AdamMC – A User's Guide

A **Flow-LTL Model Checker**
for the Analysis of Distributed
Asynchronous Models

https://uol.de/csd/adammc/

Version 1.0 – October 22, 2020

Contents

Chapter 1

About AdamMC

AdamMC is a model checker providing three different kind of input domains for asynchronous distributed systems: software-defined networks & Flow-LTL, Petri nets with transits & Flow-LTL, and 1-bounded Petri nets & LTL. Internally it reduces the problems to a verification problem of circuits which is checked by ABC, a large toolbox of hardware verification algorithms.

In *software-defined networks (SDNs)* the *data plane* of a network is separated of the *control plane* to allow for a simplified network management. *Flow-LTL* is a linear temporal logic to separately reason with LTL about the control plane and also with LTL about the data plane. AdamMC allows for checking SDNs with *concurrent overlapping updates*, i.e., updates which can be rolled out concurrently during the network's execution without demanding a package to be either routed by the initial or the final configuration but allow for a mixture. In addition to Flow-LTL formulas standard properties like *connectivity*, *loop freedom*, *drop freedom*, and *packet coherence* can be checked.

Internally, Petri nets with transits are used to represent SDNs with updates. *Petri nets with transits* refine the flow relation of standard 1-bounded Petri nets to allow for the modeling of the precise flow of the data. In general Petri nets with transits can be used to distinguish the flow of tokens in a Petri net in cases where the additional complexity introduced by Colored Petri nets is not needed. Further application domains apart from SDNs are for example the flow of work pieces in a *smart factory* or the flow of people in *access control* scenarios.

Internally, this problem is reduced to the checking of 1-bounded Petri nets against LTL. *Petri nets* are one of the most suitable formalism to model asynchronous systems with a high degree of concurrency. AdamMC takes Petri nets in PNML or APT format.

AdamMC is a command line tool, which once started does not allow any further interaction. It does not provide a graphical interface for the input models but provide automatically created outputs using the DOT format of Graphviz. Detailed information about the background can be found here:

Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Model checking data flows in concurrent network updates. In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2019*, 2019a

Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Model checking data flows in concurrent network updates (full version). *arXiv preprint arXiv:1907.11061*, 2019b. URL `https://arxiv.org/abs/1907.11061`

Chapter 2

Setting up `AdamMC`

2.1 Download

The most recent version of `AdamMC` can be found at https://uol.de/csd/adamMC/. At the given address additionally to the binary JAR file the corresponding source code is provided. The license is GNU GPLv3, for details please see the COPYING file provided in the main folder.

2.2 Dependencies

`AdamMC` needs Java in a version $\geq$ 9 and uses the following external tools for dedicated subprocesses:

- Mandatory:
  - MCHyper
  - ABC
  - Aigertools (aigtoaig and aigtodot)
- Optional:
  - DOT in a version $\geq$ 2.36.0. For the visualization of the input and intermediate results.

Note that we adapted MCHyper such that formulas can be read from a file and increased an offset such that larger formulas can be handled. Also the output is slightly adapted. A patch to integrate the changes is provided in the tarball.

Furthermore, `AdamMC` uses the following libraries included in the JAR file:

- APT
- Apache commons-cli-1.2
- Apache commons-collections4-4.0
- Apache commons-io-2.4
- antlr-4.5.1

Finally, `AdamMC` is integrated into the ADAM framework.

2.3 Installation

For unpacking the downloaded tarball into the current directory navigate to the file and type

```
$ tar -xf adam_mc.tar.gz
```

For the external tools please consult the documented installation processes provided by the given websites. For MCHyper please firstly apply the patch `mchyper.patch` located in the `res` folder to the downloaded sources, i.e., navigate to the extracted main folder of MCHyper `mchyper-0.91`

```
$ cd mchyper-0.91
$ patch -p1 < <mainfolder>/res/mchyper.patch
```

Add the corresponding paths to the installed binaries to the `ADAM.properties` file for example like this:

```
aigertools=/home/user/tools/aiger/
mcHyper=/home/user/tools/mchyper-0.91/mchyper
abcBin=/home/user/tools/abc/abc
dot=dot
time=/usr/bin/time
```

Now `AdamMC` is easily started by the given bash script

```
$ ./adam_mc
```

or direct by calling java with

```
$ java -DPROPERTY_FILE=./ADAM.properties -jar adam_mc.jar
```

When so far everything went fine, you should see a list of available modules. For some example calls of `AdamMC` please see Sec. 3.

