Pithya User Manual

January 25, 2017

Contents

Chapter 1

Introduction

Pithya is a tool for parameter synthesis of ODE-based models of dynamical systems and properties specified in a hybrid extension of Computational Tree Logic (CTL) with past called HUCTL_{P} [BBD+16b]. Figure 1.1 depicts the architecture of the Pithya tool. The tool consists of three parts: the main part composed of several stand-alone executables, the graphical user interface (GUI) used for model design and result visualisation, and the command-line interface (CLI).

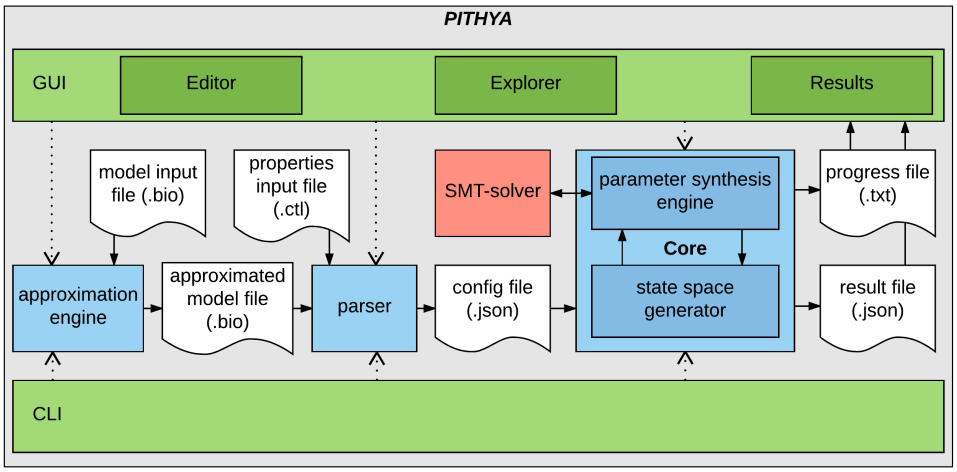

Figure 1.1: Basic architecture of Pithya with its main parts.

To perform the parameter synthesis task, the user has to provide two inputs: an ODE-based model and a set of properties of interest. The *model input file* (`.bio`) is written as a set of equations with parameters; the syntax

of the equations is given in Section 3.2. The *properties input file* (`.ctl`) is written as a set of HUCTL$_{\text{P}}$ formulae; the syntax of the formulae is given in Section 3.3.

The model input file is first run through the *approximation engine*, a stand-alone executable that approximates the functions appearing in the ODEs of the original model into a piecewise multi-affine model (using the technique defined in [GBF$^+$11]). The accuracy of the approximation is guided by the settings provided in the model input file (see Section 3.2). The resulting *approximated model file* is again expressed in terms of the `.bio` file format.

The result of the approximation is then combined with the properties input file and run through the *parser*, another stand-alone executable that verifies the syntax of the provided formulae with respect to the approximated model and prepares a *configuration file* to be used by the core executable. This file is written in the JSON format [Bra14] and includes both the input model and the properties of interest in a machine-readable way.

The configuration file is then presented to the *core engine*. The core engine consists of two parts: the *state-space generator* employs the rectangular abstraction (as defined in [BYWB07, GBF$^+$11]) to discretise the ODE model; the *parameter-synthesis engine* then performs the parameter synthesis based on the parallel semi-symbolic coloured model checking [BBD$^+$16a, BBD$^+$16b]. To do so, the engine employs an external SMT solver (we currently use Z3 [dMB08]).

While the core engine is running, it sends *progress information to the standard output*. After the parameter synthesis task has finished, the results are produced in one of the two available output formats: JSON (to be used for further processing in the GUI) or textual human-readable form.

The *command-line interface* (CLI) encapsulates all the stand-alone executables so that the user only needs to provide the model input file and the properties file. The executables are run automatically and provide the result of the parameter synthesis in the selected output format. The various CLI options are described in detail in Chapter 4.

The *graphical user interface* (GUI) consists of three parts. The *editor* allows the user to load, edit and save the description of the model and the properties of interest. The *explorer* is used to investigate the model behaviour and its approximated transition state space. The *results visualiser* provides an interactive visual analysis of the parameter synthesis results. For more information about the possibilities of the GUI see Chapter 5.

Chapter 2

Installation

2.1 Download

The latest version of the Pithya core (including the approximation engine, the parser and the CLI) can be downloaded from:

`https://github.com/sybila/pithya-core`

The GUI part can be downloaded from:

`https://github.com/sybila/pithya-gui`

The Pithya core and CLI is written in Java so it can be used on any desktop platform supporting Java. The GUI is written in the *R* language, which can be downloaded for any of the widely-used OSs from:

`https://cran.r-project.org/mirrors.html`

Follow the instructions in Section 2.3.

2.2 Dependencies

To run Pithya with the CLI, you need to have `Java 8+` and `Microsoft Z3 4.5.0` installed. If your OS is supported, we strongly recommend downloading precompiled `Z3` binaries from:

`https://github.com/Z3Prover/z3/releases`

(Pithya allows to specify a custom `Z3` location, so it is not necessary to have it set in `PATH`.)

The GUI needs `R version 3.3.+` with the following libraries installed:

- `shiny 0.14.2`
- `shinyjs 0.8`
- `shinyAce 0.2.1`

- `shinyBS 0.61`
- `stringr 1.1.0`
- `rjson 0.2.15`
- `pracma 1.9.5`
- `data.table 1.9.8`
- `matrixcalc 1.0-3`

at least in these versions with all of their dependencies. For more information about the installation procedure see the following section.

