GALTOSM handbook

Shounak Saha, Susmoy Das and Arpit Sharma

Table of Contents

Contents

1 Introduction

This handbook serves as a general guide for the GALTOSM tool. It is not specifically intended for artifact evaluation or for reproducing the case studies. For those purposes, please refer to the README files included in the artifact. The primary goal of this handbook is to assist users in understanding and using the new (modified) syntax of the APCTL, APRCTL, and APCTL* logics, as well as in writing syntactically correct grammars for the tool. In future versions, we plan to introduce more flexibility in the syntax requirements.

This guide focuses on the currently supported model checking features and excludes components still in the experimental phase. The directory paths shown throughout are illustrative examples and may not match the exact structure of the artifact. They are provided to demonstrate the expected syntax and usage format.

The core focus of the handbook is to walk users through generating an ADTMC using CADP, writing syntactically correct action logic, and converting both the model and logic using GALTOSM. Other features are still under development and are not covered in detail here.

2 Model Translator Tool

2.1 ADTMC to SDTMC

2.1.1 ADTMC(by CADP) to SDTMC(STORM or PRISM)

- ❑ ✔ We begin with **obtaining the required CADP package**. All the experiments discussed in the paper have been run on the CADP 2024-k of CADP. Here are some key points we wish to emphasize for the user to keep in mind during the installation of CADP:
  - Filling up the details for registration and downloading the INSTALLATOR from the CADP website[1].
  - Follow the steps while running the INSTALLATOR and request a license.
  - Setting up the necessary environment variables. The values we have set for carrying out the experiments are:
    1. `$CADP = /home/lab/cadp`
    2. `$CADP_BITS=64` (since the machine is 64-bit machine)
    3. `$CADP_MEMORY=50000000000`
    4. `$PATH=/bin:/usr/bin:/usr/bin/X11:$CADP/com:$CADP/bin.x64`
  - Ensure the environment variables are set in the working terminal in your Ubuntu system.
  - Run `$CADP/com/tst` to check the installation was complete or not.

  The CADP website provides detailed descriptions regarding them [2].

  ✔ **Modeling the problem in CADP**. CADP used the LNT language to model a problem. The user can look at a sample model for any of the case studies discussed in the paper in:

  `$BASE_DIRECTORY/Artifact_GALTOSM/case_studies/CADP_case_studies`

  NOTE: *If there is any version mismatch of CADP, there is a high possibility that the case studies can not be recreated.* Please refer to the change list of CADP[3].

[1] `https://cadp.inria.fr/registration/`
[2] `https://cadp.inria.fr/man/installator.html`
[3] `https://cadp.inria.fr/changes.html`

✔ **Building the state and transitions space of ADTMC**

Finally, the model specification can then be compiled to generate the state and transition space of the ADTMC using the following steps:

1. Run the command

   `$ lnt.open` *`/home/path/to/the/filename`*`.lnt generator` *`/home/path/to/the/filename`*`.bcg`

   Example usage:

   ```
   $ lnt.open /home/lab/dice/dice.lnt generator /home/lab/dice/dice.bcg
   ```

   This will create a '.bcg' file in the location you mentioned. This '.bcg' file is not a human-readable file.

   Troubleshooting advice: The environment variable may not be in the working terminal of your system.

2. Since '.bcg' is not a human-readable file, next, we have to change it to an '.aut' file, which is a human-readable one. The corresponding command is:

   `$ bcg_io` *`/home/path/to/your/filename`*`.bcg` *`/home/path/to/your/filename`*`.aut`

   Example usage:

   ```
   $ bcg_io /home/lab/dice/dice.bcg /home/lab/dice/dice.aut
   ```

3. Lastly, as per the CADP instructions [**?**], the '.aut' file needs to be modified by replacing all instances of the symbol *!* with the keyword *prob*, and all action names must be suffixed by a semicolon after them. For example, a line from the unmodified '.aut' file would look like `(0, "A !0.3", 1)`. After modification, it will transform to `(0, "A; prob 0.3", 1)`. The 'sed' command can be used to do this for all occurrences in the files as follows:

   `$ sed -i 's/ !/; prob /g'` *`/home/path/to/your/filename`*`.aut`

   Example usage:

   ```
   $ sed -i 's/ !/; prob /g' /home/lab/dice/dice.aut
   ```

The '.aut' file is our desired ADTMC model, which acts as an input to the model translator tool for ADTMC to SDTMC. Details of the conversions are mentioned in the following sub-sections.

