<hostname> ::= r"[-a-zA-Z0-9.:]+" := "host.example.com"

<helo> ::= <Client:HELO> \
    (<Server:hello> <mail_from> | <Server:error> <end>)
<HELO> ::= 'HELO ' <hostname> '\r\n'

<hello> ::= '250 Hello ' <hostname> ', glad to meet you\r\n'

<error> ::= '5' <digit> <digit> ' ' <error_message> '\r\n'
<error_message> ::= r'[^\r]*' := "Error"

<mail_from> ::= <Client:MAIL_FROM> \
    (<Server:ok> <mail_to> | <Server:error> <end>)

<MAIL_FROM> ::= 'MAIL FROM:<' <email> '>\r\n'
# Actual email addresses are much more varied
<email> ::= r"[-a-zA-Z0-9.]+" '@' <hostname> := "alice@example.com"

<ok> ::= '250 Ok\r\n'

<mail_to> ::= <Client:RCPT_TO> \
    (<Server:ok> <data> | <Server:ok> <mail_to> | <Server:error> <end>)

<RCPT_TO> ::= 'RCPT TO:<' <email> '>\r\n'

<data> ::= <Client:DATA> <Server:end_data> <Client:message> \
    (<Server:ok> <quit> | <Server:error> <end>)

<DATA> ::= 'DATA\r\n'

<end_data> ::= '354 End data with <CR><LF>.<CR><LF>\r\n'

<message> ::= r'[^.\r\n]*\r\n[.]\r\n'

<quit> ::= <Client:QUIT> <Server:bye> <end>

<QUIT> ::= 'QUIT\r\n'

<bye> ::= '221 Bye\r\n'

<end> ::= ''
```

This spec can actually handle the initial interaction (check it!).

27.7 From State Diagrams to Grammars

In the *extended SMTP spec* (page 244), you may note that the interactions follow a particular *order*, implying the *state* the server and client are in. In the "happy" path (assuming no errors), the order of possible states and interactions can be visualized as a *state diagram*:



In this state diagram,

- every *rounded rectangle* stands for a *state*;

- *arrows* denote possible *transitions* between these states;
- *labels* on the transitions denote the interactions that take place when transitioning from one state to another; and
- any *path* through the state diagram indicates a valid sequence of states (and hence a valid sequence of interactions).

State diagrams often contain *loops*. Both the SMTP grammar (and the above state diagram) accepts multiple `<mail_to>` interactions, allowing mails to be sent to multiple destinations.

Note how the set of paths through the state diagram corresponds to the possible sequence of productions in the SMTP specification. Both the state diagram and the grammar effectively specify the same sequence of interactions; the grammar on top also specifies the syntax of individual messages.

Note

In Fandango, you can apply *constraints* (page 45) to all nonterminals, whether they stand for states, interactions, messages, or parts thereof.

27.7.1 Converting State Diagrams to Grammars

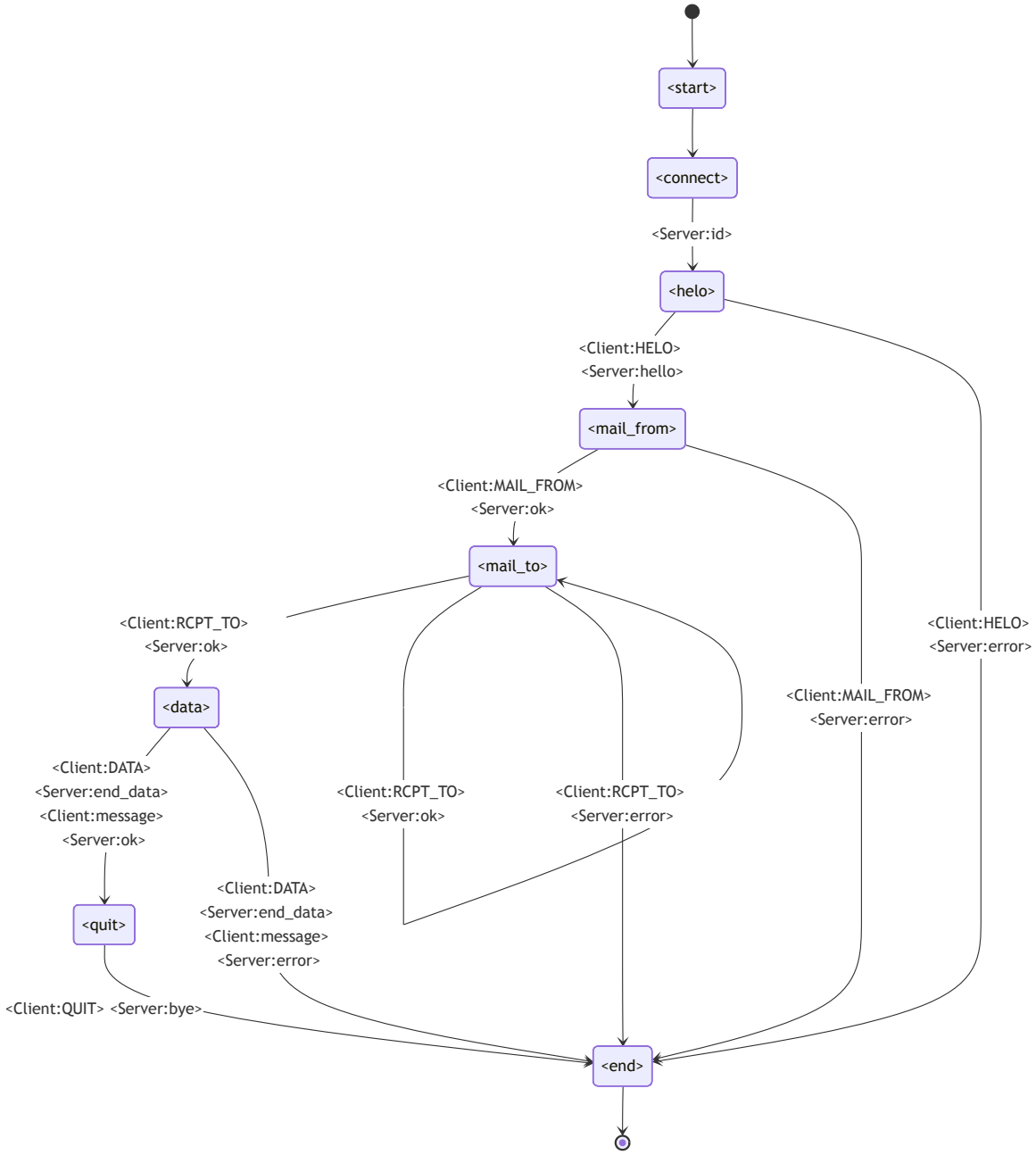
Often, a protocol specification is already given in the form of such a state diagram, also known as a *labeled transition system* (LTS) or as a *finite state automaton* (FSA). You can convert these into Fandango grammars as follows:

1. Every *state* S in the diagram becomes a *nonterminal* S in the grammar.
2. Every *transition* $A \rightarrow B$ in the diagram becomes an *expansion* of A into B, or $A ::= B$.
3. If there are multiple *alternatives* outgoing from A, each of them becomes a separate alternative for the expansion of A.

Apply these rules are to the SMTP state diagram, and check how they correspond to the Fandango SMTP spec.

27.7.2 Modeling Errors

Our SMTP spec above actually accounts for *errors*, always having the server enter an `<error>` state if the client command received cannot be parsed properly. Hence, the state diagram induced by the above grammar actually looks like this:



Having such `<error>` transitions as part of the spec allows Fandango to also cover and trigger these.

27.8 Extracting State Diagrams

You can use Fandango to *automatically extract state diagrams* such as the above. Such a visualization can be helpful for debugging.

- `fandango convert --to=mermaid` produces input for the Mermaid[62] visualizer.
- `fandango convert --to=dot` produces an input in DOT format for the Graphviz[63] visualizer.
- `fandango convert --to=state` produces a generic textual representation.

This assumes the grammar actually embeds a state diagram - the last nonterminal in each expansion is supposed to be a new state, and the nonterminals next to last will become part of the transition.

27.8.1 Visualizing State Diagrams with Mermaid

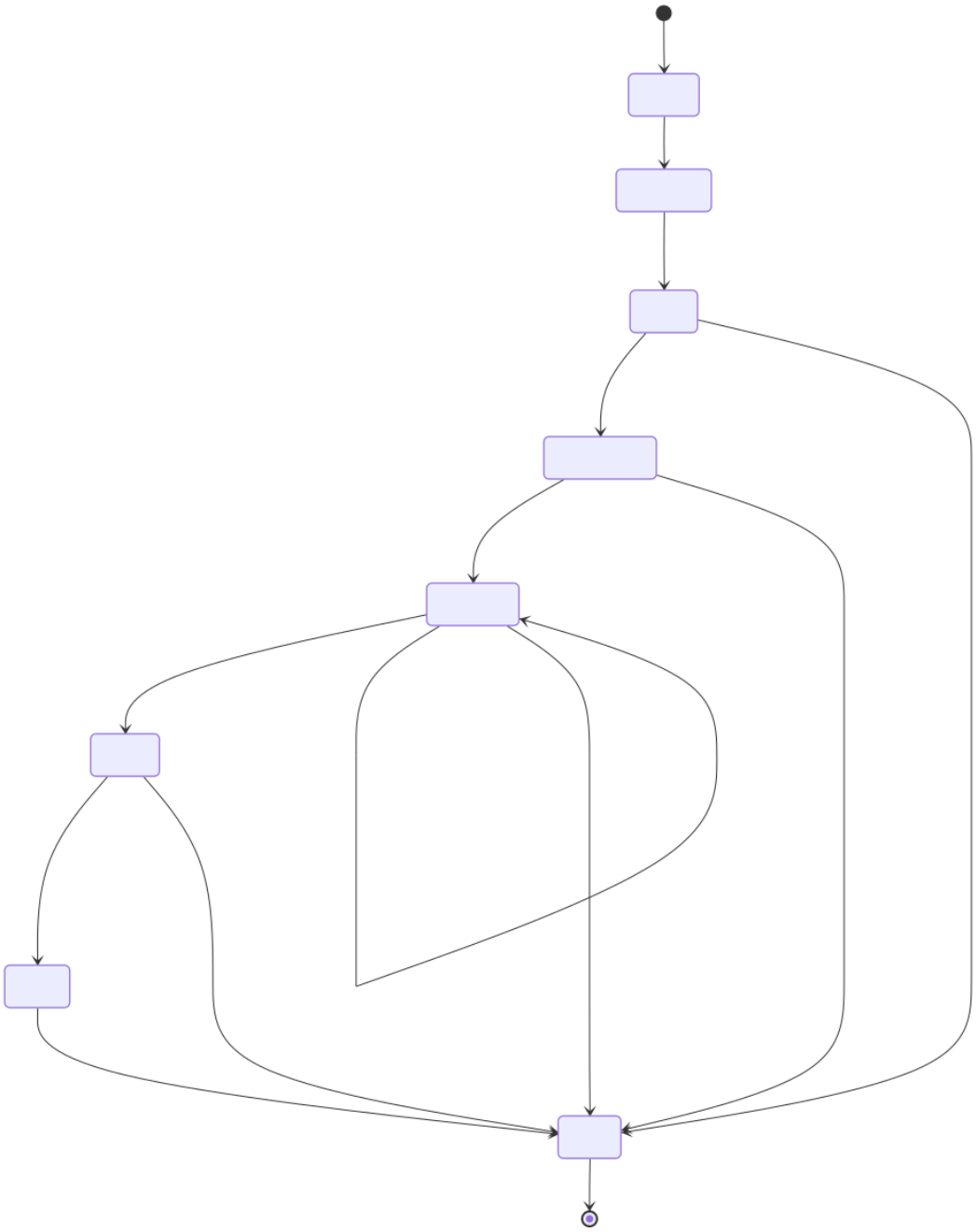
To produce the above state diagram for SMTP as an SVG file smtp-mermaid.svg, use the Mermaid `mmdc` command-line interface[64]:

Use `mmdc --help` to see how to control formats, sizes, and themes.

```
$ fandango convert --to=mermaid smtp-extended.fan | mmdc -i - -o smtp-mermaid.svg
```

The resulting SVG image is the same as embedded above:

[62] https://mermaid.ai/open-source/intro/
[63] https://graphviz.org
[64] https://github.com/mermaid-js/mermaid-cli



27.8.2 Visualizing State Diagrams with Graphviz (DOT)

To produce a similar SVG file smtp-dot.svg using Graphviz, use the `dot` command-line interface[65]:

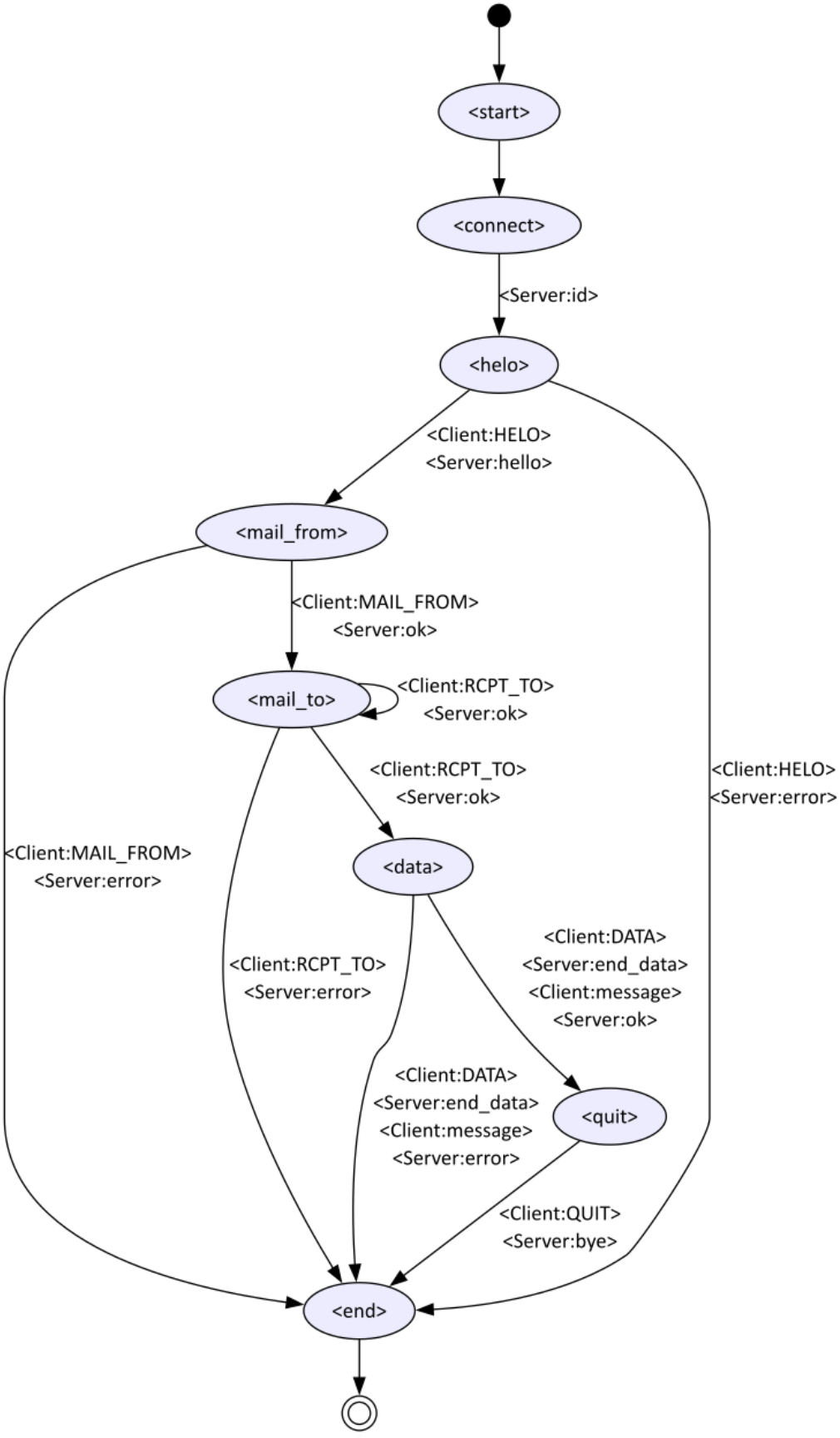
Besides `svg`, Graphviz supports dozens of output formats[66], including `jpg`, `png`, `pdf`, and many more.

[65] https://graphviz.org/doc/info/command.html

```
$ fandango convert --to=dot smtp-extended.fan | dot -T svg -o smtp-dot.svg
```

This is the resulting SVG image:

[66] https://graphviz.org/docs/outputs/


<start>
<connect>
<Server:id>
<helo>
<Client:HELO>
<Server:hello>
<mail_from>
<Client:MAIL_FROM>
<Server:ok>
<mail_to>
<Client:RCPT_TO>
<Server:ok>
<Client:RCPT_TO>
<Server:ok>
<Client:HELO>
<Server:error>
<Client:MAIL_FROM>
<Server:error>
<data>
<Client:RCPT_TO>
<Server:error>
<Client:DATA>
<Server:end_data>
<Client:message>
<Server:ok>
<Client:DATA>
<Server:end_data>
<Client:message>
<Server:error>
<quit>
<Client:QUIT>
<Server:bye>
<end>

Which one is nicer? Pick your favorite.

27.8.3 Extracting State Diagrams as Plain Text

If you want to read or further process the diagram, a simple textual representation is available as well:

```
$ fandango convert --to=state smtp-extended.fan
```

```
# Automatically generated from 'smtp-extended.fan'.
# Format: STATE_1 --> STATE_2: [ACTIONS]; `[*]` is a start/end state
#
[*] --> <start>
<start> --> <connect>
<connect> --> <helo>: <Server:id>
<helo> --> <mail_from>: <Client:HELO> <Server:hello>
<helo> --> <end>: <Client:HELO> <Server:error>
<end> --> [*]
<mail_from> --> <mail_to>: <Client:MAIL_FROM> <Server:ok>
<mail_from> --> <end>: <Client:MAIL_FROM> <Server:error>
<mail_to> --> <data>: <Client:RCPT_TO> <Server:ok>
<mail_to> --> <mail_to>: <Client:RCPT_TO> <Server:ok>
<mail_to> --> <end>: <Client:RCPT_TO> <Server:error>
<data> --> <quit>: <Client:DATA> <Server:end_data> <Client:message> <Server:ok>
<data> --> <end>: <Client:DATA> <Server:end_data> <Client:message> <Server:error>
<quit> --> <end>: <Client:QUIT> <Server:bye>
```

27.9 Simulating Individual Parties

As described in the *chapter on checking outputs* (page 165), we can use the `fuzz` command to actually show generated outputs of individual parties:

```
$ fandango fuzz --party=Client -f smtp-extended.fan
```

```
$ fandango fuzz --party=Server -f smtp-extended.fan
```

That's it for now. GO and thoroughly test your programs!

CHAPTER

TWENTYEIGHT

CUSTOMIZING PARTY COMMUNICATION

In the *previous chapter* (page 233), we have seen how to define and test protocol interactions with Fandango. In this chapter, we describe additional ways to control communication behavior.

Added in version 1.1: These features are available in Fandango 1.1 and later.

28.1 Party Classes

All the communication between Fandango and its parties takes place via dedicated *classes*. In fact, prefixes like `Client` and `Server` in *protocol testing* (page 233) or `In` and `Out` when *checking outputs* (page 165) all stand for predefined *classes* that implement a set of methods to handle the communication:

- `Client` and `Server` are `NetworkParty` classes, detailed in the *`FandangoParty` reference* (page 335).
- `In` and `Out` are `FandangoParty` classes, detailed in the *`FandangoParty` reference* (page 335).

The fact that these are classes allows you to *implement your own communication classes* and define code that should be executed when starting, stopping, or communicating with a party. This is typically done by *subclassing* one of the existing classes.

28.2 Extending Party Functionality

28.2.1 Starting a Party

The `NetworkParty` class has a `start()` method that is invoked whenever the respective party is started. You can extend this method to execute additional code, for instance to create configuration files. To this end, simply subclass the respective party class and override its `start()` method:

```
def create_configuration_file() -> None:
    ... # Your code goes here

class MyClient(Client):
    def start(self) -> None:
        create_configuration_file()
        super().start()  # Invoke the start() method of the superclass
```

This definition goes right into the `.fan` file.

Important

In the grammar, be sure to use your own class name - in this case, `MyClient`. Only then will your definition be used.