2.3 Build

2.3.1 Core and CLI

Pithya can be directly built from source if needed. After the repository `https://github.com/sybila/pithya-core` is cloned to your file system, you can run one of the following commands in the root folder of the tool (on Windows, replace `./gradlew` with `./gradlew.bat`):

- To build Pithya and place the unpacked distribution into `./build/install/pithya`

  `./gradlew installDist`

- To build Pithya and place the compressed distribution into `./build/distributions`

  `./gradlew distZip`

(If you have a local gradle[1] *installation, you can replace* `./gradlew` *with* `gradle` *for faster build.)*

2.3.2 GUI

First clone the repository `https://github.com/sybila/pithya-gui` into the same folder where the folder `pithya-core` from the previous section is located. To use the GUI you need to install the core part as described above and place the bin and lib folders from `build/install/pithya` into `pithya-gui/core`. To run the GUI you further need to install the *R* language including the libraries specified in Section 2.2. The *R* language tool suite is available at `https://cran.r-project.org/mirrors.html` with guides and advice for all usual OSs.

[1] *gradle* is a build automation system for Java, see `https://github.com/gradle/gradle`

If you have super-user rights, running the command

`Rscript /path/to/pithya-gui/runMe.R`

will automatically install all needed packages. If you do not have super-user rights, you need to install the packages manually by running the command `R` that opens R interactive mode. Then call the following:

`install.packages("`*`name_of_package`*`")` – for each of the packages. According to the developers, you are advised to run the `update.packages()` command before `install.packages` to ensure that any already installed dependencies have their latest versions. To close the interactive `R` mode you have to type `q()`.

Finally, to run the GUI go into the `pithya-gui` folder and run the command:

`Rscript runMe.R`

This starts the application server in the background. Open a browser and point it to the address: `http://127.0.0.1:8080`, the GUI should appear. It is possible to open more than one session in this way. To end a session just close the tab. To end the application server press Ctrl+C inside the terminal.

Chapter 3

Common information

3.1 Executables

Apart from the main binary, Pithya also contains these separate utilities:

- `approximation engine` takes a path to one `.bio` file as a command line argument, performs the piecewise multi-affine approximation and prints the resulting `.bio` model file to standard output. This utility can be used as a `.bio` syntax checker.
- `parser` takes a path to the (already approximated) `.bio` model and the `.ctl` property file as command line arguments and prints a configuration JSON file. This utility can be used to verify that your model and property files are valid.

3.2 Model Syntax

This section describes the syntax of model files (the `.bio` format). Every model file has to contain at least the following parts: declaration of model variables and corresponding thresholds (at least two numeric values have to be defined as the lower and upper bound thresholds for each variable), declaration of parameters (each with a lower and upper bound), and differential equations (one for each variable). The corresponding lines start with predefined keywords and have the following syntax:

- `VARS: variables` (mandatory, just one occurence) where `variables` is a list of variable names delimited by comma (`,`).

- `PARAMS: parameters` (mandatory, just one occurence) where `parameters` is a list of expressions delimited by semicolon (`;`) in the form `param_name, lower_bound, upper_bound`.
- `CONSTS: constants` (optional, just one occurence) where `constants` is a list of named constants delimited by semicolon (`;`) in the form `constant_name, constant_value`.
- `EQ: equation` (mandatory, one for each model variable) where `equation` has the form `variable_name = equation_expression` that is defined below.
- `THRES: thresholds` (mandatory, one for each model variable) where `thresholds` has the form `variable_name: threshold_values` and `threshold_values` is a list of at least two `numeric_values` delimited by comma (`,`).
- `VAR_POINTS: var_points` (optional, just one occurence) where `var_points` is a list delimited by semicolons (`;`) of the form `variable_name: valuation_points, ramps_count`. Here, `valuation_points` define the accuracy of the approximation and `ramps_count` define the number of additional thresholds determining the accuracy of the parameter synthesis process; they both have default values.

In the syntax above, the terms `lower_bound`, `upper_bound`, `constant_value` and `numeric_values` are either integers or floating-point numbers (the scientific notation is not supported), while the terms `valuation_points` and `ramps_count` are integers only. *Note that lines do not end with a semicolon (;) or any other special ending character.*

The syntax of `equation_expression` is defined as an arithmetic expression that uses only the operations `+` (addition), `-` (subtraction or unary negation), and `*` (multiplication). The operands can be either numerical values, variables, constants, parameters, or the application of one of the following functions:

- `[Hillp|Hillm](var, thres, factor_e, factor_a, factor_b)` is the so-called *Hill function* [KS09] in positive form (`Hillp`) or in negative form (`Hillm`); they are defined in the following way:

$$\texttt{Hillp} = \texttt{factor\_a} + \frac{(\texttt{factor\_b} - \texttt{factor\_a})\texttt{var}^{\texttt{factor\_e}}}{\texttt{var}^{\texttt{factor\_e}} + \texttt{thres}^{\texttt{factor\_e}}},$$

$$\texttt{Hillm} = \texttt{factor\_a} - \frac{(\texttt{factor\_a} - \texttt{factor\_b})\texttt{thres}^{\texttt{factor\_e}}}{\texttt{var}^{\texttt{factor\_e}} + \texttt{thres}^{\texttt{factor\_e}}}.$$

According to the usual notation, the parameters: `var`, `thresh`, `factor_e` have the meaning of the parameters S, K_m, n, respectively. The value of the *Hill function* is scaled from `factor_a` to `factor_b` instead of from 0 to 1. If `factor_a` is 0 then `factor_b` acts as $V_{\max}$.