❑ ✔ Build the executable**(Artifact contains the executable too)**:

1. Make sure you have `cmake` version 3.1 or above is installed. If not, please follow the instructions at [4].
2. Move the directory containing the code of the logic translator tool `$BASE_DIRECTORY/Artifact_GALTOSM/GALTOSM_tool/source_code/model_translator_tool`
3. Use 'cd' to change directory "adtmc_to_sdtmc-dev"
4. Create a new directory using 'mkdir' with the name "build"
5. Next, 'cd' again to the above "build" directory.
6. Now, we run the following commands:

   ```
   $ cmake ..
   $ cmake --build .
   ```

   An executable named "ADTMC_TO_SDTMC" will be created inside the "build" directory, which is our ADTMC to SDTMC model translator tool.

✔ Transform the ADTMC model to SDTMC model files for STORM and PRISM:

[4] `https://askubuntu.com/questions/610291/how-to-install-cmake-3-2-on-ubuntu`

1. Ensure you are in the same build directory where the "ADTMC_TO_SDTMC" executable was created.
2. (Optional) Run the command

   `$ ./ADTMC_TO_SDTMC -help`

   to get the basic information regarding the general working features of the tool.
3. Next, run the command (for model translation)

   `$ ./ADTMC_TO_SDTMC` *`/home/path/to/aut/file`*`.aut -`*`output_tool`*

   Note:
   - Example ".aut" file has been provided in the Artifact in the location. `$BASE_DIRECTORY/Artifact_GALTOSM/GALTOSM_tool/test_cases/adtmc_to_sdtmc`
   - Replace *`output_tool`* with either `PRISM` or `STORM` based on your preference.
   - The '.aut' file above is the file obtained by following the steps described in **??**.

   Example usage

   `$ ./ADTMC_TO_SDTMC /home/lab/dice/dice.aut -STORM`

   `$ ./ADTMC_TO_SDTMC /home/lab/dice/dice.aut -PRISM`

   On running the above command, two files will be created in `/home/lab/dice` with the names `dice.tra` and `dice.lab`

2.1.2 ADTMC(in mcrl2) to SDTMC(PRISM or STORM)

As the probabilistic processes in mcrl2 is still in experimental extension as in [5], it's highly likely to change over time. Therefore our tool is not highly optimized for this embedding but works for decent-sized models.
The source code for the model has been provided in the location

`$BASE_DIRECTORY/Artifact_GALTOSM/GALTOSM_tool/`
`source_code/model_translator_tool/adtmc-mcrl2_to_sdtmc-prism`

The test cases for the same has been provided in the `test_case` folder in `GALTOSM_tool`

✔ To use the tool follow the following steps.

1. Change directory to the location of the source code given above.
2. RUN THE COMMAND(for model conversion)

   `$ python3 mcrl2-aut_to_prism-pm.py` *`/home/path/to/the/aut/file.aut`*

   Example Usage:

   `$ python3 mcrl2-aut_to_prism-pm.py /home/lab/dice/dice.aut`

OUTPUT: A '.prism' fill will be created in the same location as of the '.aut' location. This file can be used in both PRISM and STORM.

[5] `https://www.mcrl2.org/web/user_manual/tutorial/probability/index.html`

2.2 SDTMC to ADTMC

The state to action model translator tool of GALTOSM is located in
`$BASE_DIRECTORY/Artifact_GALTOSM/GALTOSM_tool/source_code/model_translator_tool/sdtmc_to_adtmc`
This directory contains the following:

✔ PRISM to mcrl2
The PRISM to mcrl2 translator takes three files as input

- .sta
- .lab
- .tra

to produce a '.mcrl' file as output in the same location of the input files.