2.4 Compiling from Source

For compiling the provided sources the *make* script in the main folder can be used. For also using the tests the following jars have to be put into `./test/lib/`:

- javassist-3.21.0.GA
- testng-6.9.9
- scannotation-1.0.2

You can either change the `build.properties` of each module (`client/ui`, `logics`, `modelchecking`, `petrinetWithTransits`, `tools`) or you put the above mentioned libraries (APT, Apache commons-cli-1.2, Apache commons-collections4-4.0, Apache commons-io-2.4, antlr-4.5.1) into the `./lib/` folder. Either you use the naming and folder structure *ant* will tell you while compiling or you adapt the `libs.res` and `testlibs.res` for each module. With

```
$ make
```

a new folder `./deploy` is created containing the binary, a bash script for calling `AdamMC`, and the `ADAM.properties` copied from the main source folder (you have to adapt the path as mentioned in Section 2.3). For deleting all created files you can use

```
$ make clean
```

or to also delete the created files from the test

```
$ make clean-all
```

can be used.

The tests for each module can be executed with the help of *ant*. For each module providing tests you can choose to run all test, a test class, or a test-method. For example for the model checking module you can use

```
$  ant test
```

to run all tests provided (i.e., all classes in `test` annotated with `@Test`). With

```
$ ant test-class -Dclass.name=uniolunisaar.adam.modelchecker.libraries.
    TestingMCHyper
```

the complete class *TestingMCHyper* is run. However,

```
$ ant test-method -Dclass.name=uniolunisaar.adam.modelchecker.circuits.
    TestingModelcheckingFlowLTLParallel -Dmethod.name=updatingNetworkExample
```

tests the specific method *updatingNetworkExample* of the class *TestingMCHyper*.

Chapter 3

Usage of AdamMC

The usage of `AdamMC` should be self-explanatory by the printable helping dialogues. Calling `AdamMC` with

```
$ ./adam_mc
```

lists the available modules like

```
Usage: sh adam.sh <module> or java -jar adam.jar <module>
Available modules:
  pnwt2dot                       Converts a Petri net with transits to a
                                 dot file.
  pnwt2pdf                       Converts a Petri net with transits to a
                                 pdf file by using Graphviz (dot has to be
                                 executable).
  sdn2dot                        Converts a Software Defined Networks
                                 topology (with an concurrent update) to a
                                 Petri net with transits and saves this to
                                 a dot file.
  sdn2pdf                        Converts a Software Defined Networks
                                 topology (with an concurrent update) to a
                                 Petri net with transits and saves this to
                                 a pdf file by using Graphviz (dot has to
                                 be executable).
  mc_pn                          Modelchecking 1-bounded Petri nets with
                                 inhibitor arcs against LTL.
  mc_pnwt                        Modelchecking Petri nets with transits
                                 against FlowLTL or LTL.
  mc_sdn                         Modelchecking Software Defined Networks
                                 with concurrent updates.
  gen_mc_rm_node_update          Generates a network which has an update
                                 function to detour exactly one node (the
                                 node is chosen randomly). Saves the
                                 resulting net in APT and, if dot is
                                 executable, as pdf.
  gen_mc_redundant_flow_network  Generates a network which has two ways to
                                 the output. A update function can block
                                 one of the ways. This can be done in
                                 correct or incorrect ways. Saves the
```

```
                                    resulting net in APT and, if dot is
                                    executable, as pdf.
gen_topologie_zoo                   Generates a network from the topology
                                    given by the input file. Saves the
                                    resulting net in APT and, if dot is
                                    executable, as pdf.
```

Calling a module without any parameter results in a helping dialog printing the available and needed parameters. For a complete list see App. A.