- `[Hp|Hm](var, thres, factor_a, factor_b)` is the so-called *Heaviside step function* [KS09] in increasing/positive form (`Hp`) or in decreasing/negative form (`Hm`); they are defined in the following way:

$$\texttt{Hp} = \begin{cases} \texttt{factor\_a}, & \texttt{var} < \texttt{thres} \\ \texttt{factor\_b}, & \texttt{var} \geq \texttt{thres} \end{cases},$$

$$\texttt{Hm} = \begin{cases} \texttt{factor\_b}, & \texttt{var} < \texttt{thres} \\ \texttt{factor\_a}, & \texttt{var} \geq \texttt{thres} \end{cases}.$$

- `[Sm|Sp](var, factor_k, thres, factor_a, factor_b)` is the so-called *sigmoidal function* [GBF+11] in increasing/positive form (`Sp`) or in decreasing/negative mode (`Sm`); they are defined in the following way:

$$\texttt{Sp} = \texttt{factor\_a} + (\texttt{factor\_b} - \texttt{factor\_a})\frac{1 + \tanh(\texttt{factor\_k}(\texttt{var} - \texttt{thres}))}{2},$$

$$\texttt{Sm} = \texttt{factor\_b} + (\texttt{factor\_a} - \texttt{factor\_b})\frac{1 + \tanh(\texttt{factor\_k}(\texttt{var} - \texttt{thres}))}{2},$$

where `thres` and all `factors` have to be either numbers or named constants (defined using the `CONSTS` keyword) and `thres` has to be a valid threshold of the variable `var`.

Example

```
VARS: x, y
CONSTS: k2, 1; deg_y, 0.1
PARAMS: k1, 0, 2; deg_x, 0, 1
EQ: x = k1*Hillm(y, 5, 5, 1, 0) - deg_x*x
EQ: y = k2*Hillm(x, 5, 5, 1, 0) - deg_y*y
VAR_POINTS: x: 1500, 10; y: 1500, 10
THRES: x: 0, 15
THRES: y: 0, 15
```

3.3 Properties Syntax

This section describes the syntax of the properties file (the `.ctl` format). The first subsection describes the usage of the standard Computational Tree Logic (CTL) [CGP99]. The second subsection then describes the usage of HUCTL$_P$ [BBD+16b], the logic that extends CTL with event predicates, hybrid operators and past. The event predicates extension allows to reason about directions of the flow in the model and the hybrid extension allows the use of state variables that can be fixed in certain parts of the formula as well as quantified.

3.3.1 CTL

Each `.ctl` file contains one or more *assignment statements*. The statements are separated by new lines or semicolons (`;`). Comments can be written using `C`-style (`//`) or `Python`-style (`#`). Multi-line formulae are not supported, use references instead. The *assignment statement* has one of the forms:

- `identifier = formula`
- `identifier = expression`

where `identifier` is a sequence of alphanumeric characters starting with an alphabetic character. The term `formula` can be given as:

- an `identifier` (reference to `formula` defined elsewhere);
- a boolean constant: `True`, `False` (alternatively `tt`, `ff`);
- a float proposition: two expressions compared using one of the operators `<`, `>`;
- a direction proposition: `id:direction facet` where `id` is the name of a *model variable* (see Section 3.2), direction is either `in` or `out` and facet can be positive (symbol `+`) or negative (symbol `-`) in the meaning of upper or lower bound of states (e.g., `var:in+`);
- another `formula` in parentheses;
- a formula with a unary operator applied; the unary operators are `!` (negation), `EX`, `AX`, `EF`, `AF`, `EG`, `AG`;
- two formulae with a binary operator applied in infix notation `f1 op f2`, where `op` stands for one of `&&` (conjunction), `||` (disjunction), `->` (implication), `<->` (equivalence), `EU`, `AU`.

The term `expression` can be given as:

- `identifier`: a reference to an `expression` defined elsewhere or a *model variable* (*Note that identifiers that can't be resolved are considered as variables and produce no error or warning.*);
- a numeric constant (integer or floating-point; scientific notation not supported);
- `expression` in parentheses.

The special flag `:?` in front of a formula assignment indicates formulae that are going to be investigated by the parameter synthesis process.

Operator priority In general, unary operators have higher priority than binary operators. Binary operators have the following priority:
`&&` ≫ `||` ≫ `->` ≫ `<->` ≫ `EU` ≫ `AU`
Also note that `->`, `EU` and `AU` are right associative, so that `a EU b EU c` is parsed as `a EU (b EU c)`, as expected.

Examples

- `f = (AG ((q / Var1) EU 2 * Var2)) || (EG (p2 AU b))` FIX THIS!
- `p1 = val > 3.14`
- `:? p2 = val2 < -44`
- `p2 = var:in+ && var:in-`
- `:? a = EF (p1 \&\& p2)`
- `b = AF foo`

3.3.2 HUCTL$_P$

This subsection defines the extensions of syntax in Section 3.3.1, i.e. everything defined in Section 3.3.1 also applies for HUCTL$_P$. The *assignment statement* has one new form:

- `identifier = direction_formula`

where `direction_formula` stands for:

- an `identifier` (reference to a `direction_formula` defined elsewhere);
- a boolean constant: `True`, `False` (alternatively `tt`, `ff`);

- a loop atomic proposition: `loop` or `Loop`;
- another `direction_formula` in parentheses;
- a direction proposition: `id direction` where `id` is the name of a *model variable* (see Section 3.2) and direction can be positive (symbol `+`) or negative (symbol `-`) in the meaning of the increase or decrease of the *model variable* value (e.g., `var-`);
- a `direction_formula` with the unary operator `!` (for negation) applied;
- two `direction_formulae` with a binary operator applied in infix notation `f1 op f2`, where `op` stands for one of `&&` (conjunction), `||` (disjunction), `->` (implication), `<->` (equivalence).