1. change to the directory containing the source code.

```
$BASE_DIRECTORY/Artifact_GALTOSM/GALTOSM_tool/source_code/model_translator_tool/sdtmc_to_adtmc
```

- Run the command (form model translation)

```
$ python3 sdtmc_to_mcrl2.py <path/to/sta> <path/to/sta> <path/to/sta>
```

Example Usage

```
$ python3 sdtmc_to_mcrl2.py /home/lab/dice/dice.sta /home/lab/dice/dice.lab /home/lab/dice/dice.tra
```

Output: A '.mcrl2' file in the format of mcrl2 source code in the same location as the '.sta' '.lab' '.tra'. This '.mcrl2' can be compiled using `mcrl22lps` and `lps2lts` to get the '.aut' in the mcrl2 format..
Note: The order of the '.sta' '.lab' '.tra' files do not matter.

✔ PRISM to CADP
The PRISM to mcrl2 translator takes three files as input

- .sta
- .lab
- .tra

to produce a '.mcrl' file as output in the same location of the input files.

1. change to the directory containing the source code.

```
$BASE_DIRECTORY/Artifact_GALTOSM/GALTOSM_tool/source_code/model_translator_tool/sdtmc_to_adtmc
```

- Run the command (form model translation)

```
$ python3 sdtmc_to_cadp.py <path/to/sta> <path/to/sta> <path/to/sta>
```

Example Usage

```
$ python3 sdtmc_to_cadp.py /home/lab/dice/dice.sta /home/lab/dice/dice.lab /home/lab/dice/dice.tra
```

Output: A '.aut' file in the format of CADP in the same location as the '.sta' '.lab' '.tra'. This '.aut' can be model checked in CADP.

Note: The order of the '.sta' '.lab' '.tra' files do not matter.

3 Logic Translator Tool

3.1 Specifying properties using a Logic

This section provides a detailed explanation on how to write syntactically correct formulas using the logics discussed in the paper, emphasising on proper bracketing and the modified syntax used for AP(R)CTL and APCTL*. Since these logics share many structural elements, we only explain the APCTL syntax in detail. For the other logics, we only highlight the differences and necessary modifications required for each of them. A set of grammar rules is also provided to help the user generate valid formulas.
Throughout the section, state formulas are represented by ϕ, and ψ has been used to represent either the run or path formula based on the context of the underlying grammar. Run(s) and path(s) have been used interchangeably, representing the same formula. χ represents a valid auxiliary logic formula as defined in [1].

3.1.1 APCTL [2] [1]

✔ Grammar:

ϕ ::= `(true)` | `(false)` | (!ϕ) | (ϕ&ϕ) | (ϕ | ϕ) | ($P\lambda x[\psi]$)
begincenter6pt] ψ ::= `X_`(χ)ϕ | `X_tau`ϕ | ϕ`_`(χ)`U`ϕ | ϕ`_`(χ)`U_`(χ)ϕ
begincenter6pt] λ ::= `=?` | `>=` | `>` | `<=` | `<`
begincenter6pt] χ ::= `true` | `false` | `!`χ | `"a"` | (χ&χ) | (χ | χ)

✔ Note:

- `x`∈[0,1] if λ is not `=?`.
- `a` is an action name, which has to be enclosed within "double quotes"

✔ Details:

1. All state formulas(ϕ) are enclosed within `()`.
   Example `(true)`, `(P=?[path formula])`
2. (!ϕ) is the unary boolean NOT operator followed by the operand which is represented in literature by the symbol ¬. The syntax of the formula is `(!(state formula))`.
   Note: As the whole formula is a state formula, there will be `()` around the whole formula, and after `!` there is a state formula so there will also be `()` too.
   Example `(!(true))`, `(!(P=?[path formula]))`
3. The (ϕ&ϕ) operator is represented by the symbol &, and (ϕ | ϕ) by |. The syntax of the formula is `((state formula)&(state formula))` and `((state formula)|(state formula))` respectively.
   Note: As the whole formula is a state formula, there will be `()` around the whole formula, and on either side of `&`/| there are state formulas so there will also be `()`.
   Example `((true)&(true))`, `((true)|(P=?[path formula]))`
4. The ($P\lambda x[\psi]$) operator is represented by the symbol `P=?[path formula]`, `P>=0.36[path formula]`, or `P>.6[path formula]`, with similar forms for the less-than comparisons.
   Note: As the whole formula is a path formula, there will be `()` around the whole formula. The formula whose probability we want to calculate will be enclosed by `[]`. *Example* `(P=?[X_("action")(true)])`.
5. Run/path formulas are **not** enclosed in `()`.
   Example `X_tau(state formula)`, `(state formula)_(chi)U(state formula)`