```
<start> ::= <MyClient:send> <Server:respond>
<send> ::= ...
```

Tip

You can also *override* existing class names. With this hack, you do not have to change the grammar:

```
class Client(Client):  # extends the original `Client` class
    def start(self) -> None:
        create_configuration_file()
        super().start()  # Invoke the start() method of the superclass
```

See the *FandangoParty reference* (page 338) for details on `start()`.

28.2.2 Stopping a Party

In the same way as extending starting a party, one can also hook into *stopping* a party – for cleanup actions, for instance. The `stop()` method is useful for this:

```
def cleanup_server() -> None:
    ... # Your code goes here

class Server(Server):
    def stop(self):
        cleanup_server()
        super().stop()  # Invoke the stop() method of the superclass
```

Again, this definition goes right into the `.fan` file. See the *FandangoParty reference* (page 338) for details on `stop()`.

28.2.3 Sending Messages

Whenever Fandango wants to send messages to a party, it invokes its `send()` message. As with `start()` and `stop()`, above, you can extend the `send()` method for your own needs. This includes

- altering messages (e.g. encrypt or compress them)
- logging information
- reacting to specific messages

Here is an example in which we compress the `DerivationTree` passed via a `compress()` function.

```
def compress(msg: bytes) -> bytes:
    compressed_message = ... # Your code goes here
    return compressed_message

class Server(Server):
```

(continues on next page)

(continued from previous page)

```
    def send(self, message: DerivationTree | bytes | str, recipient: Optional[str]):
        compressed_message = compress(message.to_bytes())
        super().send(compressed_message, recipient)
```

Important

When called from Fandango, `message` always is of type *`DerivationTree` (page 319)*; this allows you to access its constituents (as in `to_bytes()`, above). However, the `FandangoParty` classes all support sending `DerivationTree`, `str`, and `bytes` types, so you do not have to re-create a `DerivationTree` object when calling `send()`.

Note

For encodings, compression, and other alterations, you can also use a *data converter* (page 141) instead.

See the *`FandangoParty` reference* (page 337) for details on `send()`.

28.2.4 Receiving Messages

Whenever Fandango receives messages from a party, it invokes its `receive()` message. Again, you can extend this for your own needs.

Here is an example in which we decompress the message received via a `decompress()` function.

```
def decompress(msg: bytes) -> bytes:
    decompressed_message = ... # Your code goes here
    return decompressed_message

class Server(Server):
    def receive(self, message: str | bytes, sender: Optional[str]):
        decompressed_message = decompress(message)
        super().receive(decompressed_message, sender)
```

Again, this definition goes right into the `.fan` file. See the *`FandangoParty` reference* (page 338) for details on `receive()`.

28.3 Own Network Parties

If your setting uses more than just `client` or `server` parties, or if the existing `--client` and `--server` options to Fandango are not sufficient, you can define your own network parties by subclassing one of the *`NetworkParty` classes* (page 338).

`Server_1` and `Server_2` could also be two channels to the same server; likewise with the client.

As an example, consider a protocol where two clients `Client_1` and `Client_2` connect to the servers `Server_1` and `Server_2`, respectively. Using the *`NetworkParty` constructor* (page 338), the respective `.fan` file would contain these definitions:

```
class Client_1(NetworkParty):
    def __init__(self):
        super().__init__("localhost:8000",
                         connection_mode=ConnectionMode.CONNECT)

class Server_1(NetworkParty):
    def __init__(self):
        super().__init__("localhost:8000",
                         connection_mode=ConnectionMode.EXTERNAL)

class Client_2(NetworkParty):
    def __init__(self):
        super().__init__("localhost:8000",
                         connection_mode=ConnectionMode.CONNECT)

class Server_2(NetworkParty):
    def __init__(self):
        super().__init__("localhost:8001",
                         connection_mode=ConnectionMode.EXTERNAL)
```

The grammar can then use `Client_1`, `Client_2`, `Server_1`, and `Server_2` as prefixes, and specify a protocol that involves all four parties. As specified, Fandango would operate as client (1 and 2) and fuzz the two servers.

Tip

Use one `.fan` file to specify the protocol, and a second one (which *includes* (page 155) the first) to specify details on where and how the clients and servers are located. This allows alternate settings to also make use (= include) the protocol spec - say, a setting in which Fandango acts as server(s) to fuzz the clients.

28.4 Accessing Network Parties at Runtime

If the party configuration changes at runtime (see the *FTP* (page 259) case study for an example), you may need to access the individual party objects. Use *`PARTY.instance()`* (page 338) for this, where `PARTY` is the class of the respective party.

For instance, to stop the `Server` object, use

```
... Server.instance().stop()
```

If you have given your party an individual name, pass its name as a string argument. In this case, the class is ignored; `FandangoParty` will do fine. The code

```
... FandangoParty.instance("Server").stop()
```

is equivalent to the above.

In the next chapters, we look at how *FTP* (page 259) and *DNS* (page 277) protocol specs make use of these features.

CHAPTER

TWENTYNINE

CASE STUDY: FTP

This Fandango specification allows testing servers and clients for File Transfer Protocol[67] (FTP, RFC 959[68]). It demonstrates

- how a nontrivial protocol with text commands is implemented; and
- how additional channels (`ClientData` and `ServerData`) are used on demand, based on the ports returned by the FTP server

The FTP spec consists of three parts, all available for download:

- `ftp.fan` - the core FTP spec without specific party definitions
- `ftp_client.fan` - for using Fandango as an FTP client (includes `ftp.fan`)
- `ftp_server.fan` - for using Fandango as an FTP server (includes `ftp.fan`)

To test it, run an FTP server on the local host at port 50100, and invoke Fandango as

Omit `-n 1` if you want Fandango to test until protocol coverage is achieved.

```
$ fandango talk -n 1 -f ftp_client.fan
```

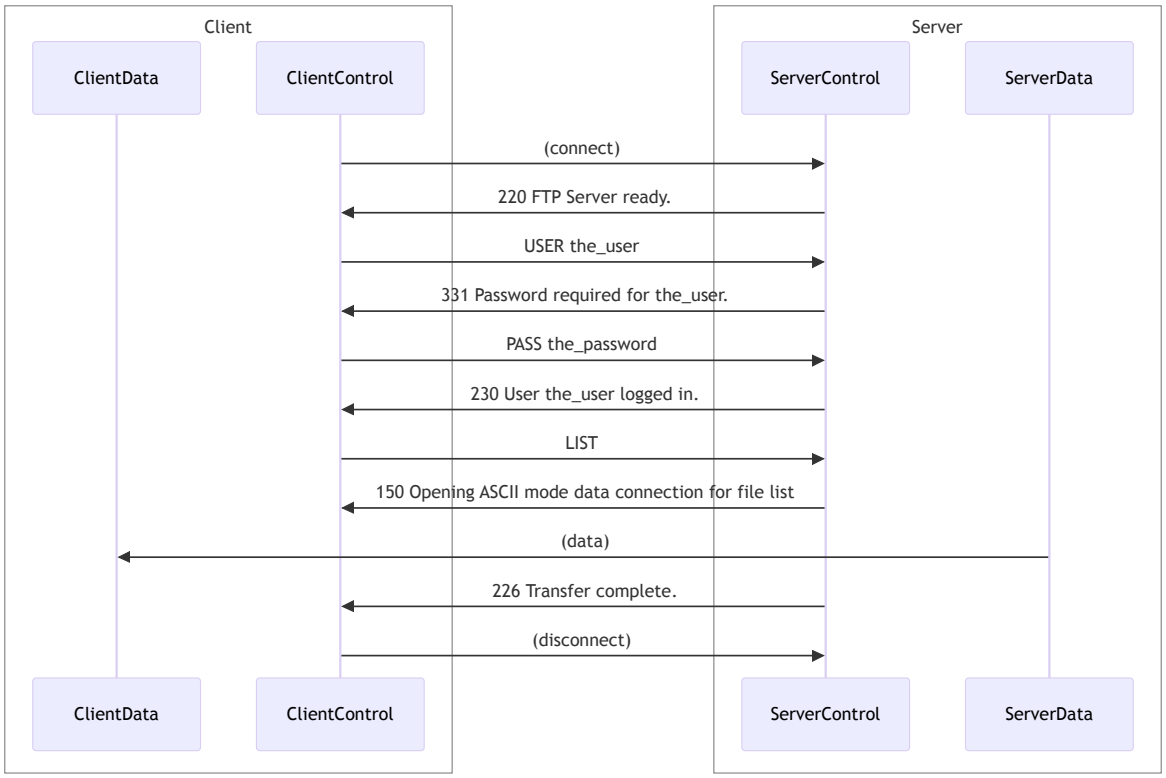
Added in version 1.1: These features are available in Fandango 1.1 and later.

29.1 The FTP Parties

In contrast to other (simpler) protocols, FTP maintains *two* communication channels: one for *control* (issuing commands and getting responses), and one for *data* (for transferring data). In our spec, we name these `ClientControl` and `ClientData` as well as `ServerControl` and `ServerData`.

A very simple interaction involving all four, first logging in, and then sending a `LIST` command to get the contents of the current directory, is illustrated below.

[67] https://en.wikipedia.org/wiki/File_Transfer_Protocol
[68] https://datatracker.ietf.org/doc/html/rfc959


Client
ClientData
ClientControl
Server
ServerControl
ServerData
(connect)
220 FTP Server ready.
USER the_user
331 Password required for the_user.
PASS the_password
230 User the_user logged in.
LIST
150 Opening ASCII mode data connection for file list
(data)
226 Transfer complete.
(disconnect)
ClientData
ClientControl
ServerControl
ServerData

29.1.1 Control Parties

In our setting, we assume that Fandango is acting as *client* to test an FTP server. Fandango connects to a server running on port 25521 on the local host. Whenever it receives a `150` message initiating a data transfer, it starts the `ClientData` party. Here, we use the *`instance()`* (page 258) method to access and reconfigure the individual parties.

```
class ClientControl(NetworkParty):
    def __init__(self):
        super().__init__(
            connection_mode=ConnectionMode.CONNECT,
            uri="tcp://[::1]:25521"
        )
        self.start()

    def receive(self, message: str | bytes, sender: Optional[str]) -> None:
        if message.decode("utf-8").startswith("150"):
            ClientData.instance().start()
```

When the server sends a `226` message, this indicates the end of a data transfer; so we stop the `ServerData` instance to disconnect.

```
class ServerControl(NetworkParty):
    def __init__(self):
        super().__init__(
            connection_mode=ConnectionMode.EXTERNAL,
            uri="tcp://[::1]:25522"
        )
        self.start()

    def receive(self, message: str | bytes, sender: Optional[str]) -> None:
        super().receive(message.decode("utf-8"), sender="ClientControl")

    def send(self, message: DerivationTree, recipient: str):
        super().send(message, recipient)
        if message.to_string().startswith("226"):
            ServerData.instance().stop()
```

29.1.2 Data Parties

In our setting, the FTP data transfer takes place via port 50100 on the local host.

```
class ClientData(NetworkParty):
    def __init__(self):
        super().__init__(
            connection_mode=ConnectionMode.CONNECT,
            uri="tcp://[::1]:50100"
        )

    def receive(self, message: str | bytes, sender: Optional[str]) -> None:
        super().receive(message.decode("utf-8"), sender="ServerData")
```

```
class ServerData(NetworkParty):
    def __init__(self):
        super().__init__(
            connection_mode=ConnectionMode.EXTERNAL,
```

(continues on next page)

(continued from previous page)

```
            uri="tcp://[::1]:50100"
        )

    def receive(self, message: str | bytes, sender: Optional[str]) -> None:
        super().receive(message.decode("utf-8"), sender="ClientData")
```

29.2 Connecting to the FTP Server

The interaction with an FTP server starts with the server `ServerControl` sending a 220 message `<response_server_info>` to the client `ClientControl`, indicating that it is ready for a new user. Afterwards, we are in the state `state_logged_out_1`.

```
<start> ::= <state_setup>
<state_setup> ::= <service_ready> <state_logged_out_1>
<service_ready> ::= <ServerControl:ClientControl:response_server_info>
<response_server_info> ::= r'(220-(?:[\x20-\x7E]*\r\n))*220 (?:[\x20-\x7E]*)\r\n'
```

29.3 Logging In

While we're logged out (in state `<state_logged_out_1>`), we can

- via username and password (`<exchange_login_...>`)

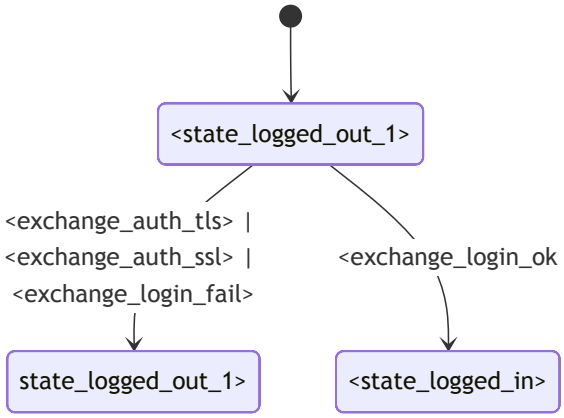
Login with username and password can

- fail (`<exchange_login_fail>`, after which we still stay logged out); or
- succeed (`<exchange_login_ok>`), then, we are logged in (`<state_logged_in>`).

We only support logging in via username and password, so authentication via

- TLS (`<exchange_auth_tls>`) or
- SSL (`<exchange_auth_ssl>`)

are never successful, we remain logged out.

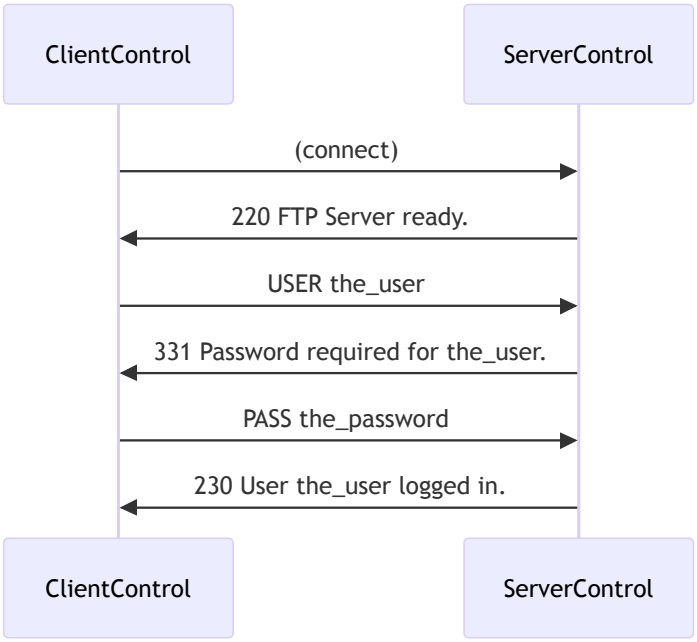


In our spec, this is modeled as exchanges followed by the resulting state.

```
<state_logged_out_1> ::= (
  <exchange_login_ok> <state_logged_in> |
  <exchange_login_fail> <state_logged_out_1>
  <exchange_auth_tls> <state_logged_out_1> |
  <exchange_auth_ssl> <state_logged_out_1> |
  )
```

29.3.1 Logging in with username and password

Our FTP server assumes one user with username `the_user` and a password `the_password`.




```
<exchange_login_ok> ::= (
    <ClientControl:ServerControl:request_login_user_ok>
```

(continues on next page)

(continued from previous page)

```
    <ServerControl:ClientControl:response_login_user>
    <ClientControl:ServerControl:request_login_pass_ok>
    <ServerControl:ClientControl:response_login_pass_ok>
)
```

```
<request_login_user_ok> ::= 'USER the_user\r\n'
<response_login_user> ::= '331 ' <command_tail> '\r\n'
<request_login_pass_ok> ::= 'PASS the_password\r\n'
<response_login_pass_ok> ::= '230 ' <command_tail> '\r\n'
```

29.3.2 Failure to log in

There are two ways logging in can go wrong - an incorrect password (not `the_password`):

```
<wrong_user_password> ::= r'^(?!the_password$)([a-zA-Z0-9_]+)'
```

and an incorrect username (not `the_user`):

```
<wrong_user_name> ::= r'^(?!the_user$)([a-zA-Z0-9_]+)'
```

Let's discuss these two options:

```
<exchange_login_fail> ::= <exchange_wrong_password> | <exchange_wrong_username>
```

First, we can have the client send a correct username, but a wrong password.

```
<exchange_wrong_password> ::= (
  <ClientControl:ServerControl:request_login_user_ok>
  <ServerControl:ClientControl:response_login_user>
  <ClientControl:ServerControl:request_login_pass_fail>
  <ServerControl:ClientControl:response_login_pass_fail>)
```

```
<request_login_user_ok> ::= 'USER the_user\r\n'
<request_login_pass_fail> ::= 'PASS ' <wrong_user_password> '\r\n'
<response_login_pass_fail> ::= '530 ' <command_tail> '\r\n'
<command_tail> ::= r'[\x20-\x7E]+'
```

Second, we can have the client send an incorrect username (with a correct or incorrect password).

```
<exchange_wrong_username> ::= (
   <ClientControl:ServerControl:request_login_user_fail>
   <ServerControl:ClientControl:response_login_user>
     (<ClientControl:ServerControl:request_login_pass_fail> |
      <ClientControl:ServerControl:request_login_pass_ok>)
   <ServerControl:ClientControl:response_login_pass_fail>
  )
```

```
<request_login_user_fail> ::= 'USER ' <wrong_user_name> '\r\n'
```

In both cases, we end up staying logged out (`<state_logged_out_1>`).

29.3.3 (Not) logging in via TLS and SSL

We do not support logging in via TLS and SSL and accept a 500 (syntax error) or 530 (not logged in) message from the server.

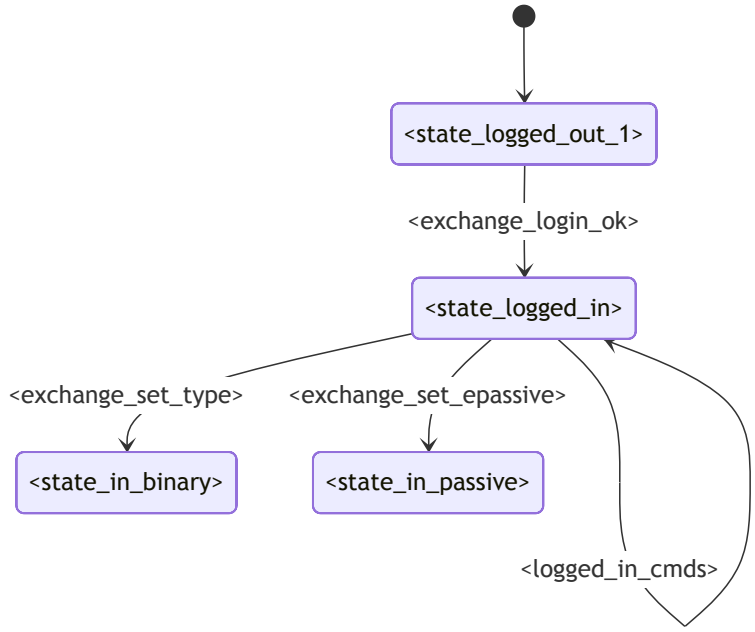
```
<exchange_auth_tls> ::= <ClientControl:ServerControl:request_auth_tls>
 ↪<ServerControl:ClientControl:response_auth_tls>
<request_auth_tls> ::= 'AUTH TLS\r\n'
<response_auth_tls> ::= r'(530|500)' ' ' <command_tail> '\r\n'
```

```
<exchange_auth_ssl> ::= <ClientControl:ServerControl:request_auth_ssl>
 ↪<ServerControl:ClientControl:response_auth_ssl>
<request_auth_ssl> ::= 'AUTH SSL\r\n'
<response_auth_ssl> ::= r'(530|500)' ' ' <command_tail> '\r\n'
```

29.4 First FTP Commands

When the client is logged in, it can send commands to the server. In the `<state_logged_in>` state, we support the commands listed in `<logged_in_cmds>`.

(The *<exchange_set_type>* (page 273) and *<exchange_set_passive>* (page 272) commands change the FTP state; see *States* (page 270) below.)