The *assignment statement* form `formula` is extended by the following expressions:

- a `formula` with a unary temporal operator:
  `modifier quantifier op formula`
  where `modifier` is optional and stands for `{ direction_formula }`, `quantifier` is either future (`A` or `E`) or past (`pA` or `pE`) and `op` stands for one of `X`, `F`, `G`, or the new operators `wF` (weak Future) and `wX` (weak neXt), e.g., `{var+ && !var-} pAX f1`, `EwF f2`;
- two `formulae` with a binary temporal operator in infix notation:
  `f1 modifier1 quantifier op modifier2 f2`
  `modifier1` and `modifier2` are both optional and stand for `{ direction_formula }`, `quantifier` is either future (`A` or `E`) or past (`pA` or `pE`) and `op` stands for `U`, e.g., `f1 {var+} AU {!var+} f2`;
- a `formula` with hybrid first-order quantifiers:
  `quantifier id bound: f1`
  where `quantifier` stands for `forall` or `exists`, `id` is a fresh hybrid variable name, `bound` (optional) stands for `in f2` and `f1` and `f2` are defined as `formula`,
  e.g., `exists s in (EF AG Var > 5.3): !{Var-} EF s`;
- a `formula` with hybrid operators: `op id : formula`
  where `op` stands for `at` or `bind`, `id` is a hybrid variable name (fresh in case of `bind`, previously bound in case of `at`), e.g., `bind x : AG EF x`.

For more information about semantics of new operators, quantifiers etc. look into [BBD+16b].

Examples

- `:? sink = bind x : AX x`
- `limit_cycle = bind s : EX EF x`
- `saddle = {var1+ || var1-} AX tt && {var1+} EX tt && {var1-} EX tt && {var2+ || var2-} pAX tt && {var2+} pEX tt && {var2-} pEX tt`
- ```
  stable = bind x : AG EF x
  :? bistable = exists s in stable : exists t in stable :
      {!E2F1+} EF s && {!E2F1-} EF t && at t : !EF s
  ```

3.4 Results Output Format

Basically, there are two results output formats. One for human eye (text format) and other one easily parsed by computer (JSON format) but they both contain the same message and that is satisfying states list of abstracted model with particular satisfying parameterisations for all formulae of interest.

3.4.1 Text Format

The text format represents all states of the abstraction satisfying the given formula (which name is on separate line before list of satisfying states). Each state is associated with satisfying parameterisations. The form of results differs for models with independent parameters and models with dependent parameters.

In the case of independent parameters, the syntax of every line of the output is the following:

`State(state_id)`$[\mathtt{x}_1^{\mathtt{b}}, \mathtt{x}_1^{\mathtt{u}}]$`, ...,` $[\mathtt{x}_n^{\mathtt{b}}, \mathtt{x}_n^{\mathtt{u}}]$ `-> Params(`$[[\mathtt{p}_1^{\mathtt{b}}, \mathtt{p}_1^{\mathtt{u}}, ..., \mathtt{p}_m^{\mathtt{b}}, \mathtt{p}_m^{\mathtt{u}}]]$`)`