6. X_(χ) operator is represented by `X_(chi)(state formula)`.
   Note: There will be `no ()` around the whole formula(rule 5). As this operator works on a state formula, the state formula will be enclosed by `()`. And χ will be enclosed within `()`,
   Example `(P>=[X_("act1" | "act2")(true)])`

7. `X_tau` operator is represented by `X_tau(state formula)`.
   Note: There will be `no ()` around the whole formula(rule 5). As this operator works on a state formula, the state formula will be enclosed by `()`.
   Example `(P>=[X_tau(true)])`

8. ϕ_(χ)`U`ϕ and ϕ_(χ)`U`_(χ)ϕ operator is represented by `U` and the format is `(state formula)_(chi)U(state formula)` and `U_(chi)` with format `(state formula)_(chi)U_(chi)(state formula)`.
   Note: On either side of the operator, there are state formulas, so they will be enclosed in `()` and the chi's are enclosed in `()` *Example* `(P=?[(true)_("ACTION")U(true)])`, `(P=?[(true)_("ACTION1")U_("ACTION2")`

3.1.2 APCTL* [2]

✔ Grammar:

ϕ ::= `(true)` | `(false)` | (!ϕ) | (ϕ&ϕ) | (ϕ | ϕ) | ($P\lambda x[\psi]$)
begincenter6pt] ψ ::= ϕ | (!ψ) | (ψ&ψ) | (ψ | ψ) | (`X`ψ) | (`X_`(χ)ψ) | (ψ`U`ψ)
begincenter6pt] λ ::= `=?` | `>=` | `>` | `<=` | `<`
begincenter6pt] χ ::= `true` | `false` | `!`χ | `"a"` | (χ&χ) | (χ | χ)

✔ Note:

- `x`∈[0,1] if λ is not `=?`.
- `a` is an action name, which has to be enclosed within "double quotes"

✔ Details:
The rules for the state formulas are common since state formula operators are the same as those in APCTL, all operators has been described below

1. (!ϕ), (ϕ&ϕ), (ϕ | ϕ), ($P\lambda x[\psi]$) has been described in 1,2,3,4 respectively in section 3.1.1.
2. The run/path formulas (ψ) are enclosed in `()` too, contrary to rule 5 of APCTL grammar.
   Note: Rules for the state formula remains the same as 1, which will be enclosed by `()`.
   Example `(P=?[(X_tau(true))])`, `(P=?[((true)&(true))])`
3. (!ψ) operator has the same rules as state not formula. The operator representation is `(!(path formula)`.
   Note: The `()` is for the path/run formula where as the `()` is for the state formula.
   Example `(!(true))`
4. (ψ&ψ) and(ψ | ψ) has the same explanation as rule 3 of 3.1.1
   Note: The `()` is for the path/run formula where as the `()` is for the state formula. *Example* `(P=?[((!(true))&(true))])`
5. `X`(ψ) operator explanation is same as `X_tau` and can be found in rule 7 of 3.1.1
   The `()` is for the path/run formula whereas the `()` is for the state formula.
   Example `(P=?[(X(true))])`

6. (`X_`$(\chi)\psi$) operator explanation can be found in rule 6 of 3.1.1
   The `()` is for the path/run formula whereas the `()` is for the state formula.
   Example `(P=?[(X_tau(true))])`

7. (ψ`U`ψ) operator is represented by `U` and the format is `(state formula)U(state formula)`.
   Note: On either side of the operator, there are state formulas, so they will be enclosed in `()`.
   Example `(P=?[(true)U(true)])`.