```
<state_logged_in> ::= <logged_in_cmds> <state_logged_in>
 | <exchange_set_type> <state_in_binary>
```

(continues on next page)

(continued from previous page)

```
 | <exchange_set_epassive> <state_in_passive>
```

```
<logged_in_cmds> ::= (
    <exchange_pwd> |
    <exchange_syst> |
    <exchange_feat> |
    <exchange_set_utf8>)
```

29.4.1 The PWD command

PWD requests the current working directory. The server answers with a (random) path.

```
<exchange_pwd> ::= (
  <ClientControl:ServerControl:request_pwd>
  <ServerControl:ClientControl:response_pwd>
)
<request_pwd> ::= 'PWD\r\n'
<response_pwd> ::= '257 \"' <directory> '\" is the current directory\r\n'
<directory> ::= '/' | ('/' <filesystem_name>)+
<filesystem_name> ::= r'[a-zA-Z0-9_]+'
<client_name> ::= r'[a-zA-Z0-9]+'
```

29.4.2 The SYST command

With the `SYST` command, we can request a `215` reply, followed by the server system name. We use `Linux` as default.

```
<exchange_syst> ::= (
  <ClientControl:ServerControl:request_syst>
  <ServerControl:ClientControl:response_syst>
)
<request_syst> ::= 'SYST\r\n'
<response_syst> ::= '215 ' <syst_name> '\r\n'
<syst_name> ::= r'[\x20-\x7E]+' := 'Linux'
```

29.4.3 The FEAT command

The FEAT command returns a list of features that the server supports. If Fandango generates the feature list, we return a fixed value - in our model, we only support the *EPSV* (page 272) command. When receiving a feature list, we parse it according to the provided regex.

```
<exchange_feat> ::= (
  <ClientControl:ServerControl:request_feat>
  <ServerControl:ClientControl:response_feat>
)
<request_feat> ::= 'FEAT\r\n'
<response_feat> ::= '211-Features:\r\n' <feat_entry>+ '211 End\r\n' := feat_response()
<feat_entry> ::= ' ' r'[\x20-\x7E]+' '\r\n'

def feat_response():
    features = '211-Features:\r\n EPSV\r\n211 End\r\n'
    return features
```

29.4.4 The OPTS UTF8 command

We can send a command to set the character set to UTF-8, expecting a `200` (okay) response.

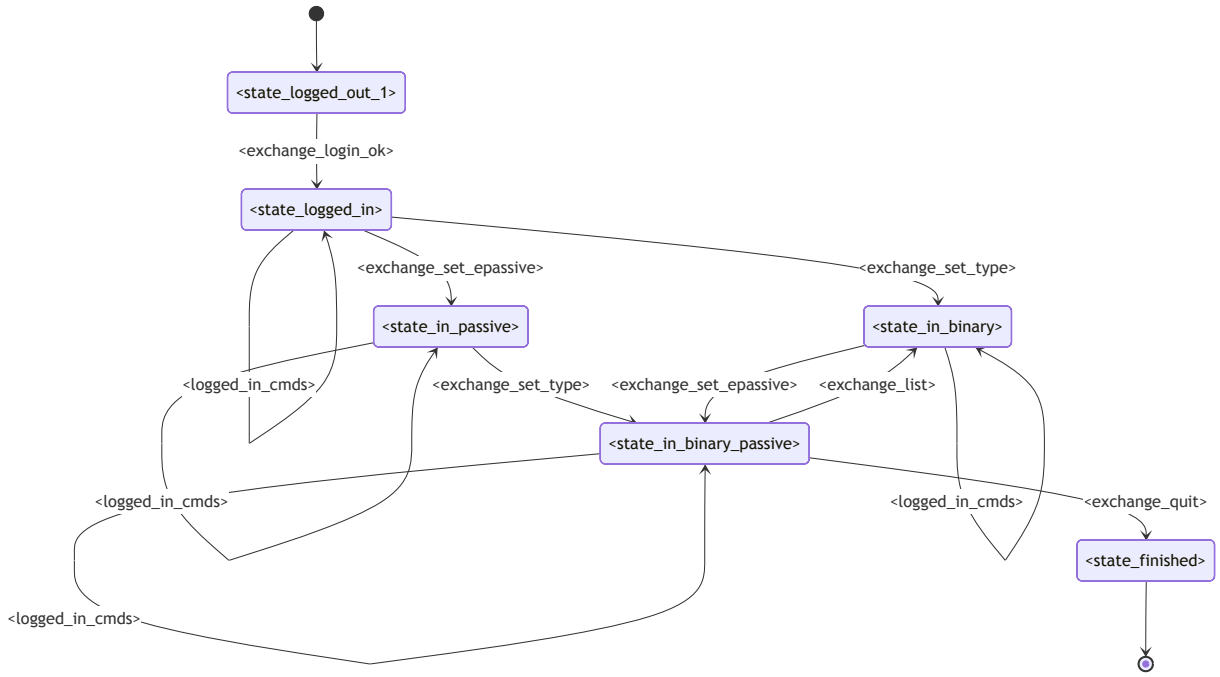
```
<exchange_set_utf8> ::= (
   <ClientControl:ServerControl:request_set_utf8>
   <ServerControl:ClientControl:response_set_utf8>
)
<request_set_utf8> ::= 'OPTS UTF8 ON\r\n'
<response_set_utf8> ::= '200 ' <command_tail> '\r\n'
```

29.5 Changing FTP States

Let us now explore more states. In our model, the FTP server can be in four states:

1. `<state_logged_in>` - the default state
2. `<state_in_binary>` - binary mode
3. `<state_in_passive>` - passive mode
4. `<state_in_binary_passive>` - binary *and* passive mode

Binary and passive modes are activated via *<exchange_set_type>* (page 273) and *<exchange_set_epassive>* (page 272) interactions, as shown below. We want to be in binary *and* passive mode, so we can actually retrieve data using `LIST` (*<exchange_list>* (page 273)) and finally quit (*exchange_quit* (page 274)).




```
<state_in_binary> ::= (
  <logged_in_cmds> <state_in_binary> |
```

(continues on next page)

(continued from previous page)

```
  <exchange_set_epassive> <state_in_binary_passive>
)
<state_in_passive> ::= (
  <logged_in_cmds> <state_in_passive> |
  <exchange_set_type> <state_in_binary_passive>
)
<state_in_binary_passive> ::= (
  <logged_in_cmds> <state_in_binary_passive> |
  <exchange_list> <state_in_binary> |
  <exchange_quit> <state_finished>
)
```

29.5.1 The EPSV command - entering passive mode

By default, FTP uses "active" mode, in which the FTP server accesses a port on the client machine. Since most firewalls block such behavior, "passive" mode is now the typical use.

The `EPSV` command directs the server to open a port for data transmission. With this, we prepare client and server for actual data transfer. The server returns the port number by which it can be accessed; we have to get and process it.

```
<exchange_set_epassive> ::= \
  <ClientControl:ServerControl:request_set_epassive> \
  <ServerControl:ClientControl:response_set_epassive>

<request_set_epassive> ::= 'EPSV\r\n'
<response_set_epassive> ::= '229 Entering Extended Passive Mode (|||' <open_port> '|)\
↪r\n'
```

When producing or parsing `<open_port>`, we call the `open_data_port()` generator function to reconfigure the data parties. This method gets called both when parsing and producing:

- When *producing*, Fandango produces a parameter `<open_port_param>`. `<open_port_param>` consists of `<passive_port>` and another generator that depends on `<open_port>`. This generator does not get executed when generating parameters for the generator from `<open_port>`, such that we do not get caught in an infinite loop between those to generators. Instead, we generate `<passive_port>` directly without invoking the generator.
- When *parsing* `<open_port>`, Fandango derives the argument `<open_port_param>` used in the generator by executing the generator from `<open_port_param>` which depends on `<open_port>`

```
<open_port> ::= <passive_port> := open_data_port(int(<open_port_param>))
<open_port_param> ::= <passive_port> := open_data_port(int(<open_port>))
```

When generating a passive port, we use a generator to randomly generate a port in [50100, 50100]

```
<passive_port> ::= r'[1-9][0-9]{0,4}' := randint(50100, 50100)
```

The function `open_data_port(port)` is a generator. When executed, it returns the value that was given to it and reconfigures the data party definitions to use that port. Again, we use the *`instance()`* (page 258) method to access and reconfigure the individual parties.

As we may use the spec file to run with `fandango fuzz` as server or client, we protect against the fact that the `ClientData` or `ServerData` instances may not have been created.

```
def open_data_port(port):
    try:
        client_data = ClientData.instance()
        server_data = ServerData.instance()
    except KeyError:
        # Party instances not created
        return port

    if client_data.port != port:
        client_data.stop()
        client_data.port = port
    if  server_data.port != port:
        server_data.stop()
        server_data.port = port
    client_data.start()
    server_data.start()
    return port
```

29.5.2 The TYPE command - set binary mode

Using the FTP `TYPE` command, we can set the server into binary mode.

```
<exchange_set_type> ::= (
  <ClientControl:ServerControl:request_set_type>
  <ServerControl:ClientControl:response_set_type>
)
<request_set_type> ::= 'TYPE I\r\n'
<response_set_type> ::= '200 ' <command_tail> '\r\n'
```

29.5.3 The LIST command

Finally, after all these preparations, we can actually retrieve data via FTP. The FTP `LIST` command makes the FTP server send the contents of the current directory.

```
<exchange_list> ::= (
  <ClientControl:ServerControl:request_list>
  <ServerControl:ClientControl:open_list>
  <list_transfer>
)
<request_list> ::= 'LIST\r\n'
<open_list> ::= '150 ' <command_tail> '\r\n'
```

`<list_data>` gets sent using the data channel. Therefore, we use `ServerData` and `ClientData` as sending and receiving parties.

```
<list_transfer> ::= <ServerData:ClientData:list_data>?
 ↪<ServerControl:ClientControl:finalize_list>
<finalize_list> ::= '226 ' <command_tail> '\r\n'
```

The list data itself contains file names, user names, permissions, and dates:

```
<list_data> ::= (<list_data_file>)+
<list_data_file> ::= <permissions> ' '+ <link_count> ' ' <user> ' '+ <group> ' '+
 ↪<file_size> ' ' <date> ' ' <filename> '\r\n'
<filename>    ::= r'[\x20-\x7E]+'
<number>      ::= r'[0-9]+' := str(randint(1, 1000))
<file_size>   ::= <number> := str(randint(0, 9999999))
<link_count>  ::= <number>

<permissions> ::= <file_type> <perm> <perm> <perm>
<file_type>   ::= r'[-dlcb]'
<perm>        ::= r'[r-]' r'[w-]' r'[x-]'
<user>        ::= r'[0-9a-zA-Z_\-]+'
<group>       ::= r'[0-9a-zA-Z_\-]+'
<date>        ::= <month> ' ' <day> ' ' <time>
<month>       ::= r'(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)'
<day>         ::= r'[0-9]{2}' := "{:02d}".format(randint(1, 28))
<time>        ::= <hour> ':' <minute>
<hour>        ::= r'[0-9]{2}' := "{:02d}".format(randint(0, 23))
<minute>      ::= r'[0-9]{2}' := "{:02d}".format(randint(0, 59))
```

This is what a typical generated entry looks like:

```
$ fandango fuzz -f ftp_client.fan --start-symbol '<list_data>' --party ServerData -n 1
```

29.5.4 The QUIT command

After a number of `LIST` commands, it is time to quit. We use the FTP `QUIT` command for this purpose.

```
<exchange_quit> ::= (
  <ClientControl:ServerControl:request_quit>
  <ServerControl:ClientControl:response_quit>
)
<request_quit> ::= 'QUIT\r\n'
<response_quit> ::= '221 ' <command_tail> '\r\n'
```

After that, the FTP server enters `<state_finished>`. There is no other interaction or state following, so we're done.

```
<state_finished> ::= ''
```

29.6 Example Interactions

We can use `fandango fuzz` in conjunction with the `--party` option to simulate the messages produced by a single party.

29.6.1 A Typical Client Interaction

Here is a valid sequence of commands as issued by a client:

```
$ fandango fuzz -f ftp_client.fan --party ClientControl -n 1
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 2.
USER the_user
PASS the_password
EPSV
TYPE I
QUIT
```

29.6.2 A Typical Server Interaction

Here is a valid sequence of responses as issued by a server:

```
$ fandango fuzz -f ftp_client.fan --party ServerControl -n 1
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 1.
```

```
fandango:WARNING: Could not generate a full population of unique individuals.␣
 ↪Population size reduced to 2.
220-,a7o9Xa~O
220-A7?p*+o
220-4x36gg3"qm9Iwd/
220-aV}
220 `Cl\C~wC;zUJBc3FU{
331 YACq%t="
230 Ign{-4$qmmMHn=xm*1J
200 ysplv$Dm
229 Entering Extended Passive Mode (|||50100|)
221 K
```

Now go and try things out for yourself!

CHAPTER

THIRTY

CASE STUDY: DNS

This Fandango specification allows testing clients and name servers of the Internet Domain Name System[69] (DNS, RFC 1035[70]). It demonstrates

- how a nontrivial protocol with bit-level *binary* commands is implemented;
- how to use constraints to *validate* responses; and
- how to embed *compression* and *decompression* functions.

The Fandango DNS protocol spec `dns.fan` is available for download. To test it with Fandango as client querying the public Cloudflare DNS server 1.1.1.1 on the default DNS port 53, invoke Fandango as

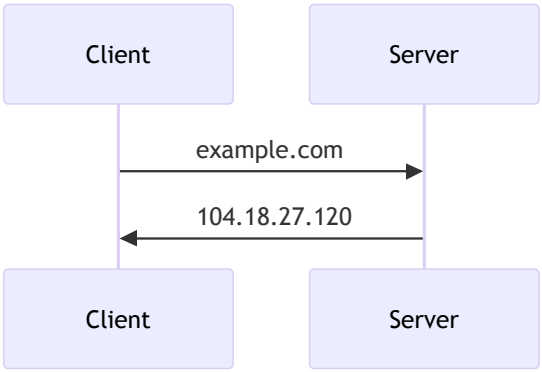
```
$ fandango -v talk -n 1 -f dns.fan --client udp://1.1.1.1:53
```

Added in version 1.1: These features are available in Fandango 1.1 and later.

30.1 A DNS Exchange

At an abstract level, the DNS *protocol* is pretty simple. A *Client* sends a DNS request to the server (via UDP, by default on port 53), and gets a DNS response. For instance, as of January 2026, the host `example.com` had the IP address `104.18.27.120`:

[69] https://en.wikipedia.org/wiki/Domain_Name_System
[70] https://datatracker.ietf.org/doc/html/rfc1035



This is also how the exchange is modeled in `dns.fan`:

```
<start> ::= <exchange>
<exchange> ::= <Client:dns_req> <Server:dns_resp>
```

Alas, the devil is in the details. Let's dig a bit deeper.

30.2 DNS Requests

A DNS request consists of a *header* and a number of *questions*. The actual number of questions is transmitted in a header field `<req_qd_count>`.

```
<dns_req> ::= <header_req> <question>{byte_to_int(<req_qd_count>)}
```

We decode `<req_qd_count>` using a function `byte_to_int()`:

```
# Convert a 2-byte byte array to an integer
def byte_to_int(byte_val):
    return int(unpack('>H', bytes(byte_val))[0])
```

Note

We do not model DNSSEC and other DNS extensions.

30.2.1 Request Headers

The header consists of a number of fields as defined below. Note the mix of bits and bytes, which is common in DNS messages.

```
<header_req> ::= <h_id> 0 <h_opcode_standard> 0 0 <h_rd> 0 0 <bit> 0 <h_rcode_noerror>
 ↪ <req_qd_count> 0{16} 0{16} 0{16}
```

The ID is an identifier of two bytes. When sending the request, we choose a random value:

```
<h_id> ::= <byte><byte>
```

The remaining fields are set to fixed values:

```
<h_opcode_standard> ::= 0 0 0 0
<h_rd> ::= 1 # 0 causes server failure with cname
<req_qd_count> ::= <byte>{2} := pack(">H", 1)
```

30.2.2 Request Questions

Now for the actual DNS questions.

```
<question> ::= <q_name> <q_type> <rr_class>
```

Note

`<q_name>` is where the host name (say, `example.com`) is sent to the DNS server.

Request Names

A request name is terminated by 8 zero bits (i.e., a NUL byte).

```
<q_name> ::= <q_name_written> 0{8}
```

The name is encoded as

- an octet specifying the *length* of the name,
- followed by a sequence of bytes holding the actual name.

```
<q_name_written> ::= (<label_len_octet> <byte>{byte_to_int(b'\x00' + bytes(<label_len_
 ↪octet>))})+ := gen_q_name()
<label_len_octet> ::= <byte>
```

To generate domain names to query, we either

- use the Faker library to obtain a random host name (which likely is not known to the DNS server),
- or we use 'github.com[71]' or 'cispa.de[72]', which should be known to a DNS server.

The individual parts (say, `github` and `com`) are also length-encoded, i.e. there is a leading byte telling the length of the part, followed by the part.

```
from random import randint
from faker import Faker
fake = Faker()

def gen_q_name():
    result = b''
    rand = randint(0, 2)
    if rand == 0:
        domain_name = 'github.com'
    elif rand == 1:
        domain_name = 'cispa.de'
    else:
        domain_name = fake.domain_name()

    domain_parts = domain_name.split('.')
    for part in domain_parts:
        result += len(part).to_bytes(1, 'big')
        result += part.encode('iso8859-1')

    return result
```

Here are some examples of `gen_q_name()`:

```
[gen_q_name() for _ in range(5)]
```

```
[b'\x05cispa\x02de',
 b'\x06github\x03com',
 b'\x06github\x03com',
 b'\rwhitney-perez\x04info',
 b'\x06github\x03com']
```

[71] http://github.com
[72] http://cispa.de

Request Types

We can request the following records from the DNS server:

- `CNAME` - return an "official" host name (or a redirect);
- `A` - return the IP address (the typical use of a DNS server); and
- `NS` - return the name server (a DNS server) for the given domain.

These are encoded using bit sequences as follows:

```
<q_type> ::= <type_id_cname> | <type_id_a> | <type_id_ns>
<type_id_cname> ::= 0{13} 1 0 1
<type_id_a> ::= 0{15} 1
<type_id_ns> ::= 0{14} 1 0
```

Request Class

All our DNS requests refer to the class `IN` (Internet).

```
<rr_class> ::= 0{15} 1 # Class IN (Internet)
```

With that, the definition of our requests are complete.

30.3 DNS Responses

Now for the responses of the DNS server.

```
<dns_resp> ::= <header_resp> <question_section> <answer_an_section> <answer_au_
 ↪section> <answer_opt_section>
```

30.3.1 DNS Response Headers

First, let us have a look at the response header.

```
<header_resp> ::= <h_id> 1 <h_opcode_standard> <bit> 0 <h_rd> <h_ra> 0 <h_aa> 0 (<h_
 ↪rcode_noerror> | <h_rcode_nxdomain>) <resp_qd_count> <resp_an_count> <resp_ns_count>
 ↪ <resp_ar_count>
```

Most of these fields have been defined above already, except for these:

```
<h_aa> ::= <bit>  # 1 if authoritative answer
<h_ra> ::= <bit>  # 1 if recursion is available
```

The most interesting part is the return code, telling success (or non-success) of the query:

```
<h_rcode> ::= <h_rcode_noerror> | <h_rcode_formerr> | <h_rcode_servfail> | <h_rcode_
 ↪nxdomain> | <h_rcode_notimp> | <h_rcode_refused> | <h_rcode_other>
<h_rcode_noerror>  ::= 0 0 0 0  # NOERROR - no error
<h_rcode_formerr>  ::= 0 0 0 1  # FORMERR - format error
<h_rcode_servfail> ::= 0 0 1 0  # SERVFAIL - server failure
<h_rcode_nxdomain> ::= 0 0 1 1  # NXDOMAIN - non existent domain
```

(continues on next page)

(continued from previous page)

```
<h_rcode_notimp>   ::= 0 1 0 0  # NOTIMP - not implemented
<h_rcode_refused>  ::= 0 1 0 1  # REFUSED - query refused
<h_rcode_other> ::= (1 <bit>{3, 3}) | (0 1 1 <bit>)
```

This is followed by four fields denoting the number of fields that follow.