There are different kind of modules. The module `*2dot` and `*2pdf` are used the visualize Petri net with transits or SDNs with the help of Graphviz. The module `mc_pnwt` allows for model checking Petri net with transits against Flow-LTL and module `mc_pn` for standard 1-bounded Petri nets against LTL. The module `mc_sdn` is used for the software-defined network scenarios. The remaining modules are used to generate example Petri nets with transits.

You can find several examples for each of the three different application areas of `AdamMC` in the `examples` folder of the tarball. In the following we list some example calls with the resulting output:

```
$ ./adam_mc mc_pnwt -i ./examples/pnwt-flowltl/detour5.apt -f "A F pOut"
    # UNSAT
$ ./adam_mc mc_pnwt -i ./examples/pnwt-flowltl/detour5.apt -f "(((F G (p2
   AND (pup AND pIn)) IMP G F tup) AND ((F G (pOut AND p3) IMP G F t4)
   AND ((F G (p3 AND p2) IMP G F t3) AND ((F G (p1 AND pIn) IMP G F t0)
   AND (F G (p1 AND p2) IMP G F t2))))) IMP A F pOut)"  # SAT
$ ./adam_mc mc_pnwt -i ./examples/pnwt-flowltl/detour5.apt -f "(((F G (p2
   AND (pup AND pIn)) IMP G F tup) AND ((F G (pOut AND p3) IMP G F t4)
   AND ((F G (p3 AND p2) IMP G F t3) AND ((F G (p1 AND pIn) IMP G F t0)
   AND (F G (p1 AND p2) IMP G F t2))))) IMP A F pOut)" -app seqIn # SAT
## with fairness assumptions in APT file
$ ./adam_mc mc_pnwt -i ./examples/pnwt-flowltl/twoWays33C.apt -f "A F out
   " -app parIn -v # UNSAT
$ ./adam_mc mc_pnwt -i ./examples/pnwt-flowltl/twoWays33C.apt -f "(NEG G
   F (tupD OR tupU) IMP A F out)" -app parIn -cr_abc # SAT
$ ./adam_mc mc_pnwt -i ./examples/pnwt-flowltl/twoWays33C.apt -f "(G F
   createFlows IMP (NEG G F (tupD OR tupU) IMP A F out))" -app parIn -max
    NONE # SAT
$ ./adam_mc mc_pnwt -i ./examples/pnwt-flowltl/factory.apt -f "((m_w AND
   m_i) IMP A ((G NEG F db_w AND F sdb_1) AND F vA_w))" -app parIn -stuck
    GFo # SAT
$ ./adam_mc mc_pnwt -i ./examples/pnwt-flowltl/factory.apt -f "(F m_w AND
   db_w)" -app parIn -max IntC -veri "IC3|BMC2|BMC3" -stats "" # UNSAT
$ ./adam_mc mc_pnwt -i ./examples/pnwt-flowltl/factory.apt -f "A ((F m_w
   AND F db_w) AND F vA_w)" -app parIn -max IntC # SAT
```