3.1.3 APRCTL [2]

✔ Grammar:

ϕ ::= `(true)` | `(false)` | (!ϕ) | (ϕ&ϕ) | (ϕ | ϕ) | ($P\lambda x[\psi]$) | ($E\lambda y[\phi]$)
begincenter6pt] ψ ::= `X_`(χ)ϕ | `X_tau`ϕ | ϕ_(χ)`U`ϕ | ϕ_(χ)`U_`(χ)ϕ
begincenter6pt] λ ::= `=?` | `>=` | `>` | `<=` | `<`
begincenter6pt] χ ::= `true` | `false` | `!`χ | `"a"` | (χ&χ) | (χ | χ)

✔ Note:

- `x`∈[0,1] if λ is not `=?`.
- `y` $\in \mathbb{N}$
- `a` is an action name, which has to be enclosed within "double quotes"

✔ Details:
All state and run/path formulas coincide with the APCTL rule, which can be found in 3.1.1 except for the ($E\lambda y[\phi]$), which is explained below.

1. The ($R\lambda y[\phi]$) operator is represented by `E=?[]` or `E>=24[state formula]` or `E>64[state formula]` symbols and same for less than.
   Note: Since E operator has a state formula, `()` as per rule 1 of 3.1.1 inside the [] of E.
   Example `(E=?[(true)])`, `(E=?[(P>=0.5[X_tau(true)])])`

3.1.4 PCTL [2]

✔ Grammar:

ϕ ::= `(true)` | `(false)` | `(ap)` | (!ϕ) | (ϕ&ϕ) | (ϕ | ϕ) | ($P\lambda x[\psi]$)
begincenter6pt] ψ ::= `X`ϕ | ϕ`U`ϕ
begincenter6pt] λ ::= `=?` | `>=` | `>` | `<=` | `<`

✔ Note:

- `x`∈[0,1] if λ is not `=?`.
- `ap` is an atomic proposition, which has to be enclosed within "double quotes"

✔ Details:
The rules are the same as specified in 3.1.1 point 1,2,3,4,5.

1. All state formulas(ϕ) are enclosed within `()`.
   Example `(true)`, `(P=?[path formula])`
2. `(ap)` is an atomic proposition and is a state formula, so put it in `()` *Example* `("DEAD")`
3. The $!\phi$ operator is same as rule 2 of 3.1.1
4. The $(\phi\&\phi)$ operator and $(\phi \mid \phi)$ operator is same as rule 3 of 3.1.1
5. The $(P\lambda x[\psi])$ operator is same as rule 4 of 3.1.1.
   Note: As the whole formula is a path formula, there will be `()` around the whole formula. The formula whose probability we want to calculate will be enclosed by `[]`.
   Example `(P=?[X(true)])`.
6. Run/path formulas are **not** enclosed in `()`.
   Example `X_tau(state formula)`, `(state formula)_(chi)U(state formula)`
7. `X`ϕ operator is represented by `X(state formula)`.
   Note: There will be `no ()` around the whole formula(rule 5). And as this operator works on a state formula, the state formula will be enclosed by `()`.
   Example `(P>=[X(true)])`
8. ϕ`U`ϕ is represented by `(state formula)U(state formula)`.
   Note: There will be no `(` around the whole formula but the state formulas in either sides of the `U` operator will be enclosed by `()`
   Example `(P=?[(true)U(true)])`

3.1.5 PCTL* [2]

✔ Grammar:

ϕ ::= `(true)` | `(false)` | `(ap)` | $(!\phi)$ | $(\phi\&\phi)$ | $(\phi \mid \phi)$ | $(P\lambda x[\psi])$
begincenter6pt] ψ ::= ϕ | $(!\psi)$ | $(\psi\&\psi)$ | $(\psi \mid \psi)$ | (`X`ψ) | (ψ`U`ψ)
begincenter6pt] λ ::= `=?` | `>=` | `>` | `<=` | `<`