```
<resp_qd_count> ::= <bit>{16} := pack(">H", 1)
<resp_an_count> ::= <bit>{16} := pack(">H", randint(1, 2))
<resp_ns_count> ::= <bit>{16} := pack(">H", randint(0, 2))
<resp_ar_count> ::= <bit>{16} := pack(">H", randint(0, 2))
```

30.3.2 DNS Response Question Section

The DNS server *replicates* the original request in its response. This is necessary as the UDP protocol can drop packets, so the response must clearly specify which request it refers to. This has the same format as the original response.

```
<question_section> ::= <question>{byte_to_int(<header_req>.<req_qd_count>)}
```

Note that we refer to the *original* number of questions from the `<header_req>` request, not the number of questions stated in the response (which is equal anyway). This is for efficiency reasons: The original number of questions is already defined, so we do not have to retrieve it from the server response.

The following constraint ensures that for each request/response pair,

- The recursion desired flag (RD) (`<h_rd>`) match;
- The DNS message identifier (ID) (`<h_id>`) match;
- The question that the response aims to answer is the same as the question in the request (`<question>`); and
- The question count in the response matches the question count in the request (`<req_qd_count>`).

Note that our constraint uses *old-style quantifiers* (page 100) for compatibility with previous Fandango versions.

```
where forall <ex> in <start>.<exchange>:
    <ex>.<dns_resp>.<header_resp>.<h_rd> == <ex>.<dns_req>.<header_req>.<h_rd> and \
    <ex>.<dns_resp>.<header_resp>.<h_id> == <ex>.<dns_req>.<header_req>.<h_id> and \
    <ex>.<dns_resp>.<question_section>.<question> == <ex>.<dns_req>.<question> and \
    bytes(<ex>.<dns_resp>.<header_resp>.<resp_qd_count>) == bytes(<ex>.<dns_req>.
 ↪<header_req>.<req_qd_count>)
```

30.3.3 DNS Response Answer Section

The actual response is contained in these answers:

```
<answer_an_section> ::= <answer_an>{byte_to_int(<resp_an_count>)}
<answer_au_section> ::= <answer_au>{byte_to_int(<resp_ns_count>)}
<answer_opt_section> ::= <answer_opt>{byte_to_int(<resp_ar_count>)}
```

AN Answers

The AN answer field <answer_an> contains the answer to the DNS question. It can return an A record, a CNAME record, or an NS record (see above).

```
<answer_an> ::= <q_name_optional> <answer_an_type>
<q_name_optional> ::= <q_name_written>? 0{8}
<answer_an_type> ::= (<type_a> |<type_cname> | <type_ns>)
```

Type A Answers

This is the answer to an A request querying the IP address of a hostname.

A <type_a> answer holds the A record of the requested domain. The <ip_address> field (finally!) holds the requested IP address, as a sequence of four bytes.

```
<type_a> ::= <type_id_a> <rr_class> <a_ttl> 0{13} 1 0 0 <ip_address>
<type_id_a> ::= 0{15} 1
<a_ttl> ::= 0 <bit>{7} <byte>{3}
<ip_address> ::= <byte>{4}
```

Note

<ip_address> is the place where the IP address (e.g. 104.18.27.120) is sent back.

Note

Our DNS server only supports IP version 4.

Type CNAME Answers

This is the answer to a CNAME request, asking for an "official" host name. <q_name> is the host name returned.

```
<type_cname> ::= <type_id_cname> <rr_class> <a_ttl> <a_rd_length> <q_name>
<a_rd_length> ::= <byte>{2} := pack(">H", randint(0, 0))
```

<type_cname> responses must have the correct length in the <a_rd_length> field (r data length), which corresponds to the following <q_name> field.

Our constraint uses *old-style quantifiers* (page 100) for compatibility with previous Fandango versions.

```
where forall <t> in <type_cname>:
    bytes(<t>.<a_rd_length>) == pack('>H', len(bytes(<t>.<q_name>)))
```

Type NS Answers

Finally, `<type_ns>` is the answer to an `NS` request, returning the name server for the given domain.

```
<type_ns> ::= <type_id_ns> <rr_class> <a_ttl> <a_rd_length> <a_rdata>{int(unpack('>H',
 ↪ bytes(<a_rd_length>))[0])}
<a_rdata> ::= <byte>  # We don't go into further details here.
```

AU Answers

An `AU` answer returns the name server for a domain. We don't go into details here.

```
<answer_au> ::= <q_name_optional> <type_soa>
<type_soa> ::= <type_id_soa> <rr_class> <a_ttl> <a_rd_length> <a_rdata>{int(unpack('>H
 ↪', bytes(<a_rd_length>))[0])}
<type_id_soa> ::= 0{13} 1 1 0
```

OPT Answers

An `OPT` answer returns optional additional details. We don't go into details here.

```
<answer_opt> ::= <q_name_optional> (<type_opt>|<type_a>)
<type_opt> ::= <type_id_opt> <udp_payload_size> <a_ttl> <a_rd_length> <a_rdata>
 ↪{int(unpack('>H', bytes(<a_rd_length>))[0])}
<type_id_opt> ::= 0{10} 1 0 1 0 0 1
<udp_payload_size> ::= <bit>{16}
```

Consistency in Responses

We check if a DNS response answers the corresponding DNS question using the `verify_transitive()` function.

The second part of the `or` clause `bytes(<a>.<answer_an_type>)[0:2]...` also checks this, but only for direct answers without allowing transitive response chains. This allows Fandango optimizations to be used to generate a valid answer more efficiently.

Our constraint uses *old-style quantifiers* (page 100) for compatibility with previous Fandango versions.

```
where forall <ex> in <start>.<exchange>:
    forall <a> in <ex>.<dns_resp>.<answer_an_section>.<answer_an>:
        exists <q> in <ex>.<dns_req>.<question>:
            verify_transitive(<q>, <ex>.<dns_resp>) or \
            (bytes(<a>.<answer_an_type>)[0:2] == bytes(<q>.<q_type>) and
             bytes(<a>.<q_name_optional>) == bytes(<q>.<q_name>))
```

The `verify_transitive()` function gets a question and the response section of a DNS response and verifies if the response does answer the question. This also handles responses that are transitive. If, for example the question is

- for a type `A` record for `example.com` and
- the response contains a `CNAME` record pointing `example.com` to `alias.com` and then an `A` record for `alias.com`,

then `verify_transitive()` will verify that the response correctly answers the original question through the `CNAME` redirection.

```
def verify_transitive(question, response):
    type_byte = bytes(question.find_direct_trees(NonTerminal("<q_type>"))[0])
    allowed_names = [bytes(question.find_direct_trees(NonTerminal("<q_name>"))[0])]

    for ans in response.find_all_trees(NonTerminal("<answer_an>")):
        if (bytes(ans.children[1])[0:2] == pack('>H', 5) and
            bytes(ans.find_direct_trees(NonTerminal("<q_name_optional>"))[0]) in␣
↪allowed_names):
            allowed_names.append(bytes(ans.children[1].children[0].children[4])) #
↪<type_cname>.<q_name>

    for ans in response.find_all_trees(NonTerminal("<answer_an>")):
        if (bytes(ans.children[1])[0:2] == type_byte and
            bytes(ans.find_direct_trees(NonTerminal("<q_name_optional>"))[0]) in␣
↪allowed_names):
            return True

    return False
```

30.4 Compression and Decompression

The DNS protocol maintains a complex system for compressing requests. We implement this by overriding the `NetworkParty` `send()` and `receive()` functions such that

- all sent messages are compressed (`compress_msg()`), and
- all received messages are decompressed (`decompress_msg()`).

We give the new class the same name `NetworkParty` so we can stick to original party names.

```
class NetworkParty(NetworkParty):
    def receive(self, message: str | bytes, sender: Optional[str]) -> None:
        super().receive(decompress_msg(message), sender)

    def send(self, message: str | bytes, recipient: Optional[str]) -> None:
        super().send(compress_msg(message.to_bytes(encoding="utf-8")), recipient)
```

30.4.1 Compression

When receiving a DNS package, some domain names may occur multiple times. To reduce traffic, DNS allows encoding domain names suffixes such that they *refer* to *suffixes* of earlier messages. For instance, if the first answer contains `host.example.com`, then a subsequent `example.com` can be replaced by a pointer into this answer. Such pointers are represented by *offsets* starting from the first answer.

We use the function `msg_suffix()` to identify such compressions. `msg_suffix()` takes an (encoded) domain name in `<q_name>` format and returns a list of tuples (offset, suffix) for a DNS name. Here,

- `suffix` is a possible suffix of `name`
- `offset` is the number of characters into `name` where `suffix` begins

```
def msg_suffix(name):
    suffixes = []
    len_idx = 0
    prefix_len = name[len_idx]
    while prefix_len != 0:
        suffixes.append((len_idx, name[len_idx:]))
        len_idx = len_idx + prefix_len + 1
        prefix_len = name[len_idx]

    return suffixes
```

Here is an example of `msg_suffix()`:

```
name = b'\x07example\x03com\x00'
msg_suffix(name)
```

```
[(0, b'\x07example\x03com\x00'), (8, b'\x03com\x00')]
```

The returned suffix `(8, b'\x03com\x00')` starts 8 bytes into the passed `name`.

```
name[8:]
```

```
b'\x03com\x00'
```

We apply the DNS name compression algorithm to a DNS name starting at `curr_idx` within the uncompressed message. `suffix_dict` is a dictionary mapping a known DNS suffix names to their offsets within the already analyzed compressed part of the message.

```
def compress_name(uncompressed: bytes, curr_idx: int,
                  len_reduction: int, suffix_dict: dict[bytes, int]) -> tuple[bytes,␣
↪int, int]:
    """
    Compress a single name in a DNS message.
    :param: uncompressed - the message before compression
    :param: curr_idx - the current index in `compress_msg()` (see below)
    :param: len_reduction - how many bytes we already have compressed
    :param: suffix_dict - the suffixes encoded so far
    :return: a tuple (new_name, length, len_reduction) - the compressed name, its␣
↪length, the new length reduction
    """
    name_len = 0
    while uncompressed[curr_idx + name_len] != 0:
        name_len += uncompressed[curr_idx + name_len] + 1
    name_len += 1
    b_name = uncompressed[curr_idx:(curr_idx+name_len)]

    if name_len == 1:
        return b_name, name_len, len_reduction

    for n_offset, suffix in msg_suffix(b_name):
        if suffix in suffix_dict:
            cpr_ptr = suffix_dict[suffix]
            bin_ptr = pack('>H', (192 << 8) | cpr_ptr)
            new_name = b_name[:n_offset] + bin_ptr
            len_reduction += len(b_name) - len(new_name)
            return new_name, name_len, len_reduction
```

(continues on next page)

(continued from previous page)

```
        else:
            offset_name_start = curr_idx
            suffix_dict[suffix] = offset_name_start + n_offset - len_reduction
            return b_name, name_len, len_reduction
```

Now for the actual compression. `compress_msg()` compresses a full DNS message applying the DNS name compression algorithm to all names present in the message.

```
def compress_msg(uncompressed: bytes) -> bytes:
    """
    Compress a single DNS message.
    """
    qd_count = byte_to_int(uncompressed[4:6])
    an_count = byte_to_int(uncompressed[6:8])
    ns_count = byte_to_int(uncompressed[8:10])
    ar_count = byte_to_int(uncompressed[10:12])
    compressed = uncompressed[0:12]
    curr_idx = 12

    suffix_dict = dict()
    len_reduction = 0
    for i in range(qd_count):
        name, decompressed_len, len_reduction = compress_name(uncompressed, curr_idx,␣
↪len_reduction, suffix_dict)
        compressed = compressed + name
        curr_idx += decompressed_len
        compressed += uncompressed[curr_idx:curr_idx+4]
        curr_idx += 4

    for i in range(an_count + ns_count + ar_count):
        name, decompressed_len, len_reduction = compress_name(uncompressed, curr_idx,␣
↪len_reduction, suffix_dict)
        compressed = compressed + name
        curr_idx += decompressed_len
        rr_type = uncompressed[curr_idx:curr_idx+2]
        compressed += rr_type
        rr_type = byte_to_int(rr_type)
        curr_idx += 2
        compressed += uncompressed[curr_idx:curr_idx+6]
        curr_idx += 6
        r_data_len = byte_to_int(uncompressed[curr_idx:curr_idx+2])
        curr_idx += 2

        if rr_type == 5: # If CNAME entry
            name, decompressed_len, len_reduction = compress_name(uncompressed, curr_
↪idx, len_reduction, suffix_dict)
            compressed += pack('>H', len(name))
            compressed = compressed + name
            curr_idx += decompressed_len
        else:
            compressed += uncompressed[curr_idx-2:curr_idx]
            compressed += uncompressed[curr_idx:curr_idx+r_data_len]
            curr_idx += r_data_len

    return compressed
```

30.4.2 Decompression

`decompress_name()` decompresses a DNS name starting at `name_idx` within the compressed message.

```
def decompress_name(compressed: bytes, name_idx: int) -> tuple[bytes, int]:
    """
    Decompress the package `compressed` at the current index `name_idx` of a name.
    :param: compressed - the package to be decompressed
    :param: name_idx - the index of a name in `compressed`
    :returns: a pair (decompressed, length) - the decompressed package and its length␣
↪increase
    """
    segment_len = compressed[name_idx]
    compressed_len = 0
    decompressed = b''

    while segment_len != 0:
        # If first two bits are 1
        if (segment_len & 192) == 192:
            name_ptr = (segment_len & 63) << 8
            name_ptr += compressed[name_idx+1]
            decompressed = decompressed + decompress_name(compressed, name_ptr)[0]
            return decompressed, compressed_len + 2
        decompressed = decompressed + bytes([segment_len])
        decompressed = decompressed + compressed[name_idx + 1 : name_idx + 1 +␣
↪segment_len]
        compressed_len = compressed_len + segment_len + 1
        name_idx = name_idx + segment_len + 1
        segment_len = compressed[name_idx]

    decompressed = decompressed + bytes([0])
    return decompressed, compressed_len + 1
```

Conversely, `decompress_msg()` decompresses a full DNS message applying the DNS name decompression algorithm to all names present in the message.

```
def decompress_msg(compressed: bytes) -> bytes:
    """
    Decompress the DNS message `compressed`.
    :param: compressed - the compressed DNS message
    :returns: the decompressed DNS message.
    """
    count_header = compressed[4:12]
    qd_count = byte_to_int(count_header[:2])
    an_count = byte_to_int(count_header[2:4])
    ns_count = byte_to_int(count_header[4:6])
    ar_count = byte_to_int(count_header[6:8])
    decompressed = compressed[0:12]
    curr_idx = 12

    for i in range(qd_count):
        name, compressed_len = decompress_name(compressed, curr_idx)
        decompressed = decompressed + name
        curr_idx += compressed_len
        decompressed += compressed[curr_idx:curr_idx+4]
        curr_idx += 4

    for i in range(an_count + ns_count + ar_count):
```

(continues on next page)

(continued from previous page)

```
        name, compressed_len = decompress_name(compressed, curr_idx)
        decompressed = decompressed + name
        curr_idx += compressed_len
        rr_type = compressed[curr_idx:curr_idx+2]
        decompressed += rr_type
        rr_type = byte_to_int(rr_type)
        curr_idx += 2
        decompressed += compressed[curr_idx:curr_idx+6]
        curr_idx += 6
        r_data_len = byte_to_int(compressed[curr_idx:curr_idx+2])
        curr_idx += 2
        if rr_type == 5: #If CNAME entry
            c_name, compressed_len = decompress_name(compressed, curr_idx)
            decompressed += pack('>H', len(c_name))
            decompressed += c_name
            curr_idx += compressed_len
        else:
            decompressed += compressed[curr_idx-2:curr_idx]
            decompressed += compressed[curr_idx:curr_idx+r_data_len]
            curr_idx += r_data_len

    return decompressed
```

That's all! Now enjoy testing DNS servers :-)

Part VII

Fandango Reference

CHAPTER

THIRTYONE

FANDANGO REFERENCE

This manual contains reference information about Fandango:

CHAPTER

THIRTYTWO

INSTALLING FANDANGO

32.1 Installing Fandango for Normal Usage

Fandango comes as a Python package. To install Fandango, run the following command:

```
$ pip install fandango-fuzzer
```

To test if everything worked well, try

```
$ fandango --help
```

which should give you a list of options:

```
usage: fandango [-h] [--version] [--verbose | --quiet]
                [--parser {python,cpp,legacy,auto}]
                {fuzz,parse,talk,convert,clear-cache,shell,help,copyright,version}␣
 ↪...

The access point to the Fandango framework

options:
  -h, --help            show this help message and exit
  --version             Show version number.
  --verbose, -v         Increase verbosity. Can be given multiple times (-vv).
  --quiet, -q           Decrease verbosity. Can be given multiple times (-qq).
  --parser {python,cpp,legacy,auto}
                        Parser implementation to use (default: 'auto': use C++
                        parser code if available, otherwise Python).

commands:
  {fuzz,parse,talk,convert,clear-cache,shell,help,copyright,version}
                        The command to execute.
    fuzz                Produce outputs from .fan files and test programs.
    parse               Parse input file(s) according to .fan spec.
    talk                Interact with programs, clients, and servers.
    convert             Convert given external spec to .fan format.
    clear-cache         Clear the Fandango parsing cache.
    shell               Run an interactive shell (default).
    help                Show this help and exit.
    copyright           Show copyright.
    version             Show version.

Use `fandango help` to get a list of commands.
```

(continues on next page)

(continued from previous page)

```
Use `fandango help COMMAND` to learn more about COMMAND.
See https://fandango-fuzzer.github.io/ for more information.
```

32.2 Installing Fandango for Development

Caution

This will get you the very latest version of Fandango, which may be unstable. Use at your own risk.

Refer to the *Getting Started with Development Guide* (page 13) for instructions on how to prepare an environment to work on Fandango.

CHAPTER

THIRTYTHREE

FANDANGO COMMAND REFERENCE

33.1 All Commands

Here is a list of all `fandango` commands:

```
usage: fandango [-h] [--version] [--verbose | --quiet]
                [--parser {python,cpp,legacy,auto}]
                {fuzz,parse,talk,convert,clear-cache,shell,help,copyright,version}␣
 ↪...

The access point to the Fandango framework

options:
  -h, --help            show this help message and exit
  --version             Show version number.
  --verbose, -v         Increase verbosity. Can be given multiple times (-vv).
  --quiet, -q           Decrease verbosity. Can be given multiple times (-qq).
  --parser {python,cpp,legacy,auto}
                        Parser implementation to use (default: 'auto': use C++
                        parser code if available, otherwise Python).

commands:
  {fuzz,parse,talk,convert,clear-cache,shell,help,copyright,version}
                        The command to execute.
    fuzz                Produce outputs from .fan files and test programs.
    parse               Parse input file(s) according to .fan spec.
    talk                Interact with prog
```

```
rams, clients, and servers.
    convert             Convert given external spec to .fan format.
    clear-cache         Clear the Fandango parsing cache.
    shell               Run an interactive shell (default).
    help                Show this help and exit.
    copyright           Show copyright.
    version             Show version.