where

- `state_id` is the unique ID of the discrete state,
- $\mathtt{x}_1, ..., \mathtt{x}_n$ are the model variables,
- $\mathtt{x}_i^{\mathtt{b}}, \mathtt{x}_i^{\mathtt{u}}$ denote the upper and the lower threshold of the state,
- $\mathtt{p}_1, ..., \mathtt{p}_m$ are the uncertain parameters,
- $\mathtt{p}_i^{\mathtt{b}}, \mathtt{p}_i^{\mathtt{u}}$ denote the upper and the lower bound of the satisfying interval of valuations for parameter $\mathtt{p}_i$, respectively.
  ```

In the case of dependent parameters, the syntax of every line of the output is the following:

`State(state_id)`$[x_1^b, x_1^u]$`, ...,` $[x_n^b, x_n^u]$ `-> Params({null:Formula})`

where `Formula` is the exact output from SMT solver in *SMT-LIB 2.0* format [BST+10] mathematically expressing satisfying parameterisations.

Note, that information about the computation is displayed too by default if you don't change CLI arguments settings (see Section 4.2).

3.4.2 JSON Format

This subsection defines output format for parameter synthesis procedure suitable for GUI (see Chapter 5). Let's call the whole content `ParamSet` list which has the form:

`{`

`"variables":[String],` - the array of variables names (e.g., `["x","y"]`)

`"parameters":[String],` - the array of parameter names (e.g., `["a","b"]`)

`"thresholds":[[Double]],` - the array where each element is another array of threshold values for some variable (inner arrays are ordered as variables) (e.g., `[[0,1.2,5],[0,1,2,3]]`)

`"parameter_bounds":[Pair],` - the array of pairs of bounding values for particular parameter (e.g., `[[0,1],[0,10]]`)

`"states":[State],` - the array of all states of abstracted model where `State` equals `{"id":Int,"bounds":[Pair]}` with `Int` standing for integer and `[Pair]` standing for the array of pairs of bounding thresholds for particular state in particular dimension/variable (pairs are ordered as variables)
(e.g., `[{"id":0,"bounds":[[0,1.2],[0,1]]},`
`{"id":1,"bounds":[[0,1.2],[1,2]]}]`)

`"type":Enum,` - where `Enum` is either `"rectangular"` (if the model contains just linearly-independent parameters) or `"smt"` (if not) (on this value depends the format of next list element)

`"parameter_values":[[[Pair]]],` - if `Enum` is `"rectangular"`; the array of all unique parameterisations: `[[Pair]]` - where each parameterisation is the union of $\mathcal{P}$-dimensional rectangles: `[Pair]` (where $\mathcal{P}$ is number of parameters) - and each rectangle is defined by the array

of pairs of bounding values in particular dimension/parameter (pairs are ordered as parameters)
(e.g., `[[[[0,0.2],[0,1]],[[0,0.1],[0,2]]],[[[0.1,0.3],[2,5]]]]`)

`"parameter_values":[Formulae],` - if `Enum` is `"smt"`; the array of all unique parameterisations where `Formulae` stands for `{"smtlib2Formula":String,"Rexpression":String}` - where the first `String` is the exact output of SMT solver in *SMT-LIB 2.0* format [BST+10] mathematically expressing satisfying parameterisations in the form of character string and the second `String` contains the same data but formated for GUI needs also in the form of character string

`"results":[Result]` - the array of all combinations of property, satisfying state and particular parameterisation where `Result` stands for `{"formula":String,"data":[IntPair]}` - where `String` is the `identifier` of a property from Section 3.3 and `"data"` contains the array of pairs of integers where the first value in particular pair is an index into `"states"` array and the second value is an index into `"parameter_values` array so the information in this list element is not redundant

`}`

Chapter 4

Command Line Interface

4.1 Run

CLI has one main binary `bin/pithya`. The bin folder also contains other executables, however, these are used directly only when pithya operates together with GUI (Chapter 5), so you don't need to worry about them (for more information about each see Section 3.1).

4.2 Arguments

4.2.1 Input and Output

- `[-m,--model] filePath` *required* Path to the `.bio` file from which the model should be loaded. Detailed description of the `.bio` format can be found in Section 3.2.
- `[-p,--property] filePath` *required* Path to the .huctl file from which verified properties are loaded. Detailed description of the `.ctl` format can be found in Section 3.3.
- `[-ro,--result-output] [stdout, stderr, filePath]` *default: stdout* File or stream to which verification results should be printed. *You can use this option to print log and results separately. Note: errors are always printed to stderr.*
- `[-r,--result] [human, json]` *default: human* Output format that is used when printing results.
  - `human` Text output in the form described in Section 3.4.1.

  - `json` A more formal output format that can be easily parsed by other tools. Description in Section 3.4.2.

- `[-lo,--log-output] [stdout, stderr, filePath]` *default: stdout* File or stream to which logging info should be printed. *You can use this option to print log and results separately. Note: errors are always printed to stderr.*

- `[-l,--log] [none, info, verbose, debug]` *default: verbose* Amount of log data to print during execution.

  - `none` No logging.
  - `info` Print coarse verification progress and statistics (started operators, final solver throughput).
  - `verbose` Print interactive progress with dynamic throughput statistics (roughly every 2s).
  - `debug` Print everything.

4.2.2 Verification Options

- `--parallelism integer` *default: runtime.avaialableProcessors* The maximum number of threads that are used for parallel computation (this is an upper bound, for some specific models or properties, desired level of parallelism might not be achievable).

- `--z3-path filePath` *default:* `z3` Relative or absolute path to the `z3` command line executable.

- `--transition-preprocessing [true, false]` *default: true* To speed up on-the-fly transition generator, parts of the transition system are evaluated before the verification starts. This might not be desirable when the model is very large, but the property is expected to require only small amount of explored state space. Use this flag to turn off this preprocessing step.

- `--fast-approximation [true, false]` *default: false* Uses much faster, but not necessarily optimal version of the PMA approximation when evaluating the model ODEs.

- `--create-self-loops [true, false]` *default: true* Creating selfloops can cause significant overhead even though they have no impact on some

types of properties (mainly reachability). You can disable selfloops using this switch.

4.3 Example of Use

- `./bin/pithya -m model.bio -p property.ctl -r human -lo stderr > result.txt 2> log.txt`
- `./bin/pithya -m model.bio -p property.ctl -r json -lo stderr --parallelism 4 > result.json`

Chapter 5

Graphical User Interface

5.1 GUI in General

At the top of the tool, there is a header with name and its description. Below, there is **Advanced Settings** checkbox which affects many other controllers across the tool GUI marked as *(only in advanced mode)* (see following sections). By default, it is switched off.

Below it there is a tab panel with three tabs each of which has its own purpose described in a respective section of this chapter.

5.2 Editor

This is the first tab and also the place where you should start. Here, model and properties are specified (or loaded) and the process of their exploration is initiated. Area is visually divided into two horizontal parts.

5.2.1 First Part

It is vertically divided into three sub-areas:

1. **Model Editor Control Panel**

    - **Browse...** button: is actually a file loader. It is responsible for loading of prepared or saved models in `.bio` format. When used, below it appears uploading progress bar and next to it the name of loaded file. In this way you can load any `.bio` file into editor. For syntax see Section 3.2.