Listing 3.1: Example calls for checking Petri nets with transits against Flow-LTL.

```
$ ./adam_mc mc_pn -i ./examples/pn-ltl/AutoFlight-PT-04a/model.pnml -pnml
    -f "NEG (p73 AND NEG p74)" -psst # SAT
$ ./adam_mc mc_pn -i ./examples/pn-ltl/AutoFlight-PT-24a/model.pnml -pnml
    -f "(NEG p453 OR p129)" -v # SAT
$ ./adam_mc mc_pn -i ./examples/pn-ltl/BusinessProcesses-PT-01/model.pnml
    -pnml -f "FALSE" -veri "IC3|BMC2" # UNSAT
```

```
$ ./adam_mc mc_pn -i ./examples/pn-ltl/BusinessProcesses-PT-01/model.pnml
    -pnml -f "G X F G p171" -veri "BMC3" # UNSAT
$ ./adam_mc mc_pn -i ./examples/pn-ltl/DES-PT-02a/model.pnml -pnml -f "F
    F (FALSE U NEG (p35 AND NEG p93))" -veri "IC3" # SAT
$ ./adam_mc mc_pn -i ./examples/pn-ltl/DES-PT-02a/model.pnml -pnml -f "(
    NEG (p66 AND NEG p51) U (NEG (p94 AND NEG p73) U NEG (p47 AND NEG p7))
    )" -max IntF -v # SAT
$ ./adam_mc mc_pn -i ./examples/pn-ltl/Dekker-PT-010/model.pnml -pnml -f
    "F NEG (p34 AND NEG p0_8)"  -stats "" # SAT
$ ./adam_mc mc_pn -i ./examples/pn-ltl/Dekker-PT-010/model.pnml -pnml -f
    "G G G F NEG (p0_9 AND NEG flag_0_4)"  -veri "IC3|BMC2" # UNSAT
$ ./adam_mc mc_pn -i ./examples/pn-ltl/Raft-PT-05/model.pnml -pnml -f "X
    G X F NEG (p62 AND NEG p120)" -veri "IC3|BMC3" # UNSAT
$ ./adam_mc mc_pn -i ./examples/pn-ltl/Raft-PT-05/model.pnml -pnml -f "G
    p43" -veri "BMC2" -max NONE -psst # UNSAT
```

Listing 3.2: Example calls for checking Petri nets against LTL.

```
$ ./adam_mc mc_sdn -i ./examples/sdn/pipelineTopology.txt -u "[[upd(s1.
    fwd(s3/s2)) >> upd(s2.fwd(-/s4))] || upd(s3.fwd(s5))]" -f "A F (s4 OR
    s5)" # SAT
$ ./adam_mc mc_sdn -i ./examples/sdn/pipelineTopology.txt -u "[[upd(s1.
    fwd(s3/s2)) >> upd(s2.fwd(-/s4))] || upd(s3.fwd(s5))]" -c loopFreedom
    -v # SAT
$ ./adam_mc mc_sdn -i ./examples/sdn/pipelineTopology.txt -u "[upd(s1.fwd
    (s4/s2)) >> upd(s2.fwd(-/s4))]" -c connectivity # SAT
$ ./adam_mc mc_sdn -i ./examples/sdn/pipelineTopology.txt -u "[upd(s1.fwd
    (s4/s2)) >> upd(s2.fwd(-/s4))]" -f "A F s2" -t # UNSAT
$ ./adam_mc mc_sdn -i ./examples/sdn/pipelineTopology.txt -u "[upd(s2.fwd
    (s3/s4)) >> upd(s3.fwd(s5))]" -c dropFreedom # SAT
$ ./adam_mc mc_sdn -i ./examples/sdn/campus.txt -u "[[upd(S.fwd(C/A)) >>
    upd(L.fwd(P/C))] || upd(C.fwd(L/P))]" -c dropFreedom # SAT
$ ./adam_mc mc_sdn -i ./examples/sdn/campus.txt -u "[[upd(S.fwd(C/A)) >>
    upd(L.fwd(P/C))] || upd(C.fwd(L/P))]" -c connectivity # SAT
$ ./adam_mc mc_sdn -i ./examples/sdn/campus.txt -u "[[upd(S.fwd(C/A)) >>
    upd(L.fwd(P/C))] || upd(C.fwd(L/P))]" -c loopFreedom # SAT
```

Listing 3.3: Example calls for checking SDNs with updates.

These calls are provided as bash scripts `pnwt-FlowLTLExamples`, `pn-LTLExamples`, `sdnExamples` in the main folder.

To compare your own settings for the optimizations of `AdamMC` on your benchmark families, we provide example scripts in the folder `compare`.

Chapter 4

File Formats

4.1 The Input Formats

4.1.1 Petri Nets

Petri nets can be given to `AdamMC` in the Petri Net Markup Language (PNML) and in the APT format. Since the format of Petri nets with transits in Sec. 4.1.2 extends the APT format, we shortly introduce this format here.