Use `fandango help` to get a list of commands.
Use `fandango help COMMAND` to learn more about COMMAND.
See https://fandango-fuzzer.github.io/ for more information.
```

33.2 Fuzzing

To *produce outputs with `fandango`* (page 31), use `fandango fuzz`:

```
usage: fandango fuzz [-h] [-f FAN_FILE] [-c CONSTRAINT] [-S START_SYMBOL]
                     [--max MAXCONSTRAINT] [--min MINCONSTRAINTS] [-I DIR]
                     [--file-mode {text,binary,auto}] [--no-cache]
                     [--no-stdlib] [-s SEPARATOR] [-d DIRECTORY]
                     [-x FILENAME_EXTENSION]
                     [--format {string,bits,tree,grammar,value,repr,none}]
                     [--validate] [-N MAX_GENERATIONS] [--infinite]
                     [--population-size POPULATION_SIZE]
                     [--elitism-rate ELITISM_RATE]
                     [--crossover-rate CROSSOVER_RATE]
                     [--mutation-rate MUTATION_RATE]
                     [--random-seed RANDOM_SEED]
                     [--destruction-rate DESTRUCTION_RATE]
                     [--max-repetition-rate MAX_REPETITION_RATE]
                     [--max-repetitions MAX_REPETITIONS]
                     [--max-node-rate MAX_NODE_RATE] [--max-nodes MAX_NODES]
                     [-n DESIRED_SOLUTIONS] [--best-effort]
                     [-i INITIAL_POPULATION] [--progress-bar {on,off,auto}]
                     [--warnings-are-errors] [--party PARTY] [-o OUTPUT]
                     [--input-method {stdin,filename,libfuzzer}]
                     [command] ...

options:
  -h, --help            show this help message and exit
  -o, --output OUTPUT   Write output to OUTPUT (default: stdout).

Fandango file settings:
  -f, --fandango-file FAN_FILE
                        Fandango file (.fan, .py) to be processed. Can be
                        given multiple times. Use '-' for stdin.
  -c, --constraint CONSTRAINT
                        Define an additional constraint CONSTRAINT. Can be
                        given multiple times.
  -S, --start-symbol START_SYMBOL
                        The grammar start symbol (default: '<start>').
  --max, --maximize MAXCONSTRAINT
                        Define an additional constraint MAXCONSTRAINT to be
                        maximized. Can be given multiple times.
  --min, --minimize MINCONSTRAINTS
                        Define an additional constraint MINCONSTRAINT to be
                        minimized. Can be given multiple times.
  -I, --include-dir DIR
                        Specify a directory DIR to search for included
                        Fandango files.
  --file-mode {text,binary,auto}
                        Mode in which to open and write files (default is
                        'auto': 'binary' if grammar has bits or bytes, 'text'
                        otherwise).
  --no-cache            Do not cache parsed Fandango files.
  --no-stdlib           Do not include the standard Fandango library.

Output settings:
  -s, --separator SEPARATOR
```

(continues on next page)

(continued from previous page)

```
                        Output SEPARATOR between individual inputs. (default:
                        newline).
  -d, --directory DIRECTORY
                        Create individual output files in DIRECTORY.
  -x, --filename-extension FILENAME_EXTENSION
                        Extension of generated file names (default: '.txt').
  --format {string,bits,tree,grammar,value,repr,none}
                        Produce output(s) as string (default), as a bit
                        string, as a derivation tree, as a grammar, as a
                        Python value, in internal representation, or none.
  --validate            Run internal consistency checks for debugging.

Generation settings:
  -N, --max-generations MAX_GENERATIONS
                        Maximum number of generations to run the algorithm
                        (ignored if --infinite is set).
  --infinite            Run the algorithm indefinitely.
  --population-size POPULATION_SIZE
                        Size of the population.
  --elitism-rate ELITISM_RATE
                        Rate of individuals preserved in the next generation.
  --crossover-rate CROSSOVER_RATE
                        Rate of individuals that will undergo crossover.
  --mutation-rate MUTATION_RATE
                        Rate of individuals that will undergo mutation.
  --random-seed RANDOM_SEED
                        Random seed to use for the algorithm. You probably
                        also want to specify 'PYTHONHASHSEED=<some-value>' to
                        achieve full reproducibility.
  --destruction-rate DESTRUCTION_RATE
                        Rate of individuals that will be randomly destroyed in
                        every generation.
  --max-repetition-rate MAX_REPETITION_RATE
                        Rate at which the number of maximal repetitions should
                        be increased.
  --max-repetitions MAX_REPETITIONS
                        Maximal value the number of repetitions can be
                        increased to.
  --max-node-rate MAX_NODE_RATE
                        Rate at which the maximal number of nodes in a tree is
                        increased.
  --max-nodes MAX_NODES
                        Maximal value, the number of nodes in a tree can be
                        increased to.
  -n, --desired-solutions, --num-outputs DESIRED_SOLUTIONS
                        Number of outputs to produce.
  --best-effort         Produce a 'best effort' population (may not satisfy
                        all constraints).
  -i, --initial-population INITIAL_POPULATION
                        Directory or ZIP archive with initial population.
  --progress-bar {on,off,auto}
                        Whether to show the progress bar. 'auto' (default)
                        shows the progress bar only if stderr is a terminal.

General settings:
  --warnings-are-errors
                        Treat warnings as errors.
```

(continues on next page)

(continued from previous page)

```
Party settings:
  --party PARTY         Only consider the PARTY part of the interaction in the
                        .fan file.

command invocation settings:
  --input-method {stdin,filename,libfuzzer}
                        When invoking COMMAND, choose whether Fandango input
                        will be passed as standard input (`stdin`), as last
                        argument on the command line (`filename`) (default),
                        or to a libFuzzer style harness compiled to a shared
                        .so/.dylib object (`libfuzzer`).
  command               Command to be invoked with a Fandango input.
  args                  The arguments of the command.
```

33.3 Parsing

To *parse inputs with `fandango`* (page 159), use `fandango parse`:

```
usage: fandango parse [-h] [-f FAN_FILE] [-c CONSTRAINT] [-S START_SYMBOL]
                      [--max MAXCONSTRAINT] [--min MINCONSTRAINTS] [-I DIR]
                      [--file-mode {text,binary,auto}] [--no-cache]
                      [--no-stdlib] [-s SEPARATOR] [-d DIRECTORY]
                      [-x FILENAME_EXTENSION]
                      [--format {string,bits,tree,grammar,value,repr,none}]
                      [--validate] [--warnings-are-errors] [--party PARTY]
                      [--prefix] [-o OUTPUT]
                      [files ...]

positional arguments:
  files                 Files to be parsed. Use '-' for stdin.

options:
  -h, --help            show this help message and exit
  --prefix              Parse a prefix only.
  -o, --output OUTPUT   Write output to OUTPUT (default: none). Use '-' for
                        stdout.

Fandango file settings:
  -f, --fandango-file FAN_FILE
                        Fandango file (.fan, .py) to be processed. Can be
                        given multiple times. Use '-' for stdin.
  -c, --constraint CONSTRAINT
                        Define an additional constraint CONSTRAINT. Can be
                        given multiple times.
  -S, --start-symbol START_SYMBOL
                        The grammar start symbol (default: '<start>').
  --max, --maximize MAXCONSTRAINT
                        Define an additional constraint MAXCONSTRAINT to be
                        maximized. Can be given multiple times.
  --min, --minimize MINCONSTRAINTS
                        Define an additional constraint MINCONSTRAINT to be
                        minimized. Can be given multiple times.
  -I, --include-dir DIR
```

(continues on next page)

(continued from previous page)

```
                        Specify a directory DIR to search for included
                        Fandango files.
  --file-mode {text,binary,auto}
                        Mode in which to open and write files (default is
                        'auto': 'binary' if grammar has bits or bytes, 'text'
                        otherwise).
  --no-cache            Do not cache parsed Fandango files.
  --no-stdlib           Do not include the standard Fandango library.

Output settings:
  -s, --separator SEPARATOR
                        Output SEPARATOR between individual inputs. (default:
                        newline).
  -d, --directory DIRECTORY
                        Create individual output files in DIRECTORY.
  -x, --filename-extension FILENAME_EXTENSION
                        Extension of generated file names (default: '.txt').
  --format {string,bits,tree,grammar,value,repr,none}
                        Produce output(s) as string (default), as a bit
                        string, as a derivation tree, as a grammar, as a
                        Python value, in internal representation, or none.
  --validate            Run internal consistency checks for debugging.

General settings:
  --warnings-are-errors
                        Treat warnings as errors.

Party settings:
  --party PARTY         Only consider the PARTY part of the interaction in the
                        .fan file.
```

33.4 Converting

To *convert existing language specs with `fandango`* (page 103), use `fandango convert`:

```
usage: fandango convert [-h] [--party PARTY]
                        [--from {antlr,g4,dtd,010,bt,fan,auto}]
                        [--to {fan,state,mermaid,dot}]
                        [--endianness {little,big}]
                        [--bitfield-order {left-to-right,right-to-left}]
                        [-o OUTPUT]
                        FILENAME [FILENAME ...]

positional arguments:
  FILENAME              External spec file to be converted. Use '-' for stdin.

options:
  -h, --help            show this help message and exit
  --from {antlr,g4,dtd,010,bt,fan,auto}
                        Format of the external spec file: 'antlr'/'g4'
                        (ANTLR), 'dtd' (XML DTD), '010'/'bt' (010 Editor
                        Binary Template), 'fan' (Fandango spec), or 'auto'
                        (default: try to guess from file extension).
  --to {fan,state,mermaid,dot}
```

(continues on next page)

(continued from previous page)

```
                        Format of the output file: 'fan' (Fandango spec;
                        default), 'state' (state diagram), 'mermaid' (Mermaid
                        state diagram), or 'dot' (DOT graph).
  --endianness {little,big}
                        Set endianness for .bt files.
  --bitfield-order {left-to-right,right-to-left}
                        Set bitfield order for .bt files.
  -o, --output OUTPUT   Write output to OUTPUT (default: stdout).

Party settings:
  --party PARTY         Only consider the PARTY part of the interaction in the
                        .fan file.
```

33.5 Interacting

To *have Fandango interact with programs* (page 165) and *other parties* (page 233), use `fandango talk`:

```
usage: fandango talk [-h] [-f FAN_FILE] [-c CONSTRAINT] [-S START_SYMBOL]
                     [--max MAXCONSTRAINT] [--min MINCONSTRAINTS] [-I DIR]
                     [--file-mode {text,binary,auto}] [--no-cache]
                     [--no-stdlib] [-s SEPARATOR] [-d DIRECTORY]
                     [-x FILENAME_EXTENSION]
                     [--format {string,bits,tree,grammar,value,repr,none}]
                     [--validate] [-N MAX_GENERATIONS] [--infinite]
                     [--population-size POPULATION_SIZE]
                     [--elitism-rate ELITISM_RATE]
                     [--crossover-rate CROSSOVER_RATE]
                     [--mutation-rate MUTATION_RATE]
                     [--random-seed RANDOM_SEED]
                     [--destruction-rate DESTRUCTION_RATE]
                     [--max-repetition-rate MAX_REPETITION_RATE]
                     [--max-repetitions MAX_REPETITIONS]
                     [--max-node-rate MAX_NODE_RATE] [--max-nodes MAX_NODES]
                     [-n DESIRED_SOLUTIONS] [--best-effort]
                     [-i INITIAL_POPULATION] [--progress-bar {on,off,auto}]
                     [--warnings-are-errors]
                     [--client [NAME=][PROTOCOL:][HOST:]PORT]
                     [--server [NAME=][PROTOCOL:][HOST:]PORT]
                     [command] ...

positional arguments:
  command               Optional command to be interacted with.
  args                  The arguments of the command.

options:
  -h, --help            show this help message and exit
  --client [NAME=][PROTOCOL:][HOST:]PORT
                        Act as a client NAME (default: 'Client') connecting to
                        PORT on HOST (default: 127.0.0.1; use '[...]' for IPv6
                        addresses) using PROTOCOL ('tcp' (default)/'udp').
  --server [NAME=][PROTOCOL:][HOST:]PORT
                        Act as a server NAME (default: 'Server') running at
                        PORT on HOST (default: 127.0.0.1; use '[...]' for IPv6
                        addresses) using PROTOCOL ('tcp' (default)/'udp').
```

(continues on next page)

(continued from previous page)

```
Fandango file settings:
  -f, --fandango-file FAN_FILE
                        Fandango file (.fan, .py) to be processed. Can be
                        given multiple times. Use '-' for stdin.
  -c, --constraint CONSTRAINT
                        Define an additional constraint CONSTRAINT. Can be
                        given multiple times.
  -S, --start-symbol START_SYMBOL
                        The grammar start symbol (default: '<start>').
  --max, --maximize MAXCONSTRAINT
                        Define an additional constraint MAXCONSTRAINT to be
                        maximized. Can be given multiple times.
  --min, --minimize MINCONSTRAINTS
                        Define an additional constraint MINCONSTRAINT to be
                        minimized. Can be given multiple times.
  -I, --include-dir DIR
                        Specify a directory DIR to search for included
                        Fandango files.
  --file-mode {text,binary,auto}
                        Mode in which to open and write files (default is
                        'auto': 'binary' if grammar has bits or bytes, 'text'
                        otherwise).
  --no-cache            Do not cache parsed Fandango files.
  --no-stdlib           Do not include the standard Fandango library.

Output settings:
  -s, --separator SEPARATOR
                        Output SEPARATOR between individual inputs. (default:
                        newline).
  -d, --directory DIRECTORY
                        Create individual output files in DIRECTORY.
  -x, --filename-extension FILENAME_EXTENSION
                        Extension of generated file names (default: '.txt').
  --format {string,bits,tree,grammar,value,repr,none}
                        Produce output(s) as string (default), as a bit
                        string, as a derivation tree, as a grammar, as a
                        Python value, in internal representation, or none.
  --validate            Run internal consistency checks for debugging.

Generation settings:
  -N, --max-generations MAX_GENERATIONS
                        Maximum number of generations to run the algorithm
                        (ignored if --infinite is set).
  --infinite            Run the algorithm indefinitely.
  --population-size POPULATION_SIZE
                        Size of the population.
  --elitism-rate ELITISM_RATE
                        Rate of individuals preserved in the next generation.
  --crossover-rate CROSSOVER_RATE
                        Rate of individuals that will undergo crossover.
  --mutation-rate MUTATION_RATE
                        Rate of individuals that will undergo mutation.
  --random-seed RANDOM_SEED
                        Random seed to use for the algorithm. You probably
                        also want to specify 'PYTHONHASHSEED=<some-value>' to
                        achieve full reproducibility.
```

(continues on next page)

(continued from previous page)

```
  --destruction-rate DESTRUCTION_RATE
                        Rate of individuals that will be randomly destroyed in
                        every generation.
  --max-repetition-rate MAX_REPETITION_RATE
                        Rate at which the number of maximal repetitions should
                        be increased.
  --max-repetitions MAX_REPETITIONS
                        Maximal value the number of repetitions can be
                        increased to.
  --max-node-rate MAX_NODE_RATE
                        Rate at which the maximal number of nodes in a tree is
                        increased.
  --max-nodes MAX_NODES
                        Maximal value, the number of nodes in a tree can be
                        increased to.
  -n, --desired-solutions, --num-outputs DESIRED_SOLUTIONS
                        Number of outputs to produce.
  --best-effort         Produce a 'best effort' population (may not satisfy
                        all constraints).
  -i, --initial-population INITIAL_POPULATION
                        Directory or ZIP archive with initial population.
  --progress-bar {on,off,auto}
                        Whether to show the progress bar. 'auto' (default)
                        shows the progress bar only if stderr is a terminal.

General settings:
  --warnings-are-errors
                        Treat warnings as errors.
```

33.6 Shell

To *enter commands in `fandango`* (page 61), use `fandango shell` or just `fandango`:

```
usage: fandango shell [-h]

options:
  -h, --help  show this help message and exit
```

33.7 Clearing

To *have Fandango clear its parser cache* (page 342), use `fandango clear-cache`:

```
usage: fandango clear-cache [-h] [-n]

options:
  -h, --help     show this help message and exit
  -n, --dry-run  Just output the action to be performed; do not actually clear
                 the cache.
```

CHAPTER

THIRTYFOUR

FANDANGO LANGUAGE REFERENCE

This chapter specifies the exact syntax (and semantics) of Fandango specifications (`.fan` files).

34.1 General Structure

We specify the Fandango syntax in form of a Fandango specification.

A `.fan` Fandango specification file consists of

- grammar productions (page 306) (`<production>`)
- constraints (page 311) (`<constraint>`)
- Python code (page 311) (`<python_statement>`).

```
<start> ::= <fandango>
<fandango> ::= <statement>*
<statement> ::= <production> | <constraint> | <python_statement> | <newline> |
 ↪<comment>
```

34.2 Whitespace

Besides grammar productions, constraints, and code, a `.fan` file can contain *newlines* (`<newline>`) and *whitespace* (`<_>`).

The *generators* (`:= '\n'` and `:= ' '`) specify useful default values when fuzzing.

```
<newline> ::= ('\r'? '\n' | '\r' | '\f') := '\n'
<_> ::= r'[ \t]*' := ' '
```

34.3 Physical and Logical Lines

As in Python[73], one can join two physical lines into a logical by adding a backslash \ at the end of the first line. For example:

```
<element> := <a_very_long_line> \
    <continued_on_the_next_line>
```

As in Python[74], expressions in parentheses can be split over more than one physical line without using backslashes. For example:

```
<element> := (<an_alternative> |
              <another_alternative>)
```

Grammar rules can be indented; this has no effect on semantics.

34.4 Comments

Comments are as in Python; they are introduced by a # character and continue until the end of the line. By convention, there are two spaces in front of the # and one space after.

```
<comment> ::= <_>{2} '#' <_> r'[^\r\n\f]'* <newline>
```

For example:

```
<element> := <another_element>    # a comment
```

Note

Extending beyond this grammar, the actual implementation allows a comment at any end of a line.

34.5 Grammars

A *grammar* is a set of *productions*. Each production defines a nonterminal (page 307) followed by `::=` and a number of alternatives (page 307).

An optional generator (page 310) can define a Python function to produce a value during fuzzing.

Productions end with a *newline* or a *`;` character*.

```
<production> ::= (
    <nonterminal> <_> '::=' <_> <alternatives>
    <generator>? <comment>? (';' | <newline>))
```

[73] https://docs.python.org/3/reference/lexical_analysis.html#explicit-line-joining
[74] https://docs.python.org/3/reference/lexical_analysis.html#implicit-line-joining

34.6 Nonterminals

A *nonterminal* is a Python identifier enclosed in angle brackets `<...>`. It starts with a letter (regular expression `\w`) or an underscore (_), followed by more letters, digits (regular expression `\d`), or underscores.

```
<nonterminal> ::= '<' <name> '>'
<name> ::= r'(\w|_)(\w|\d|_)*'
```

Like Python, Fandango allows all Unicode letters and digits in identifiers.

> **Note**
>
> For portability, we recommend to use only ASCII letters `a`...`z`, `A`...`Z`, digits `0`...`9`, and underscores _ in identifiers.

34.7 Alternatives

The *alternatives* part of a production rule defines possible *expansions* (page 307) (`<concatenation>`) for a nonterminal, separated by `|`.

```
<alternatives> ::= <concatenation> (<_> '|' <_> <concatenation>)*
```

34.8 Concatenations

A *concatenation* is a sequence of individual *operators* (page 307) (typically symbols such as strings).

```
<concatenation> ::= <operator> (<_> <operator>)*
```

34.9 Symbols and Operators

An operator is a *symbol* (`<symbol>`), followed by an optional *repetition* (page 308) operator.

```
<operator> ::= <symbol> | <kleene> | <plus> | <option> | <repeat>
```

A symbol can be a nonterminal (page 307), a *string* (page 308) or *bytes* (page 309) *literal*, a number (page 310) (for bits), a generator call (page 310), or (parenthesized) *alternatives* (page 307).