- **Reload Model** button: allows to reset all changes done in the model since it's been loaded.
- **Save Model** button: calls file-download routine of your browser with current content of **Model Editor** (below).
- **Cut Thresholds** checkbox (only in advanced mode): if checked, thresholds computed by PMA approximation are cut by the maximal threshold explicitly defined in the model. In some cases the non-linear function (such as Hill function) could be defined beyond the maximal thresholds. Default settings is off provided that the thresholds exceeding the maximal threshold are automatically added to the model.
- **Fast Approximation** checkbox (only in advanced mode): sets the mode of PMA approximation to be used. Default is to use slower but more precise algorithm.
- **Generate Approximation** button: is the starting point for exploration of the model (see Section 5.3). It starts PMA approximation and also checks the model syntax (see Section 3.2). It turns green whenever the model is changed which indicating that approximation should be updated. In other words, the green status of this button indicates that the model no longer matches information available in the explorer and results tabs.

2. **Properties Editor Control Panel**

   - **Browse...** button: is actually a file loader. It is responsible for loading of prepared or saved properties in `.ctl` format. When used, below it appears uploading progress bar and next to it the name of loaded file. In this way you can load any `.ctl` file into editor. For syntax see Section 3.3.
   - **Reload Properties** button: allows to reset all changes done in the properties since they've been loaded.
   - **Save Properties** button: calls file-download routine of your browser with current content of **Properties Editor** (below).

3. **Parameter Synthesis Control Panel**

   - **Number of Threads** slider (only in advanced mode): defines number of threads to use for parameter synthesis process. By default, it is set to the maximum number of cores of your PC.

- **Run Parameter Synthesis** button: is the starting point for exploration of results (see Section 5.4). It starts parameter synthesis process and also checks properties syntax (see Section 3.3). It turns green when approximation is done or properties change or in case the previous run of parameter synthesis has not been successful indicating that it needs to be executed again. In other words, your properties description no longer matches the results computed before. After the process is finished a dialog box appears with this announcement. In case there are some thresholds in the properties that are missing in the model the process stops and a dialog box appears with notification that lets you decide whether missing threshold(s) should be added into the model and PMA approximation should be regenerated before parameter synthesis could start again.
- **Stop Parameter Synthesis** button: is enabled right after the parameter synthesis process started and along with the button turns orange. It allows to stop the currently running parameter synthesis process.

5.2.2 Second Part

It is vertically divided into two similar sub-areas:

1. first one is text editor for model description (`.bio` file) — labeled **Model Editor**. Changes in the content of the editor can be reverted (in terms of *undo*) by the shortcut `ctrl+z` and applied back (in terms of *redo*) by the shortcut `ctrl+shift+z` or `ctrl+y` (depending on your OS). For more information about correct syntax of model see Section 3.2.

2. second one is the text editor for properties (`.ctl` file) — labeled **Properties Editor**. Changes in the content of the editor can be reverted (in terms of *undo*) by the shortcut `ctrl+z` and applied back (in terms of *redo*) by the shortcut `ctrl+shift+z` or `ctrl+y` (depending on your OS). For more information about correct syntax of properties see Section 3.3.

5.3 Explorer

This is the second tab in the row and usually also the second one used, i.e., to explore the model and its dynamics while parameter synthesis is in progress.

You can visualise 2D vector fields plots (cut of phase space) and compare these with corresponding abstracted state-transition systems. In both cases on the axes of all plots, there are model variables defined in model editor. If the model contains just one variable the showing plots are considered to be 1D with only one axis displayed (for which the variable name is selected) and the second one empty (for which the constant *(none)* is selected). Each of the plots can be downloaded by clicking right mouse button inside the plot. Explorer area is visually divided into two horizontal parts.

5.3.1 First Part

It contains controllers for setting of visual output of all plots. It is vertically divided into 4 columns:

1. First column

    - **Arrows Count** slider: sets the number of arrows (per dimension) inside vector field(s).
    - **Colouring Threshold** slider: sets the maximal magnitude of vector in vector field(s) to be considered as neutral (in black). Any vector with value above this magnitude is considered as positive (in green) and any vector with value below negative value of this magnitude is considered as negative (in red).
    - **Colouring Orientation** radiobutton: sets the vector field arrow component(s) to which colouring is applied. Horizontal, vertical, both (simple addition of both) or none (all vector arrows are black).

2. Second column

    - **Arrows Length** slider: sets the scaling factor for length of arrows in vector field(s).
    - **Arrows Width** slider: sets the scaling factor for width of arrows in vector field(s) and the arrows representing transitions in discrete state space(s).

3. Third column

    - **Trajectory Points Number** numeric input (only in advanced mode): sets the number of steps from the starting point (see

click-inside-plot settings in Section 5.3.2) simulating vector field trajectory. Increase to prolong the trajectory. Default is 500.

- **Trajectory Points Scaling Factor** slider (only in advanced mode): sets accuracy (or precision) of vector field trajectory. Actually, it is scaling factor for computation of steps in phase space trajectory. Hence, the smaller is the factor the more accurate is the trajectory. Default is 1.

4. Fourth column

   - first, it looks empty and needs approximation to be correctly done. After that some controller(s) for setting up precise value of each parameter appear(s) (according to number of parameters defined in model) - labeled **Parameter XYZ**. This control of parameters gives you the opportunity to explore behaviour of the model with your naked eye.

5.3.2 Second Part

At first, there is just one button - **Add Plot** - which is disabled until approximation is done correctly.

Basically, it is an area showing several similar rows of plots with their personal controllers and you can set each row separately but the possibilities of controllers are the same for each row in general. Therefore, we describe just one row. Right after **Add Plot** button is clicked new row appears below the others. Each row contains visually highlighted setting area at the top of the row and below it the visual area. Setting area compounds of:

- **Horizontal Axis** selector: sets variable for horizontal axis of particular pair of plots. Defaultly, it is filled with first variable according to the model description in model editor (see Section 5.2.2).
- **Vertical Axis** selector: sets variable for vertical axis of particular pair of plots. Defaultly, it is filled with second variable according to the model description in model editor (see Section 5.2.2). If this is not possible (model has just one variable) the constant *(none)* is selected for this axis and plots are displayed in 1D mode.