An input file in APT format contains sections for places (*.places*), transitions (*.transitions*) and the connections between them (*.flows*). You have to name the Petri net (*.name "my name"*) and set its type (*.type LPN*). If you like to, you can give a description of the net with *.description "lorem"*. This is the only string which allows line breaks. In general white spaces are ignored and comments are allowed in C syntax. Thus, either whole lines can be commented by //, or an area is commented starting with /* and ending with */. The section *.initial_marking* contains the initial marking of the Petri net. A simplified grammar for Petri nets in APT format is given by:

```
pn          = ( description | flows | initialMarking |
                name | netOptions | places | transitions | type )*
name        = '.name' STR
type        = '.type' ('LPN' | 'PN')
description = '.description' (STR | STR_MULTI)
netOptions  = '.options' (option (',' option)*)?

places      = '.places' place*
place       =  (ID | NAT) (opts)?

transitions = '.transitions' transition*
transition  = (ID | NAT) (opts)?

opts        = '[' option (',' option)* ']'
option      =   ID '=' STR | ID '=' NAT | ID '=' NEGNAT |
```

```
                    ID  '='  DOUBLE     | ID

flows          = '.flows'  flow*
flow           = (ID | NAT)  ':'  set  '->'  set  (opts)?

set            = '{'  ( | obj (',' obj)*)  '}'
obj            = NAT '*'  (ID | NAT)  |  (ID | NAT)

initialMarking = '.initial_marking'  (set)?

NAT            =  ('0'..'9')+
NEGNAT         = '-' ('0'..'9')+
DOUBLE         = '-'? ('0'..'9')+ '.' ('0'..'9')+
ID             = ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*

COMMENT        = ('//' ~('\n'|'\r')*   |   '/*' (. )*? '*/')
WS             = ( ' ' | '\n' | '\r' | '\t')

STR            = '"' ~('"' | '\n' | '\r' | '\t')*  '"'
STR_MULTI      = '"' ~('"' | '\t' )*  '"'
```

Listing 4.1: Simplified grammar of the APT format for labeled Petri nets.

4.1.2 Petri Nets with Transits

The APT format allows to equip places or transitions with additional information. We exploit this generality to obtain a input format for Petri nets with transits. We add for every transition with transit the definition of the transits to the keyword *tfl*. This definition contains a comma separated list of transits. Each transit states the preset place where the flow comes from, or the special character > if a flow is newly started and a set of postset places where the flow is transited to. We additionally allow to mark a transition as weak or as strong fair by the keyword *weakFair* or *strongFair*, respectively. As an example see the following Petri net with transits:

```
.name "twoWays33C"
.type LPN

.places
in
mD
mU
mutex
out[reach="true"]
p0
p1
p2
p3
p4
p5
pupD
pupU
```