```
<symbol> ::= (
      <nonterminal>
    | <string_literal>
    | <bytes_literal>
    | <number>
    | <generator_call>
    | '(' <alternatives> ')'
    )
```

34.10 Repetitions

Any symbol can be followed by a repetition specification. The syntax `{N,M}` stands for a number of repetitions from `N` to `M`. For example, `<a>{3,5}` will match from 3 to 5 `<a>` symbols.

Both `N` and `M` can be omitted:

- Omitting `N` creates a lower bound of zero.
- Omitting `M` creates an infinite upper bound (i.e, any number of repetitions).
- The comma may not be omitted, as this would create confusion with `{N}` (see below).

> **Tip**
>
> In Fandango, the number of repetitions is limited. Use the `--max-repetitions M` flag to change the limit.

Fandango supports a number of abbreviations for repetitions:

- The form `{N}` stands for `{N, N}` (exactly `N` repetitions)
- The form `*` stands for `{0,}` (zero or more repetitions)
- The form `+` stands for `{1,}` (one or more repetitions)
- The form `?` stands for `{0,1}` (an optional element)

```
<kleene> ::= <symbol> '*'
<plus>   ::= <symbol> '+'
<option> ::= <symbol> '?'
<repeat> ::= (
      <symbol> '{' <python_expression> '}'
    | <symbol> '{' <python_expression>? ',' <python_expression>? '}')
```

34.11 String Literals

Fandango supports the full Python syntax for string literals[75]:

- Short strings are enclosed in single (`'...'`) or double (`"..."`) quotes
- Long strings are enclosed in triple quotes (`'''...'''` and `"""..."""`)
- One can use escape sequences[76] (`\n`) to express special characters

Fandango interprets Python *raw strings* (using an `r` prefix, as in `r'foo'`) as *regular expressions*. During parsing, these are matched against the input; during fuzzing, they are instantiated into a matching string.

```
<string_literal> ::= (
       r'[rR]'
     | r'[uU]'
     | r'[fF]'
     | r'[fF][rR]'
     | r'[rR][fF]')? ( <short_string> | <long_string>)
```

[75] https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals
[76] https://docs.python.org/3/reference/lexical_analysis.html#escape-sequences

```
<short_string> ::= (
      "'" (<string_escape_seq> | r"[^\\\r\n\f']")* "'"
    | '"' (<string_escape_seq> | r'[^\\\r\n\f"]')* '"')
```

```
<string_escape_seq> ::= '\\' r'.' | '\\' <newline>
```

```
<long_string> ::= (
      "'''" <long_string_item>* "'''"
    | '"""' <long_string_item>* '"""')

<long_string_item> ::= <long_string_char> | <string_escape_seq>
<long_string_char> ::= r'[^\\]'
```

34.12 Byte Literals

Byte literals (a string prefixed with `b`) are interpreted as in Python; the *rules for strings* (page 308) apply as well.

- If a grammar can produce bytes (or *bits* (page 310)), the associated files will be read and written in `binary` mode, reading and writing *bytes* instead of (encoded) characters.
- If the grammar contains bytes *and* strings, then strings will be written in UTF-8 encoding into the binary file.
- See the *section on binary files* (page 129) for more details.

```
<bytes_literal> ::= (
      r'[bB]'
    | r'[bB][rR]'
    | r'[rR][bB]') (<short_bytes> | <long_bytes>)
```

```
<short_bytes> ::= (
      "'" (<short_bytes_char_no_single_quote> | <bytes_escape_seq>)* "'"
    | '"' ( <short_bytes_char_no_double_quote> | <bytes_escape_seq> )* '"')
```

```
<bytes_escape_seq> ::= '\\' r'[\u0000-\u007F]'
```

```
<short_bytes_char_no_single_quote> ::= (
      r'[\u0000-\u0009]'
    | r'[\u000B-\u000C]'
    | r'[\u000E-\u0026]'
    | r'[\u0028-\u005B]'
    | r'[\u005D-\u007F]')

<short_bytes_char_no_double_quote> ::= (
      r'[\u0000-\u0009]'
    | r'[\u000B-\u000C]'
    | r'[\u000E-\u0021]'
    | r'[\u0023-\u005B]'
    | r'[\u005D-\u007F]')
```

```
<long_bytes> ::= (
      "'''" <long_bytes_item>* "'''"
    | '"""' <long_bytes_item>* '"""')
```

(continues on next page)

(continued from previous page)

```
<long_bytes_item> ::= <long_bytes_char> | <bytes_escape_seq>
<long_bytes_char> ::= r'[\u0000-\u005B]' | r'[\u005D-\u007F]'
```

34.13 Numbers

Grammars can contain *numbers*, which are interpreted as bits (page 137). While the Fandango grammar supports arbitrary numbers, only the number literals `0` and `1` are supported (possibly with repetitions).

```
<number> ::= <integer> | <float_number> | <imag_number>
```

```
<integer> ::= <decimal_integer> | <oct_integer> | <hex_integer> | <bin_integer>
```

```
<decimal_integer> ::= <non_zero_digit> <digit>* | '0'+
<non_zero_digit> ::= r'[1-9]'
<digit> ::= r'[0-9]'
```

```
<oct_integer> ::= '0' r'[oO]' <oct_digit>+
<oct_digit> ::= r'[0-7]'

<hex_integer> ::= '0' r'[xX]' <hex_digit>+
<hex_digit> ::= r'[0-9a-fA-F]'

<bin_integer> ::= '0' r'[bB]' <bin_digit>+
<bin_digit> ::= r'[01]'
```

```
<float_number> ::= <point_float> | <exponent_float>
<point_float> ::= <int_part>? <fraction> | <int_part> '.'
<exponent_float> ::= ( <int_part> | <point_float> ) <exponent>
<int_part> ::= <digit>+
<fraction> ::= '.' <digit>+
<exponent> ::= r'[eE]' r'[+-]'? <digit>+

<imag_number> ::= (<float_number> | <int_part>) r'[jJ]'
```

34.14 Generators

A *generator* is an expression that is evaluated during fuzzing to produce a value, which is then *parsed* into the given nonterminal. See *the section on generators* (page 67) for details. It is added at the end of a production rule, separated by `:=`.

```
<generator> ::= <_> ':=' <_> <python_expression>
```

Future Fandango versions will also support invoking a generator as if it were a symbol.

```
<generator_call> ::= (<name>
    | <generator_call> '.' <name>
    | <generator_call> '[' <python_slices> ']'
    | <generator_call> <python_genexp>
    | <generator_call> '(' <python_arguments>? ')')
```

34.15 Constraints

A *constraint* is a Python expression that is to be satisfied during fuzzing and parsing. It is introduced by the keyword `where`.

Constraints typically contain symbols (page 311) (`<...>`); these are allowed wherever values are allowed. The constraint has to hold for all values of the given symbols.

Symbols in constraints have a `DerivationTree` type; see the *Derivation Tree Reference* (page 319) for details.

```
<constraint> ::= 'where' <_> <python_expression> <comment>? (';' | <newline>)
```

34.16 Selectors

Symbols in constraints can take the following special forms:

- `<A>.<B>`: The constraint has to hold for all elements `<B>` that are a direct child of `<A>`
- `<A>..<B>`: The constraint has to hold for all elements `<B>` that are a direct or indirect child of `<A>`

For details, see *the section on derivation trees* (page 319).

```
<selector> ::= (
      <selection>
    | <selector> '.' <selection>
    | <selector> '..' <selection>)
```

The `[...]` operator allows accessing individual children of a derivation tree:

```
<selection> ::= <base_selection> | <base_selection> '[' <python_slices> ']'
```

These operators can be combined and parenthesized:

```
<base_selection> ::= <nonterminal> | '(' <selector> ')'
```

34.17 Python Code

In a `.fan` file, anything that is neither a *grammar production rule* (page 306) nor a *constraint* (page 311) is interpreted as *Python code*, parsed as `<statement>` in the official Python grammar[77].

Also, in the above spec, any nonterminal in the form `<python_NAME>` (say, `<python_expression>`) refers to `<NAME>` (say, `<expression>`) in the official Python grammar[78].

For more details on Python syntax and semantics, consult the Python language reference[79].

[77] https://docs.python.org/3/reference/grammar.html
[78] https://docs.python.org/3/reference/grammar.html
[79] https://docs.python.org/3/reference/index.html

34.18 The Full Spec

You can access the above spec `fandango.fan` for reference.

`fandango.fan` is sufficient for parsing `.fan` input without Python expressions or code:

```
$ echo '<start> ::= "a" | "b" | "c"' | fandango parse -f fandango.fan -o -
```

```
<start> ::= "a" | "b" | "c"
```

To complete the grammar, `fandango.fan` provides placeholders for included Python elements:

```
<python_statement> ::= 'pass' <newline>
<python_slices> ::= '0:1'
<python_arguments> ::= '1'
<python_expression> ::= '1' | <selector>
<python_genexp> ::= '[for' <_> <name> <_> 'in' <_> <name> ':' <_> <python_expression>
 ↪']'
```

Hence, it is also possible to produce Fandango specs (with set Python code) using `fandango.fan`. Hence, Fandango can be fuzzed with itself:

```
$ fandango fuzz -f fandango.fan -n 1
```

```
pass

pass
```

Note that such generated files satisfy the Fandango syntax, but not its *semantics*. For instance, one would have to add extra constraints such that all used nonterminals are defined.

CHAPTER

THIRTYFIVE

FANDANGO STANDARD LIBRARY

Fandango provides a set of predefined grammar symbols. Each symbol is defined as

- `<_SYMBOL>` (with the actual "official" definition), and
- `<SYMBOL>` (defined as `<_SYMBOL>` by default; can be overridden in individual specifications)

If you'd like to narrow the definition of, say, punctuation characters, you can redefine `<punctuation>` to your liking:

```
<punctuation> ::= '!' | '?' | ',' | '.' | ';' | ':'
```

The original definition of `<_punctuation>`, however, must not be changed, as other definitions may depend on it.

Important

Symbols starting with an underscore must *not* be redefined.

35.1 Characters

A `<char>` represents any Unicode character, including newline.

```
<_char> ::= r'(.|\n)'
<char> ::= <_char>
```

35.2 Printable Characters

These symbols mimic the string constants from the Python `string` module[80]. Use `<digit>`, `<ascii_letter>`, `<whitespace>`, and more to your liking.

```
<_printable> ::= r'[0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!
 ↪\x22#$%&\x27()*+,-./:;<=>?@[\]^_`{|}~ \t\n\r\x0b\x0c]'
<printable> ::= <_printable>

<_whitespace> ::= r'[ \t\n\r\x0b\x0c]'
<whitespace> ::= <_whitespace>

<_digit> ::= r'[0-9]'
```

(continues on next page)

[80] https://docs.python.org/3/library/string.html

(continued from previous page)

```
<digit> ::= <_digit>

<_hexdigit> ::= r'[0-9a-fA-F]'
<hexdigit> ::= <_hexdigit>

<_octdigit> ::= r'[0-7]'
<octdigit> ::= <_octdigit>

<_ascii_letter> ::= r'[a-zA-Z]'
<ascii_letter> ::= <_ascii_letter>

<_ascii_lowercase_letter> ::= r'[a-z]'
<ascii_lowercase_letter> ::= <_ascii_lowercase_letter>

<_ascii_uppercase_letter> ::= r'[A-Z]'
<ascii_uppercase_letter> ::= <_ascii_uppercase_letter>

<_punctuation> ::= r'[!\x22#$%&\x27()*+,-./:;<=>?@[\]^_`{|}~]'
<punctuation> ::= <_punctuation>

<_alphanum> ::= r'[a-zA-Z0-9]'
<alphanum> ::= <_alphanum>

<_alphanum> ::= r'[a-zA-Z0-9_]'
<alphanum> ::= <_alphanum>
```

35.3 Unicode Characters

A `<any_letter>` is any Unicode alphanumeric character, as well as the underscore (_).

An `<any_digit>` is any character in the Unicode character category `[Nd]`. This includes `[0-9]`, and also many other digit characters.

An `<any_whitespace>` is any Unicode whitespace character. This includes `[ \t\n\r\f\v]`, and also many other characters, for example the non-breaking spaces mandated by typography rules in many languages.

The symbols `<any_non_letter>`, `<any_non_digit>`, and `<any_non_whitespace>` match any character that is not in `<any_letter>`, `<any_digit>`, and `<any_whitespace>`, respectively.

```
<_any_letter> ::= r'\w'
<any_letter> ::= <_any_letter>

<_any_digit> ::= r'\d'
<any_digit> ::= <_any_digit>

<_any_whitespace> ::= r'\s'
<any_whitespace> ::= <_any_whitespace>

<_any_non_letter> ::= r'\W'
<any_non_letter> ::= <_any_non_letter>

<_any_non_digit> ::= r'\D'
<any_non_digit> ::= <_any_non_digit>
```

(continues on next page)

(continued from previous page)

```
<_any_non_whitespace> ::= r'\S'
<any_non_whitespace> ::= <_any_non_whitespace>
```

35.4 ASCII Characters

`<ascii_char>` expands into all 7-bit characters in the ASCII range 0x00-0x7F, printable and non-printable.

```
<_ascii_char> ::= rb'[\x00-\x7f]'
<ascii_char> ::= <_ascii_char>
```

35.5 ASCII Control Characters

`<ascii_control>` expands into any of the ASCII control characters 0x00-0x1F and 0x7F. We also provide ASCII codes such as `<ESC>`, `<LF>`, or `<NUL>`.

```
<_NUL> ::= b'\x00'
<NUL> ::= <_NUL>

<_SOH> ::= b'\x01'
<SOH> ::= <_SOH>

<_STX> ::= b'\x02'
<STX> ::= <_STX>

<_ETX> ::= b'\x03'
<ETX> ::= <_ETX>

<_EOT> ::= b'\x04'
<EOT> ::= <_EOT>

<_ENQ> ::= b'\x05'
<ENQ> ::= <_ENQ>

<_ACK> ::= b'\x06'
<ACK> ::= <_ACK>

<_BEL> ::= b'\x07'
<BEL> ::= <_BEL>

<_BS> ::= b'\x08'
<BS> ::= <_BS>

<_HT> ::= b'\x09'
<HT> ::= <_HT>

<_LF> ::= b'\x0a'
<LF> ::= <_LF>

<_VT> ::= b'\x0b'
<VT> ::= <_VT>
```

(continues on next page)

(continued from previous page)

```
<_FF> ::= b'\x0c'
<FF> ::= <_FF>

<_CR> ::= b'\x0d'
<CR> ::= <_CR>

<_SO> ::= b'\x0e'
<SO> ::= <_SO>

<_SI> ::= b'\x0f'
<SI> ::= <_SI>

<_DLE> ::= b'\x10'
<DLE> ::= <_DLE>

<_DC1> ::= b'\x11'
<DC1> ::= <_DC1>

<_DC2> ::= b'\x12'
<DC2> ::= <_DC2>

<_DC3> ::= b'\x13'
<DC3> ::= <_DC3>

<_DC4> ::= b'\x14'
<DC4> ::= <_DC4>

<_NAK> ::= b'\x15'
<NAK> ::= <_NAK>

<_SYN> ::= b'\x16'
<SYN> ::= <_SYN>

<_ETB> ::= b'\x17'
<ETB> ::= <_ETB>

<_CAN> ::= b'\x18'
<CAN> ::= <_CAN>

<_EM> ::= b'\x19'
<EM> ::= <_EM>

<_SUB> ::= b'\x1a'
<SUB> ::= <_SUB>

<_ESC> ::= b'\x1b'
<ESC> ::= <_ESC>

<_FS> ::= b'\x1c'
<FS> ::= <_FS>

<_GS> ::= b'\x1d'
<GS> ::= <_GS>

<_RS> ::= b'\x1e'
<RS> ::= <_RS>
```

(continues on next page)

(continued from previous page)

```
<_US> ::= b'\x1f'
<US> ::= <_US>

<_SP> ::= b'\x20'
<SP> ::= <_SP>

<_DEL> ::= b'\x7f'
<DEL> ::= <_DEL>

<_ascii_control> ::= <NUL> | <SOH> | <STX> | <ETX> | <EOT> | <ENQ> | <ACK> | <BEL>␣
↪| <BS> | <HT> | <LF> | <VT> | <FF> | <CR> | <SO> | <SI> | <DLE> | <DC1> | <DC2>␣
↪| <DC3> | <DC4> | <NAK> | <SYN> | <ETB> | <CAN> | <EM> | <SUB> | <ESC> | <FS> |
↪<GS> | <RS> | <US> | <SP> | <DEL>
<ascii_control> ::= <_ascii_control>
```

35.6 Bits

A `<bit>` represents a bit of 0 or 1. Use `<bit>{N}` to specify a sequence of `N` bits.

```
<_bit> ::= 0 | 1
<bit> ::= <_bit>
```

35.7 Bytes

A `<byte>` is any byte 0x00-0xFF. During parsing and production, it will always be interpreted as a single byte.

```
<_byte> ::= rb'[\x00-\xff]'
<byte> ::= <_byte>
```

35.8 UTF-8 characters

A `<utf8_char>` is a UTF-8 encoding of a character, occupying one (`<utf8_char1>`) to four (`<utf8_char4`) bytes.

```
<_utf8_char1> ::= rb'[\x00-\x7f]'
<utf8_char1> ::= <_utf8_char1>

<_utf8_continuation_byte> ::= rb'[\x80-\xbf]'
<utf8_continuation_byte> ::= <_utf8_continuation_byte>

<_utf8_char2> ::= rb'[\xc2-\xdf]' <utf8_continuation_byte>
<utf8_char2> ::= <_utf8_char2>

<_utf8_char3> ::= rb'[\xe0-\xef]' <utf8_continuation_byte>{2}
<utf8_char3> ::= <_utf8_char3>

<_utf8_char4> ::= rb'[\xf0-\xf5]' <utf8_continuation_byte>{3}
<utf8_char4> ::= <_utf8_char4>
```

(continues on next page)

(continued from previous page)

```
<_utf8_char> ::= <utf8_char1> | <utf8_char2> | <utf8_char3> | <utf8_char4>
<utf8_char> ::= <_utf8_char>
```

35.9 Binary Numbers

For binary representations of numbers, use symbols such as `<int8>` (8 bits) or `<int32>` (32 bits). Note that these symbols only specify the *length*; they do not cover signs, endianness, or byte ordering.

```
<_int8> ::= <byte>
<int8> ::= <_int8>

<_int16> ::= <byte><byte>
<int16> ::= <_int16>

<_int32> ::= <byte>{4}
<int32> ::= <_int32>

<_int64> ::= <byte>{8}
<int64> ::= <_int64>

<_float32> ::= <byte>{4}
<float32> ::= <_float32>

<_float64> ::= <byte>{8}
<float64> ::= <_float64>
```

35.10 Fandango Dancer

The `<fandango-dancer>` symbol is used to test UTF-8 compatibility.