✔ Note:

- `x`$\in$[0,1] if λ is not `=?`.
- `ap` is an atomic proposition, which has to be enclosed within "double quotes"

✔ Details:
Since the operators of PCTL* is a subset of APCTL*, the rules are the same as 3.1.2

3.1.6 PRCTL [2]

✔ Grammar:

ϕ ::= `(true)` | `(false)` | `(ap)` | $(!\phi)$ | $(\phi\&\phi)$ | $(\phi \mid \phi)$ | $(P\lambda x[\psi])$ | $(R\lambda y[\phi])$
begincenter6pt] ψ ::= `X`ϕ | ϕ`U`ϕ | $\phi \leq rU\phi$
begincenter6pt] λ ::= `=?` | `>=` | `>` | `<=` | `<`

✔ Note:

- `x`∈[0,1] if λ is not `=?`.
- `r,y` $\in \mathbb{N}$
- `ap` is an atomic proposition, which has to be enclosed within "double quotes"

✔ Details:
All state operators are common to the PCTL state operators in section 3.1.4 except the $(R\lambda y[\phi])$. The path/run formula of PRCTL has one additional operator from PCTL, which is $\phi \leq rU\phi$, they are explained below:

1. The $(R\lambda y[\phi])$ operator is represented by `R=?[]` or `R>=24[state formula]` or `R>64[state formula]` symbols and same for less than.
   Note: Since R operator has a state formula, `()` as per rule 1 of 3.1.1 inside the [] of E.
   Example `(R=?[(P>=1[X(true)])])`
2. $\phi \leq rU\phi$ operator is represented in the format `(state formula)<=rU(state formula)`. The rules of the until operator as defined in rule 8 in section 3.1.4
   Example `(R=?[(P>=0.1[("act1")<=25U("act2")])])`

3.2 Running the Logic translator tool

✔ Choice of the Logic type:
Section 3.1 elaborates the steps to get a syntactically correct logic, `logic`, of a specific type, `logic_type` among:

1. `APCTL` - For APCTL logic, which will get converted to PCTL
2. `APCTLS` - For APCTL* logic, which will get converted to PCTL*
3. `APRCTL` - For APCTL with reward logic, which will get converted to PCTL with rewards
4. `PCTL` - For PCTL logic, which will get converted to APCTL
5. `PCTLS` - For PCTL* logic, which will get converted to APCTL*
6. `PRCTL` - For PCTL with reward logic, which will get converted to APCTL with rewards

✔ Translating the logic specification:

1. Make sure you have Python 3 installed.
2. Navigate to the directory containing the 'logic_translator.py' file.
3. Run the command

   `$ python3 logic_translator.py` *`logic_type`* `"`*`logic`*`"`

   Example usage:

   ```
   $ python3 logic_translator.py APCTLS "(P=?[((X(true))&(X(true)))])"
   ```

   ```
   $ python3 logic_translator.py APCTL "(P=?[X_(\"ONE\"|\"TWO\"|\"THREE\")(true)])"
   ```

   This will give the corresponding logic as output in the terminal.
4. To get the output in a '.props' file, run the command

   `$ python3 logic_translator.py` *`logic_type`* `"`*`logic`*`" > `*`filename`*`.props`

✔ Note:

- Make sure when passing " as input in the formula through the command line, put a \before ".
- If one forgets to enter the `logic_type`, it will prompt an appropriate error message.
- Not giving a correct `logic_type` will result in the prompt of the correct types as output.

References

[1] Das, S., Sharma, A.: On the use of model and logical embeddings for model checking of probabilistic systems. In: Huisman, M., Ravara, A. (eds.) Formal Techniques for Distributed Objects, Components, and Systems. pp. 115–131. Springer Nature Switzerland, Cham (2023)

[2] Das, S., Sharma, A.: Embeddings between state and action based probabilistic logics. Form. Asp. Comput. **37**(2) (Mar 2025). https://doi.org/10.1145/3696431, `https://doi.org/10.1145/3696431`