- **Delete** button: deletes particular pair of plots with all of their settings.
- **Hide** checkbox: temporary hides particular pair of plots with their controllers. They can be shown at your command by unticking this checkbox.

Visual area contains pair of plots (left-hand with vector field and right-hand with transitions-state space) with their personal controllers. Explicitly:

1. First column (controllers for manipulation with vector field of the model). For more information about *click-inside-plot settings* see description of vector field area (Second column)

   - **Apply to All VF** button: applies *click-inside-plot settings* to all shown vector field plots.
   - **Apply to TSS** button: applies *click-inside-plot settings* to its corresponding state space plot.
   - **Clear Plot** button: deletes *click-inside-plot settings* of this plot.
   - **Unzoom** button: sets axes of particular plot to the default ranges.
   - **Use PMA Model** checkbox (only in advanced mode): switches a model used for computing of vector fields plot. Defaultly, it is used a model before PMA approximation but it might be interesting to compare the differencies before and after PMA approximation which is necessary for computing of state space.
   - set of scale sliders labeled **Continuous Value of XYZ**: one for each variable not shown in particular plot so you could investigate each corner of vector field. Each slider is in range of minimal and maximal thresholds after PMA approximation. If there is no unshown variable, there is also no scale slider.
   - text field: shows all variables on separate line with particular value you are looking at in the plot. For variables used in axes, coordinates are shown right after you move mouse inside this plot.

2. Second column (vector field area labeled**ODE Model Vector Field**)

   it contains a plot of vector field which is a cut of phase space of the model. This plot is shown for variables selected in particular selectors. Click left mouse button in plot and drag over to zoom in selected area. For zooming out use **unzoom** button. By double-clicking inside the plot (without moving your mouse) you select specific starting point from which a phase space trajectory is projected (this point is set as *click-inside-plot settings* of this plot). Note, that such trajectory is projection across all dimensions of the phase space and its length can be set by **Trajectory Points Number** and **Trajectory Points Scaling Factor** controllers (only in advanced mode).

3. Third column (transition-state space area labeled **Transition-State Space**)

   it contains a plot of state space of the model with transitions (selfloops are displayed as black bold points). This plot is shown for variables selected in particular selectors. You can zoom in by clicking left mouse button and moving your mouse until you release the button. For zooming out use **unzoom** button. By double-clicking inside the plot (without moving your mouse) you select specific starting state from which all reachable states in this cut are highlighted (this state is set as *click-inside-plot settings* of this plot). Note, that displayed reachable area is not complete if model contains more than two variables.

4. Fourth column (controllers for manipulation with transition state space plot). For more information about *click-inside-plot settings* see description of transition state space area (Third column)

   - **Apply to All TSS** button: applies *click-inside-plot settings* to all shown state space plots.
   - **Clear Plot** button: deletes *click-inside-plot settings* of this plot.
   - **Unzoom** button: sets axes of particular plot to the default ranges.
   - set of discrete scale sliders labeled **Discrete Value of XYZ**: one for each variable not shown in particular plot so you could investigate each state of state space. Each slider is in range of all layers each defined by some adjacent pair of thresholds from PMA approximation. If there is no unshown variable, there is also no scale slider.
   - text field: shows all variables on separate line with particular range of values (bounding particular state you are pointing at in the plot).

5.4 Results

This is the last tab in the row. It is used for exploration of parameter space and satisfiable states. You can visualise 2D parameter space plots with satisfying parameterisations or the combination of one parameter and one variable to explore mutual dependencies besides plots of satisfying state space to capture all boundaries of initial conditions meeting particular property. If the model contains just one parameter the showing parameter space plots are

considered to be 1D with only one axis displayed (for which the parameter name is selected) and the second one empty (for which the constant *(none)* is selected) or you can still show 2D plot for the combination of the parameter and one of the variables. Also, if the model contains just one variable the showing satisfying state space plots are considered to be 1D with only one axis displayed (for which the variable name is selected) and the second one empty (for which the constant *(none)* is selected). All of these plots can be downloaded by clicking right mouse button inside them. Moreover, file with the results (in JSON format - see Section 3.4.2) can be loaded separately so you don't have to run parameter synthesis process all over again. So we encourage you to save results once they are ready. Area is visually divided into two horizontal parts.

5.4.1 First Part

It contains controllers for setting of visual output of all plots. It compounds of:

- **Browse...** button: is actually a file loader. It is responsible for loading of prepared or saved results (JSON format). When used, below it appears uploading progress bar and next to it the name of loaded file.
- **Save Results** button: calls file-download routine of your browser with current results (last one loaded or computed by parameter synthesis.
- **Show Parameters Coverage** checkbox: is a switch between two modes of parameter space visualisation. Default mode (when checkbox is untick) shows overlapping satisfying parameterisations in one shade of green colour (so called flat) while other mode (when checkbox is tick) shows overlapping satisfying parameterisations with different shades of green colour so you can tell which part of parameter space covers particular property the best. Problem is that such visualisation is approximated by many rectangles and its accuracy depends on following **Resolution** slider. Be careful with it as it could be computationally expensive.
- **Colour Shade Degree** slider (only for the previous checkbox on): sets the magnitude of shade of overlapping parameterisations.
- **Resolution** slider (only for the previous checkbox on): sets the number of rectangles in the row which defines granularity of resulting parameter

space. The more of these rectangles harder will be the computational process.

5.4.2 Second Part

At first, there is just one button - **Add Plot** — which is disabled until parameter synthesis is done correctly or result file is loaded and the text giving you a hint to run parameter synthesis or to load some results file.

Basically, it is area showing several similar rows of plots with their personal controllers and you can set each row separately but the possibilities of controllers are the same for each row in general. Therefore, we describe just one row. Right after **Add Plot** button is clicked new row appears below the others. Each row contains visually highlighted setting area at the top of the row and below it the visual area. Setting area compunds of:

- **Select Formula** selector: sets property for which to display the results in this row. It contains the list of all investigated properties. First one is selected by default but you can choose whichever you want.
- **Delete** button: deletes particular pair of plots with all of their settings.
- **Hide** checkbox: temporary hides particular pair of plots. They can be shown at your command by unticking this checkbox.

Visual area contains pair of plots (one parameter space with satisfying parameterisations and one state space with highlighted satisfying states) with their personal controllers. Explicitly:

1. First column (controllers for manipulation with parameter space plot). For more information about *click-inside-plot settings* see description of parameter space area (Second column)
    - **Clear Plot** button: deletes *click-inside-plot settings* of this plot.
    - **Unzoom** button: sets axes of particular plot to the default ranges.
    - **Horizontal Axis** selector: sets parameter or variable for horizontal axis of particular parameter space plot. Defaultly, it is filled with first parameter according to the model description in model editor (see Section 5.2.2).
    - **Vertical Axis** selector: sets parameter or variable for vertical axis of particular parameter space plot. Defaultly, it is filled with second parameter according to the model description in model

editor (see Section 5.2.2). If this is not possible (the model has just one parameter) the first variable is selected for this axis according to model description and parameter space plot becomes a parameter-variable combination plot. It is also possible to select the constant *(none)* in which case parameter space plot is considered to be in 1D mode. Note, that it is not allowed in this kind of plot to select for both axis just variables.

- set of checkboxes (only in advanced mode): one for each variable (implicitly turned off) and one for each parameter not shown in particular plot (implicitly turned on). They control whether a parameter or variable has shown scale slider or not. Turning slider off means that whole range of its values is taking into account for showing satisfying parameterisations.
- set of scale sliders labeled **Value of XYZ**: one for each parameter not shown in particular plot so you could investigate each corner of parameter space. Each slider is in the range of that particular parameter according to model description in model editor (see Section 5.2.2). If there is no unshown parameter, there is also no scale slider. In advanced mode, there is present scale slider also for each model variable in the range of its minimal and maximal thresholds after PMA approximation.
- text field: shows all parameters and variables on separate line with particular value used for displaying particular plot. For parameters, this might be one particular value (with slider on) or the range of values (with slider off) and for variables, it is always the range of values bounding one state (with slider on) or the group of states (with slider off). Note, that presence of sliders is depending also on advanced mode.

2. Second column (parameter space area labeled either **Parameter Space** or **Parameter-Variable Combination Plot**)

   contains either a plot of parameter space or a plot of parameter-variable combination. The plot is shown for parameters or combination of one parameter and one variable selected in particular selectors. You can zoom in by clicking left mouse button and moving your mouse until you release the button. For zooming out use **Unzoom** button. By double-clicking inside the plot (without moving your mouse) you select specific point. If this point crosses some satisfying parameterisation, in state space next to this plot there will be highlighted (by blue borders)

all satisfying states for which particular formula holds in at least one selected parameterisation. This point is set as *click-inside-plot settings* of this plot. Note, that some satisfying state can be shown only if selected point crosses some shown satisfying parameterisation. Also note, that in 1D plot selected point is displayed as line from obvious reasons.

3. Third column (satisfying state space area labeled **Satisfying State Space**)

   contains a plot of state space of the model with highlighted (with green background) satisfying states. This plot is shown for variables selected in particular selectors. You can zoom in by clicking left mouse button and moving your mouse until you release the button. For zooming out use **Unzoom** button. By double-clicking inside the plot (without moving your mouse) you select specific state for which only corresponding satisfying parameterisations in parameter space plot will be shown. Such state turns into dark green. It is possible to select more than one state in this way. To deselect just one state repeat double-clicking on it, to deselect all states use **Clear Plot** button. These states are set as *click-inside-plot settings* of this plot. Note, that if no state is selected (by default) satisfying parameterisations for all states are shown in the particular parameter space plot.

4. Fourth column (controllers for manipulation with satisfying state space plot). For more information about *click-inside-plot settings* see description of satisfying state space area (Third column)

   - **Clear Plot** button: deletes *click-inside-plot settings* of this plot.
   - **Unzoom** button: sets axes of particular plot to the default ranges.
   - **Horizontal Axis** selector: sets variable for horizontal axis of this plot only. Defaultly, it is filled with first variable according to the model description in model editor (see Section 5.2.2).
   - **Vertical Axis** selector: sets variable for vertical axis of this plot only. Defaultly, it is filled with second variable according to the model description in model editor (see Section 5.2.2). If this is not possible (model has just one variable) the constant *(none)* is selected for this axis and plots are displayed in 1D mode.
   - set of scale sliders labeled **Value of XYZ**: one for each variable not shown in particular plot so you could investigate each state of

state space. Each slider is in the range of minimal and maximal thresholds of that particular variable after PMA approximation. If there is no unshown variable, there is also no scale slider.

- text field: shows all variables on separate line with particular range of values (bounding particular state you are pointing at in the plot).

Bibliography

[BBD+16a] Nikola Beneš, Luboš Brim, Martin Demko, Samuel Pastva, and David Šafránek. Parallel SMT-based parameter synthesis with application to piecewise multi-affine systems. In *ATVA*, volume 9936 of *LNCS*, pages 1–17. Springer, 2016.

[BBD+16b] Nikola Beneš, Luboš Brim, Martin Demko, Samuel Pastva, and David Šafránek. A model checking approach to discrete bifurcation analysis. In *FM 2016*, volume 9995 of *LNCS*, pages 85–101, 2016.

[Bra14] Tim Bray. The JavaScript object notation (JSON) data interchange format. RFC 7159, RFC Editor, March 2014.

[BST+10] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. The smt-lib standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*, volume 13, page 14, 2010.

[BYWB07] Grégory Batt, Boyan Yordanov, Ron Weiss, and Calin Belta. Robustness analysis and tuning of synthetic gene networks. *Bioinformatics*, 23(18):2415–2422, 2007.

[CGP99] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS'08*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[GBF+11] Radu Grosu, Grégory Batt, Flavio H. Fenton, James Glimm, Colas Le Guernic, Scott A. Smolka, and Ezio Bartocci. From cardiac cells to genetic regulatory networks. In *CAV'11*, volume 6806 of *LNCS*, pages 396–411, 2011.

[KS09] James P Keener and James Sneyd. *Mathematical physiology*, volume 1. Springer, 2009.