```
<_fandango_dancer> ::= '🕺'
<fandango_dancer> ::= <_fandango_dancer>
```

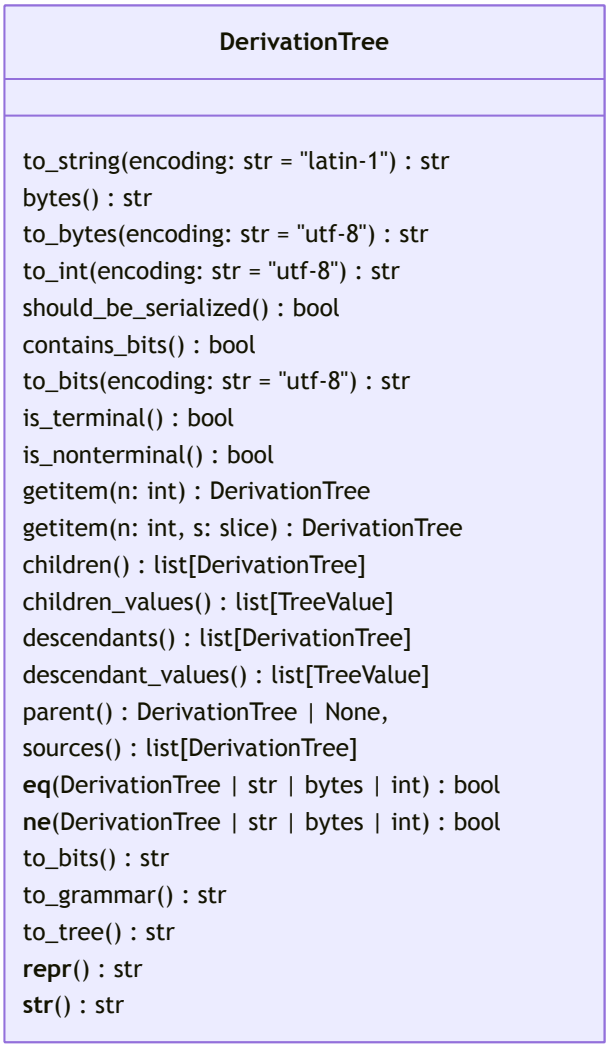
CHAPTER

THIRTYSIX

DERIVATION TREE REFERENCE

Fandango constraints make use of *symbols* (anything enclosed in <...>) to express desired or required properties of generated inputs. During evaluation, any <SYMBOL> returns a *derivation tree* representing the composition of the produced or parsed string.

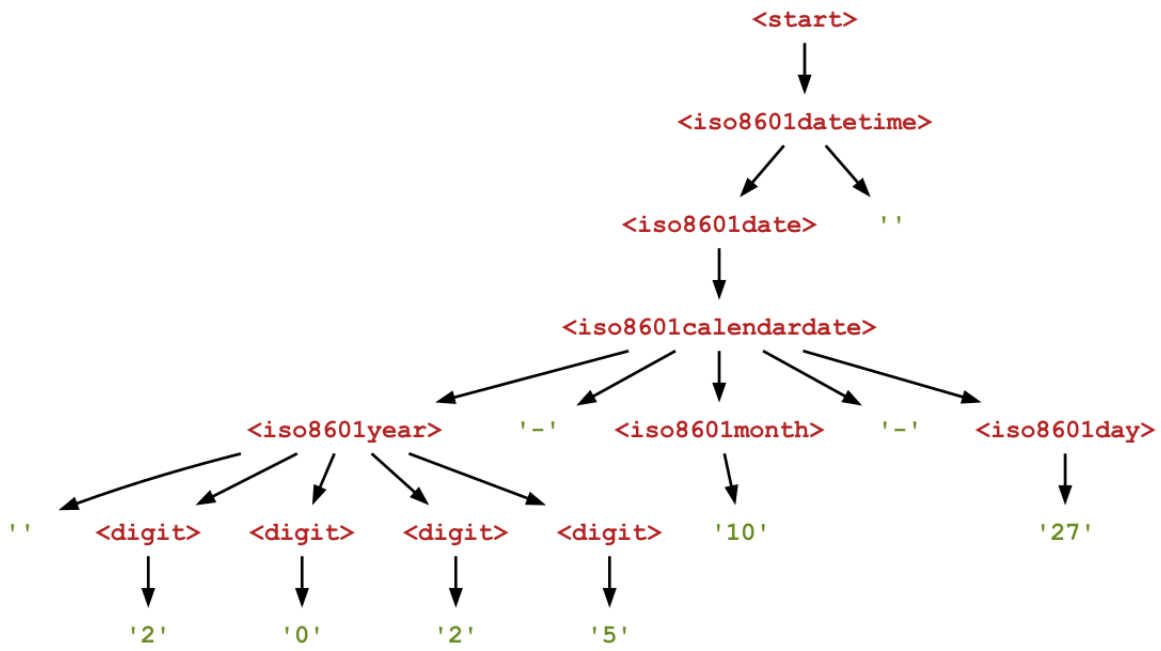
This derivation tree has a type of `DerivationTree`, as do all subtrees. `DerivationTree` objects support *265 functions, methods, and operators* directly; after converting `DerivationTree` objects into standard Python types such as `str`, the entire Python ecosystem is available.


DerivationTree
to_string(encoding: str = "latin-1") : str
bytes() : str
to_bytes(encoding: str = "utf-8") : str
to_int(encoding: str = "utf-8") : str
should_be_serialized() : bool
contains_bits() : bool
to_bits(encoding: str = "utf-8") : str
is_terminal() : bool
is_nonterminal() : bool
getitem(n: int) : DerivationTree
getitem(n: int, s: slice) : DerivationTree
children() : list[DerivationTree]
children_values() : list[TreeValue]
descendants() : list[DerivationTree]
descendant_values() : list[TreeValue]
parent() : DerivationTree | None,
sources() : list[DerivationTree]
eq(DerivationTree | str | bytes | int) : bool
ne(DerivationTree | str | bytes | int) : bool
to_bits() : str
to_grammar() : str
to_tree() : str
repr() : str
str() : str

36.1 Derivation Tree Structure

Let's have a look at the structure of a derivation tree. Consider this derivation tree from the *ISO 8601 date and time grammar* (page 115):

Note that nonterminals can be empty strings, such as the second child of `<iso8601datetime>`.



The elements of the tree are designated as follows:

Node
: Any connected element of a tree.

Child
: An immediate descendant of a node. In the above tree, the `<start>` node has one child, `<iso8601datetime>`.

Descendant
: A child of a node, or a descendant of one of its children. `<iso8601month>` is a descendant of `<iso8601date>`. All nodes (except the root node) are descendants of the root node.

Parent
: The parent of a node N is the node that has N as a child. `<start>` is the parent of `<iso8601datetime>`.

Root
: The node without parent; typically `<start>`.

Terminal Symbol
: A node with at least one child.

Nonterminal Symbol
: A node without children.

Concatenating all terminal symbols (using `str(<SYMBOL>)`) in a derivation tree yields the string represented. For the above tree, this would be `2025-10-27`.

36.2 Evaluating Derivation Trees

To write constraints, you may want to serialize derivation trees into standard Python types such as `str`ings, `bytes`, or `int`s. To do so, simply call `str(<SYMBOL>)`, `bytes(<SYMBOL>)`, or `int(<SYMBOL>)` respectively.

Internally, the serialization is done in `str` objects as long as the subtree only contains string values, and in `bytes` otherwise. During concatenation, strings are converted to bytes in with `utf-8` encoding, for conversion from bytes to strings, `latin-1` is used to prevent breaking non-utf-8 strings. If you would like to convert them using utf-8, you can call `<SYMBOL>.to_string("utf-8")`.

Concatenation of non-bit types with bit types always triggers Fandango to convert the remaining trailing bits into `bytes` (with the first bit being the most significant). If the number of trailing bits isn't a multiple of 8, Fandango will throw an error.

`int(<SYMBOL>)` on a subtree consisting only of strings will attempt to parse the string as a number. If the tree consists of bits only, it will be converted to a number with the first bit being the most significant. Bytes are converted to strings and then treated accordingly.

Deprecated since version 1.0: Previous versions of Fandango automatically converted certain values — this no longer works to prevent accidental mistakes while writing constraints.

> **Tip**
>
> It is generally unwise to rely on the type of the internal representation of linearized `DerivationTree` values as these may change with future expansions of Fandango. Use functions directly implemented on them (`<SYMBOL>.startswith("Hello")`), or explicitly convert them to a base type in Python instead (`str(<SYMBOL>)`).

36.3 General `DerivationTree` Functions

These functions are available for all `DerivationTree` objects, regardless of the type they evaluate into.

36.3.1 Converters

Since any `<SYMBOL>` has the type `DerivationTree`, one must convert it first into a standard Python type before passing it as argument to a standard Python function.

`str(<SYMBOL>) -> str`
: Convert `<SYMBOL>` into a Unicode string. Byte strings in `<SYMBOL>` are converted using `latin-1` encoding.

`<SYMBOL>.to_string(encoding: str = "latin-1") -> str`
: More flexible alternative to `str(<SYMBOL>)`

`bytes(<SYMBOL>) -> bytes`
: Convert `<SYMBOL>` into a byte string. Unicode strings in `<SYMBOL>` are converted using `utf-8` encoding.

`<SYMBOL>.to_bytes(encoding: str = "utf-8") -> bytes`
: More flexible alternative to `bytes(<SYMBOL>)`

int(<SYMBOL>) -> int
Convert <SYMBOL> into an integer, like the Python int() function. <SYMBOL> must be exclusively bits, or a Unicode string or byte string representing an integer literal.

<SYMBOL>.to_int(encoding: str = "utf-8") -> int
More flexible alternative to int(<SYMBOL>)

<SYMBOL>.should_be_serialized_to_bytes() -> bool
Returns True if the sub-tree contains bits or bytes and thus is natively serialized to bytes

<SYMBOL>.contains_bits() -> bool
Returns True if the sub-tree contains bits

<SYMBOL>.contains_bytes() -> bool
Returns True if the sub-tree contains bytes

<SYMBOL>.to_bits(encoding: str = "utf-8") -> int
Provides a bitwise representation of the input, in a string of 0s and 1s

<SYMBOL>.is_terminal() -> bool
True if <SYMBOL> is a terminal node.

<SYMBOL>.is_nonterminal() -> bool
True if <SYMBOL> is a nonterminal node.

36.3.2 Accessing Children

len(<SYMBOL>) -> int
Return the number of children of <SYMBOL>.

> **Important**
>
> To access the length of the *string* represented by <SYMBOL>, use len(str(<SYMBOL>)).

<SYMBOL>[n] -> DerivationTree
Access the nth child of <SYMBOL>, as a DerivationTree. <SYMBOL>[0] is the first child; <SYMBOL>[-1] is the last child.

> **Important**
>
> To access the nth *character* of <SYMBOL>, use str(<SYMBOL>)[n].

<SYMBOL>[start:stop] -> DerivationTree
Return a new DerivationTree which has the children <SYMBOL>[start] to <SYMBOL>[stop-1] as children. If start is omitted, children start from the beginning; if stop is omitted, children go up to the end, including the last one.

<SYMBOL>.children() -> list[DerivationTree]
Return a list containing all children of <SYMBOL>.

<SYMBOL>.children_values() -> list[TreeValue]
Return a list containing the values of all children of <SYMBOL>.

> **Note**
>
> `TreeValue` objects can be converted into standard Python objects with `str()`, `bytes()`, or `int()`. Alternatively, use the same base functionality from `str`, `bytes`, and `int` on them directly that is *implemented on entire `DerivationTrees`* (page 325).

`<SYMBOL_1> in <SYMBOL_2>`
Return True if `<SYMBOL_1> == CHILD` for any of the children of `<SYMBOL_2>`.

`VALUE in <SYMBOL>`
Return True if `VALUE == CHILD.value()` for any of the children of `<SYMBOL>`.

36.3.3 Accessing Descendants

`<SYMBOL>.descendants() -> list[DerivationTree]`
Return a list containing all descendants of `<SYMBOL>`; that is, all children and their transitive children.

`<SYMBOL>.descendant_values() -> list[TreeValue]`
Return a list containing the values of all descendants of `<SYMBOL>`; that is, the values of all children and their transitive children.

> **Note**
>
> `TreeValue` objects can be converted into standard Python objects with `str()`, `bytes()`, or `int()`. Alternatively, use the same base functionality from `str`, `bytes`, and `int` on them directly that is *implemented on entire `DerivationTrees`* (page 325).

36.3.4 Accessing Parents

`<SYMBOL>.parent() -> Optional[DerivationTree]`
Return the parent of the current node, or `None` for the root node.

36.3.5 Accessing Sources

`<SYMBOL>.sources() -> list[DerivationTree]`
Return a list containing all sources of `<SYMBOL>`. Sources are symbols used in generator expressions out of which the value of `<SYMBOL>` was created; see *the section on data conversions* (page 141) for details.

36.3.6 Comparisons

`<SYMBOL_1> == <SYMBOL_2>`
Returns True if both trees have the same structure and all nodes have the same values.

`<SYMBOL> == VALUE`
Returns True if `<SYMBOL>.value() == VALUE` — requires the types to match

`<SYMBOL_1> != <SYMBOL_2>`
Returns True if both trees have a different structure or any nodes have different values.

<SYMBOL> != VALUE
Inverse of `<SYMBOL> == VALUE`.

To compare values with <, >, <=, >=, etc., explicitly convert them to the type you would like to use for comparison first.

36.3.7 Debugging

See the *section on output formats* (page 162) for details on these representations.

<SYMBOL>.to_bits() -> str
Return a bit representation (`0` and `1` characters) of `<SYMBOL>`.

<SYMBOL>.to_grammar() -> str
Return a grammar-like representation of `<SYMBOL>`.

<SYMBOL>.to_tree() -> str
Return a tree representation of `<SYMBOL>`, using `Tree(...)` constructors.

repr(<SYMBOL>) -> str
Return the internal representation of `<SYMBOL>`, as a `DerivationTree` constructor that can be evaluated as a Python expression.

36.4 Type-Specific Functions

The bulk of available functions comes from the Python standard library. The vast majority of methods implemented on `str`, `bytes`, and `int` are also implemented on `DerivationTrees`:

- If a method is only implemented on one of the three base types, the `DerivationTree` internally is first transformed into that type (using `str(<SYMBOL>)` etc.).
- If a method is implemented on multiple base types, but they always have different signatures (such as `.startswith`, which takes a `str` if called on a `str`, and `bytes` if called on `bytes`), the `DerivationTree` is transformed into the appropriate base type.
- If a method is implemented on multiple base types, and there is no way to distinguish based on the signatures, the method is invoked on the underlying type. This is not recommended as these methods rely on knowledge of the type of the internal representation (which may change in the future). Simply convert the tree to the desired base type first (`str(<SYMBOL>).upper()`).

CHAPTER

THIRTYSEVEN

PYTHON API REFERENCE

Fandango provides a simple API for Python programs to

- load *a `.fan` specification with `Fandango()`* (page 327);
- produce outputs *from the spec with `fuzz()`* (page 330); and
- parse inputs *using the spec with `parse()`* (page 330).

Note

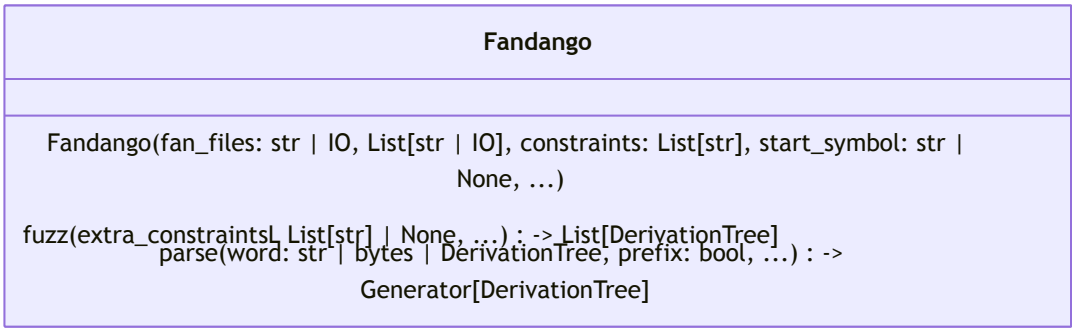
This API will be extended over time.

37.1 Installing the API

When you *install Fandango* (page 295), the Python Fandango API is installed as well.

37.2 The `Fandango` class

The Fandango API comes as a class, named `Fandango`:



To use it, write

```
from fandango import Fandango
```

The `Fandango` constructor allows reading in a `.fan` specification, either from an (open) file, a string, or a list of strings or files.

```
class Fandango(fan_files: str | IO | List[str | IO],
    constraints: List[str] = None, *,
    start_symbol: Optional[str] = None,
    use_cache: bool = True,
    use_stdlib: bool = True,
    logging_level: Optional[int] = None)
```

Create a `Fandango` object.

- `fan_files` is the Fandango specification to load. This is either
  - a *string* holding a `.fan` specification; or
  - an (open) `.fan` *file*; or
  - a *list* of strings or files.
- `constraints`, if given, is a list of additional constraints (as strings).
- `start_symbol` is the start symbol to use (default: `<start>`).
- `use_cache` can be set to `False` to avoid loading the input from cache.
- `use_stdlib` can be set to `False` to avoid loading the *standard library* (page 313).
- `includes`: A list of directories to search for include files before the *Fandango spec locations* (page 341).
- `logging_level` controls the logging output. It can be set to any of the values in the Python logging module[81], such as `logging.DEBUG` or `logging.INFO`. Default is `logging.WARNING`.

Warning

Be aware that `.fan` files can contain Python code that is *executed when loaded*. This code can execute arbitrary commands.

Warning

Code in the `.fan` spec cannot access identifiers from the API-calling code or vice versa. However, as both are executed in the same Python interpreter, there is a risk that loaded `.fan` code may bypass these restrictions and gain access to the API-calling code.

Caution

Only load `.fan` files you trust.

`Fandango()` can raise a number of exceptions, including

- `FandangoSyntaxError` if the `.fan` input has syntax errors. The exception attributes `line`, `column`, and `msg` hold the line, column, and error message.

[81] https://docs.python.org/3/library/logging.html

- `FandangoValueError` if the `.fan` input has consistency errors.

The exception class `FandangoError` is the superclass of these exceptions.

37.3 The `fuzz()` method

On a `Fandango` object, use the `fuzz()` method to produce outputs from the loaded specification.

```
fuzz(extra_constraints: Optional[List[str]] = None, **settings)
    -> List[DerivationTree])
```

Create outputs from the specification, as a list of *derivation trees* (page 319).

In the future, the set of available `settings` may change dependent on the chosen algorithm.

- `extra_constraints`: if given, use this list of strings as additional constraints
- `settings`: pass extra values to control the fuzzer algorithm. These include
  - `population_size: int`: set the population size (default: 100).
  - `desired_solutions: int`: set the number of desired solutions.
  - `initial_population: List[Union[DerivationTree, str]]`: set the initial population.
  - `max_generations: int`: set the maximum number of generations (default: 500).
  - `warnings_are_errors: bool` can be set to True to raise an exception on warnings.
  - `best_effort: bool` can be set to True to return the population even if it does not satisfy the constraints.

The `fuzz()` method returns a list of *`DerivationTree` objects* (page 319). These are typically converted into Python data types (typically using `str()` or `bytes()`) to be used in standard Python functions.

`fuzz()` can raise a number of exceptions, including

- `FandangoFailedError` if the algorithm failed *and* `warnings_are_errors` is set.
- `FandangoValueError` if algorithm settings are invalid.
- `FandangoParseError` if a generator value could not be parsed.

The exception class `FandangoError` is the superclass of these exceptions.

37.4 The `parse()` method

On a `Fandango` object, use the `parse()` method to parse an input using the loaded specification.

```
parse(word: str | bytes | DerivationTree, *, prefix: bool = False, **settings)
    -> Generator[DerivationTree, None, None]
```

Parse a word; return a generator for *derivation trees* (page 319).

- `word`: the word to be parsed. This can be a string, a byte string, or a derivation tree.
- `prefix`: if True, allow incomplete inputs that form a prefix of the inputs accepted by the grammar.

- `settings`: additional settings for the parser.
  - There are no additional user-facing settings for the parser at this point.

The return value is a *generator* of derivation trees - if the `.fan` grammar is ambiguous, a single word may be parsed into different trees. To iterate over all trees parsed, use a construct such as

```
for tree in fan.parse(word):
    ...  # do something with the tree
```

`parse()` can raise exceptions, notably

- `FandangoParseError` if parsing fails. The attribute `position` of the exception indicates the position in `word` at which the syntax error was detected.[82]

> **Note**
>
> At this point, the `parse()` method does not check whether constraints are satisfied.

37.5 API Usage Examples

37.5.1 Fuzzing from a `.fan` Spec

Let us read and produce inputs from `persons-faker.fan` discussed in *the section on generators and fakers* (page 67):

```
from fandango import Fandango

# Read in a .fan spec from a file
with open('persons-faker.fan') as persons:
    fan = Fandango(persons)

for tree in fan.fuzz(desired_solutions=10):
    print(str(tree))
```

```
Ann Armstrong,7473831876
Jennifer Warren,16696343702708256795
David Miller,73397
Erin Delgado,63064075598184
Amy Francis,39529362233735764627
Sarah Williams,176
John Moody,8
Maria Patel,9572674
Jason Crane,0
Samantha Trujillo,65075521485
```

[82] Note that due to parser lookahead, the position may be slightly off the position of the actual error.

37.5.2 Fuzzing from a `.fan` String

We can also read a `.fan` spec from a string. This also demonstrates the usage of the `logging_level` parameter.

```
from fandango import Fandango
import logging

# Read in a .fan spec from a string
spec = """
    <start> ::= ('a' | 'b' | 'c')+
    where str(<start>) != 'd'
"""

fan = Fandango(spec, logging_level=logging.INFO)
for tree in fan.fuzz(desired_solutions=10):
    print(str(tree))
```

```
fandango:INFO: <string>: saving spec to cache /Users/zeller/Library/Caches/
 ↪Fandango/002ad60db75a3537e5414d7f02d18da44a27db8b3ce1cff886ccad45b7077409.pickle
```

```
fandango:INFO: Redefining <start>
```

```
fandango:INFO: ---------- Initializing base population ----------
```

```
fandango:INFO: ---------- Initializing FANDANGO algorithm ----------
```

```
fandango:INFO: ---------- Done initializing base population ----------
```

```
fandango:INFO: ---------- Generating ----------
```

```
fandango:INFO: Generating (additional) initial population (size: 100)...
```

```
bbbabac
cbcabbccaccacbba
caaaaaccaabab
caacab
cc
bb
ccbacccabbcabcbcb
abbbc
bccbbbbabbccba
babcaacbcbbbcaaba
```

37.5.3 Parsing an Input

This example illustrates usage of the `parse()` method.

```
from fandango import Fandango

spec = """
    <start> ::= ('a' | 'b' | 'c')+
    where str(<start>) != 'd'
"""
```

(continues on next page)

(continued from previous page)

```
fan = Fandango(spec)
word = 'abc'

for tree in fan.parse(word):
    print(f"tree = {repr(str(tree))}")
    print(tree.to_grammar())
```

```
tree = 'abc'
<start> ::= 'a' 'b' 'c'  # Position 0x0000 (0); abc
```

Use the *DerivationTree functions* (page 319) to convert and traverse the resulting trees.

37.5.4 Parsing an Incomplete Input

This example illustrates how to parse a prefix (`'ab'`) for a grammar that expects a final `d` letter.

```
from fandango import Fandango

spec = """
    <start> ::= ('a' | 'b' | 'c')+ 'd'
    where str(<start>) != 'd'
"""

fan = Fandango(spec)
word = 'ab'

for tree in fan.parse(word, prefix=True):
    print(f"tree = {repr(str(tree))}")
    print(tree.to_grammar())
```

```
tree = 'ab'
<start> ::= 'a' 'b'  # Position 0x0000 (0); ab
```

Without `prefix=True`, parsing would fail:

```
from fandango import Fandango

spec = """
    <start> ::= ('a' | 'b' | 'c')+ 'd'
    where str(<start>) != 'd'
"""

fan = Fandango(spec)
word = 'ab'

fan.parse(word)
```

```
<generator object Fandango.parse at 0x10dc3bef0>
```

37.5.5 Handling Parsing Errors

This example illustrates handling of `parse()` errors.

```
from fandango import Fandango, FandangoParseError

spec = """
    <start> ::= ('a' | 'b' | 'c')+
    where str(<start>) != 'd'
"""

fan = Fandango(spec)
invalid_word = 'abcdef'

try:
    fan.parse(invalid_word)
except FandangoParseError as exc:
    error_position = exc.position
    print("Syntax error at", repr(invalid_word[error_position]))
```

CHAPTER

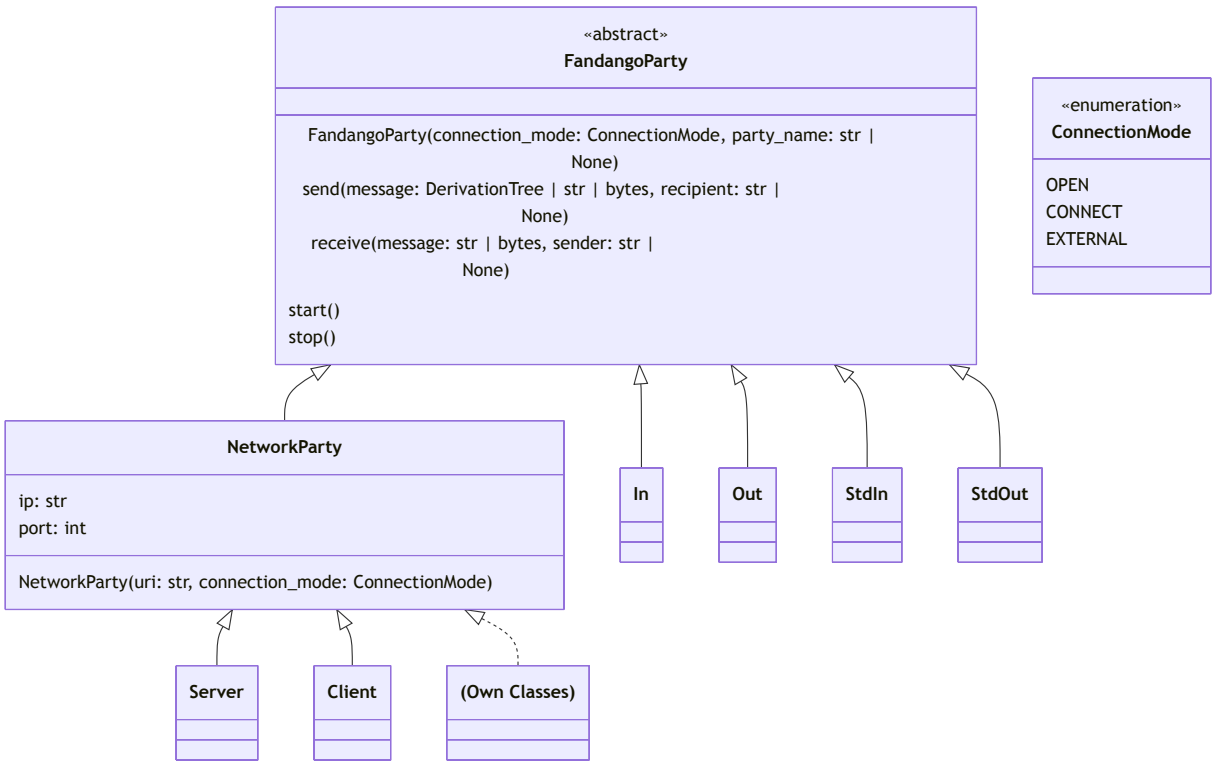
THIRTYEIGHT

FANDANGOPARTY REFERENCE

All communication between Fandango and its parties is established via `FandangoParty` classes.

Added in version 1.1: These features are available in Fandango 1.1 and later.

The following diagram illustrates the classes and methods discussed in this chapter:



All classes are predefined within `.fan` files; they need not be imported.

38.1 The `FandangoParty` class

`FandangoParty` is an abstract base class. It is meant to serve as base class for subclasses and cannot be directly instantiated.

38.1.1 Constructor

```
FandangoParty(connection_mode: ConnectionMode, party_name: str | None = None)
```

- `connection_mode`: Can be one of three values:
  - `ConnectionMode.OPEN` - Fandango behaves as a *server*. It opens a port and waits for incoming connections.
  - `ConnectionMode.CONNECT` - Fandango behaves as a *client*. It uses an existing port and connects to it.
  - `ConnectionMode.EXTERNAL` - The party is an external party; no connection is made by Fandango. Messages annotated with this party are not produced by Fandango, but are expected to be received from an external party.
- `party_name`: The name of the party. If `None`, the class name is used.

38.1.2 The `send()` method

On a `FandangoParty` object, the `send()` method is used to send a message (as a *`DerivationTree` object* (page 319)) to a party.

```
send(message: DerivationTree | str | bytes, recipient: Optional[str]) -> None
```

- `message: DerivationTree | str | bytes`: The message to send.
- `recipient: str`: The recipient of the message. Only present if the grammar specifies a recipient.

This method is typically overloaded and extended in subclasses. For instance, to apply a `compress()` method to every message sent, write

```
class CompressedNetworkParty(NetworkParty):
    def send(self, message: DerivationTree | str | bytes, recipient: Optional[str]) ->
↪ None:
        super().send(compress(message), recipient)
```

Important

Fandango itself will always invoke `send()` with a *`DerivationTree`* (page 319) as argument. However, the `FandangoParty` classes all support sending `DerivationTree`, `str`, and `bytes` types, so you do not have to re-create a `DerivationTree` object when calling `send()`.

38.1.3 The `receive()` method

On a `FandangoParty` object, the `receive()` method is invoked when data has been received by the party.

```
receive(message: str | bytes), sender: Optional[str]) -> None
```

- `message: str | bytes`: The message received.
- `recipient: str`: The sender of the message.

This method is typically overloaded and extended in subclasses. For instance, to apply a `decompress()` method to every message received, write

```
class CompressedNetworkParty(NetworkParty):
    def receive(self, message: DerivationTree, sender: Optional[str]) -> None:
        super().receive(uncompress(message), sender)
```

38.1.4 The `start()` and `stop()` methods

On a `FandangoParty` object, the `start()` and `stop()` methods are invoked to establish or shut down a connection.

```
start() -> None
stop() -> None
```

These methods can be used in subclasses to add additional behavior.

38.1.5 The `instance()` class method

`FandangoParty` provides an `instance()` class method that retrieves the last created instance of the given class.

```
instance(party_name: str | None = None) -> FandangoParty
```

`instance()` is typically invoked prefixed with the respective class, as in

```
Client.instance().stop()  # Stop the client
```

One can also invoke it with a party name; in that case, the class is ignored (use `FandangoParty`), and `instance()` retrieves the instance created with that name.

```
FandangoParty.instance("Client").stop()  # Equivalent to the above.
```

38.2 The `NetworkParty` class

The Fandango `NetworkParty` class (a subclass of `FandangoParty`) implements an Internet connection to a communication party. It implements the `FandangoParty` methods, as described above, plus the following methods:

38.2.1 Constructor

```
NetworkParty(uri: str, connection_mode: ConnectionMode = ConnectionMode.CONNECT)
```

- `uri` (string): The party specification of the party to connect to. Format: `[NAME=][PROTOCOL:][//][HOST:]PORT`.
  - `NAME`: Party name. Default: the class name.
  - `PROTOCOL`: `tcp` (default) or `udp`.
  - `HOST`: host name; use `[...]` for IPv6 host names. Default: 127.0.0.1
  - `PORT`: the port to connect to (0-65535)
- `connection_mode`: Can be one of three values:
  - `ConnectionMode.OPEN` - Fandango behaves as a *server*. It opens the given URI and waits for incoming connections.
  - `ConnectionMode.CONNECT` - Fandango behaves as a *client*. It connects to the given URI.
  - `ConnectionMode.EXTERNAL` - The party is an external party; no connection is made by Fandango. Messages annotated with this party are not produced by Fandango, but are expected to be received from an external party.

38.2.2 Properties

```
ip: str
```

The IP address or host used. Can be assigned to.

```
port: int
```

The port. Can be assigned to.

38.3 Predefined Parties

The following parties are predefined in Fandango:

38.3.1 `In` and `Out`

`In` and `Out` are `FandangoParty` classes connecting to the standard input and the standard output of an invoked program.

```
In()
```

Constructor.

```
Out()
```

Constructor.

38.3.2 `StdIn` and `StdOut`

`StdIn` and `StdOut` are `FandangoParty` classes connecting to the standard input and the standard output of Fandango.

```
StdIn()
```

Constructor.

```
StdOut()
```

Constructor.

38.3.3 `Client` and `Server`

`Server` are `NetworkParty` classes created with the `--server` option on the `fandango` executable command line.

If `--client URI` is given, `fandango` becomes a `Client`, and the classes are created as

```
class Client(NetworkParty):
    def __init__(self):
        super().__init__(
            URI,  # the argument in `--client URI`
            connection_mode=ConnectionMode.CONNECT,
        )
        self.start()

class Server(NetworkParty):
    def __init__(self):
        super().__init__(
            URI,  # the argument in `--client URI`
            connection_mode=ConnectionMode.EXTERNAL,
        )
        self.start()
```

If `--server URI` is given, `fandango` becomes a `Server`, and the classes are created as

```
class Client(NetworkParty):
    def __init__(self):
        super().__init__(
            URI,  # the argument in `--server URI`
            connection_mode=ConnectionMode.EXTERNAL,
        )
        self.start()

class Server(NetworkParty):
    def __init__(self):
        super().__init__(
            URI,  # the argument in `--server URI`
            connection_mode=ConnectionMode.OPEN,
        )
        self.start()
```

CHAPTER

THIRTYNINE

FANDANGO FILE LOCATIONS

39.1 Where Does Fandango Look for Included Specs?

In a Fandango spec, *the `include()` function* (page 155) searches for Fandango files in a number of locations. The actual rules for where Fandango searches are complex, versatile, and adhere to common standards. Specifically, when including a `.fan` file using `include()`, Fandango searches for `.fan` files in the following locations, in this order:

1. In any directory `DIR` explicitly specified by `-I DIR` or `--include-dir DIR`.
2. In any directory specified by the `$FANDANGO_PATH` environment variable, if set. This variable is a colon-separated list of directories, e.g. `$HOME/my_cool_fan_specs:/Volumes/Fandango:/opt/local/fandango-1.27` which are searched from left to right.
3. In the directory of the *including file*. This is the most common usage. If the including file has no file name (say, because it is a stream), the current directory is used.
4. In the directory `$HOME/.local/share/fandango`. You can control this location by setting the `$XDG_DATA_HOME` environment variable; see the XDG Base Directory Specification[83]. On a Mac, the location `$HOME/Library/Fandango` is searched first.
5. In one of the system directories `/usr/local/share/fandango` or `/usr/share/fandango`. You can control these locations by setting the `$XDG_DATA_DIRS` environment variable; see the XDG Base Directory Specification[84]. On a Mac, the location `/Library/Fandango` is searched first.
6. If the file to be included cannot be found in any of these locations, a `FileNotFoundError` is raised.

And then, there are two extra rules:

- If the included file has an *absolute* path (say, `/usr/local/specs/spec.fan`), only this absolute path will be used.
- If the included file has a *relative* path (say, `../spec.fan`), then this relative path will be applied to the locations above. This is mostly useful when you have a path relative to the including file.

Tip

We recommend to store included Fandango specs either

- in the directory where the *including* specs are, or
- in `$HOME/.local/share/fandango` (or `$HOME/Library/Fandango` on a Mac).

[83] https://specifications.freedesktop.org/basedir-spec/latest/
[84] https://specifications.freedesktop.org/basedir-spec/latest/

Finally, keep in mind that `include()` is a Python function, so in principle, your spec can compute the file name and location in any way you like. The simplest solution, however, is to use one of the "standard" locations as listed above.

39.2 Where Does Fandango Store Parsed Files?

Fandango maintains a *cache* for parsed `.fan` files in order to retrieve them faster.

Use `fandango clear-cache -n` to retrieve its location:

```
$ fandango clear-cache -n
```

```
Would clear /Users/zeller/Library/Caches/Fandango
```

If you want to reclaim space, or if there is some incompatibility between parsed and cached files, use `fandango clear-cache` to clear the cache:

```
$ fandango clear-cache
```

```
Clearing /Users/zeller/Library/Caches/Fandango
```

39.3 Where is Fandango installed?

This depends on your Python installation. You can query the location using

```
$ pip show fandango-fuzzer | grep Location
```

```
Location: /Users/zeller/.virtualenvs/python3.13/lib/python3.13/site-packages
```

CHAPTER

FORTY

FANDANGO LANGUAGE SERVER

The Fandango project includes a simple language server that makes writing Fandango grammars significantly easier. Currently, it supports the following:

- Autocomplete of nonterminals
- Jump to definition of nonterminal
- Find all references of nonterminal
- Rename nonterminal
- Create a definition for an undefined nonterminal
- Initial Semantic Token support

Under Construction

The language server is experimental.

Download or clone the Fandango source repository[85].

Start the language server by running

```
$ python3 src/fandango/language/server/language_server.py
```

40.1 Visual Studio Code Extension

A simple extension for Visual Studio Code (based on the one provided by the `pygls` project) is provided in the source repository in `.vscode/extensions/fandango-language-server`. To use it, clone the source repository, and compile it by running the following:

```
$ cd .vscode/extensions/fandango-language-server
$ npm install --no-save
$ npm run compile
```

You can then install it from the workspace-recommended extension section in the extension manager. The extension will automatically start the language server and interface with it once you start editing `.fan` files in this workspace. For additional documentation, refer to `.vscode/extensions/fandango-language-server/README.md`.

[85] https://github.com/fandango-fuzzer/fandango

40.2 IntelliJ / Pycharm Code Extension

To integrate the Fandango language server into IntelliJ or PyCharm, follow these steps:

1. Download and install the LSP4IJ plugin[86] for IntelliJ / Pycharm.
2. Download the Fandango LSP4J Language Server configuration.
3. In your IDE, navigate to the `Language Servers` tab.
4. Click the + button or right-click an empty area in the list to add a new language server.
5. In the `Add New Language Server` popup:
    1. For `Template`, select `Import from custom template...`
    2. A file browser will appear. Select the previously downloaded language server configuration.
    3. If necessary, adjust the command in the Server tab to ensure the correct Python executable is used (matching your desired Python version and virtual environment).

The extension will automatically start and interface with the language server.

Running the Language Server Manually

To manually run the language server, run the following Python script:

```
$ python3 [fandango-source-repository]/src/fandango/language/server/language_server.py
```

[86] https://plugins.jetbrains.com/plugin/23257-lsp4ij

BIBLIOGRAPHY

[BZ24] Leon Bettscheider and Andreas Zeller. Look ma, no input samples! mining input grammars from code with symbolic parsing. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, FSE 2024, 522–526. New York, NY, USA, 2024. Association for Computing Machinery. URL: https://doi.org/10.1145/3663529.3663790, doi:10.1145/3663529.3663790[5].

[GMZ20] Rahul Gopinath, Björn Mathis, and Andreas Zeller. Mining input grammars from dynamic control flow. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, 172–183. New York, NY, USA, 2020. Association for Computing Machinery. URL: https://doi.org/10.1145/3368089.3409679, doi:10.1145/3368089.3409679[6].

[HZ20] Nikolas Havrikov and Andreas Zeller. Systematically covering input structure. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ASE '19, 189–199. IEEE Press, 2020. URL: https://doi.org/10.1109/ASE.2019.00027, doi:10.1109/ASE.2019.00027[7].

[KLS22] Neil Kulkarni, Caroline Lemieux, and Koushik Sen. Learning highly recursive input grammars. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering*, ASE '21, 456–467. IEEE Press, 2022. URL: https://doi.org/10.1109/ASE51524.2021.9678879, doi:10.1109/ASE51524.2021.9678879[8].

[SchroderC22] Michael Schröder and Jürgen Cito. Grammars for free: toward grammar inference for ad hoc parsers. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '22, 41–45. New York, NY, USA, 2022. Association for Computing Machinery. URL: https://doi.org/10.1145/3510455.3512787, doi:10.1145/3510455.3512787[9].

[SteinhofelZ22] Dominic Steinhöfel and Andreas Zeller. Input invariants. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, 583–594. New York, NY, USA, 2022. Association for Computing Machinery. URL: https://doi.org/10.1145/3540250.3549139, doi:10.1145/3540250.3549139[10].

[SteinhofelZ24] Dominic Steinhöfel and Andreas Zeller. Language-based software testing. *Commun. ACM*, 67(4):80–84, March 2024. URL: https://doi.org/10.1145/3631520, doi:10.1145/3631520[11].

[ZASZ25] José Antonio Zamudio Amaya, Marius Smytzek, and Andreas Zeller. FANDANGO: Evolving language-based testing. *Proc. ACM Softw. Eng.*, June 2025. URL: https://doi.org/10.1145/3728915, doi:10.1145/3728915[12].

[5] https://doi.org/10.1145/3663529.3663790
[6] https://doi.org/10.1145/3368089.3409679
[7] https://doi.org/10.1109/ASE.2019.00027
[8] https://doi.org/10.1109/ASE51524.2021.9678879
[9] https://doi.org/10.1145/3510455.3512787
[10] https://doi.org/10.1145/3540250.3549139
[11] https://doi.org/10.1145/3631520
[12] https://doi.org/10.1145/3728915