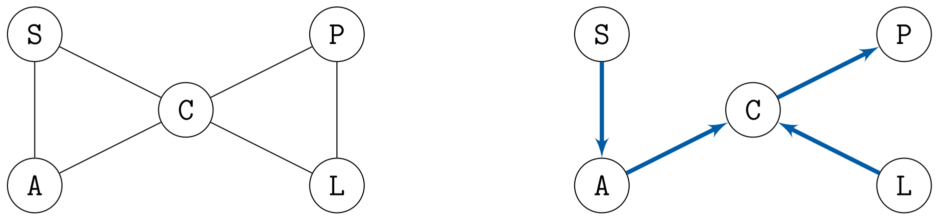
```
.transitions
createFlows[tfl="in -> {in},> -> {in}"]
mtD[tfl="pupD -> {p3}"]
mtU[tfl="pupU -> {p0}"]
resD[strongFair="true", tfl="in -> {in},pupD -> {in}"]
resU[strongFair="true", tfl="in -> {in},pupU -> {in}"]
tD0[strongFair="true", tfl="p3 -> {p3},in -> {p3}"]
tD1[strongFair="true", tfl="p3 -> {p4},p4 -> {p4}"]
tD2[strongFair="true", tfl="p4 -> {p5},p5 -> {p5}"]
tDout[strongFair="true", tfl="p5 -> {out},out -> {out}"]
tU0[strongFair="true", tfl="p0 -> {p0},in -> {p0}"]
tU1[strongFair="true", tfl="p0 -> {p1},p1 -> {p1}"]
tU2[strongFair="true", tfl="p1 -> {p2},p2 -> {p2}"]
tUout[strongFair="true", tfl="p2 -> {out},out -> {out}"]
tupD[tfl="p3 -> {pupD}"]
tupU[tfl="p0 -> {pupU}"]

.flows
createFlows: {1*in} -> {1*in}
mtD: {1*pupD, 1*mD} -> {1*mutex, 1*p3}
mtU: {1*mU, 1*pupU} -> {1*p0, 1*mutex}
resD: {1*pupD, 1*in} -> {1*pupD, 1*in}
resU: {1*pupU, 1*in} -> {1*pupU, 1*in}
tD0: {1*in, 1*p3} -> {1*in, 1*p3}
tD1: {1*p3, 1*p4} -> {1*p3, 1*p4}
tD2: {1*p4, 1*p5} -> {1*p4, 1*p5}
tDout: {1*out, 1*p5} -> {1*out, 1*p5}
tU0: {1*p0, 1*in} -> {1*in, 1*p0}
tU1: {1*p1, 1*p0} -> {1*p0, 1*p1}
tU2: {1*p2, 1*p1} -> {1*p2, 1*p1}
tUout: {1*out, 1*p2} -> {1*out, 1*p2}
tupD: {1*mutex, 1*p3} -> {1*pupD, 1*mD}
tupU: {1*p0, 1*mutex} -> {1*pupU, 1*mU}

.initial_marking {1*in, 1*mutex, 1*out, 1*p0, 1*p1, 1*p2, 1*p3, 1*p4, 1*
    p5}
```

Listing 4.2: An example Petri net with transits in `AdamMC`'s format.

The grammar of the transit relation is given by the following listing:

```
tfl      = (flow (',' flow)*) EOF

flow     = init '->' set
init     = (obj | GR)

set      = '{' ( obj (',' obj)*) '}'
obj      =   ID | INT

INT      = '0'..'9'+
ID       = ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
GR       = '>'

COMMENT = ('//' ~('\n'|'\r')* | '/*' (. )*? '*/')
WS       = ( ' ' | '\n' | '\r' | '\t')
```

Listing 4.3: Grammar of the transit relation of Petri nets with transits.



(a) The connections of the example. (b) The forwardings of the example.

Figure 4.1: Example network topology and initial configuration of a software defined network. The corresponding input file is given in Listing 4.4.

4.1.3 Software Defined Networks and Concurrent Overlapping Updates

For SDNs `AdamMC` provides an input format to describe a topology with an initial configuration. As an example see the input file in Listing 4.4.

```
.name "campus"
.switches
S
A
C
P
L

.connections
S A
S C
A C
C P
C L
P L

.ingress={S}
.egress={P}

// initial config
.forwarding
S.fwd(A)
A.fwd(C)
L.fwd(C)
C.fwd(P)
```

Listing 4.4: Example for a topology and inital configuration of an SDN.

We define a network with five switches S, A, C, P, and L, with an ingress node S and an egress node P. The connections of the topology and the forwardings of the initial configuration can be seen in Fig. 4.1. The following grammar defines the input format.

```
ts          = name description? genOptions? switches cons
              ingress egress forwarding EOF
name        = '.name' STR
description = '.description' (STR | STR_MULTI)
genOptions  = '.options' (option (',' option)*)?

switches    = '.switches' switchT*
switchT     = sw (opts)?

opts        = '[' option (',' option)* ']'
option      = ID '=' STR

cons        = '.connections' con*
con         = sw sw (opts)?

sw          = ID | NAT

ingress     = '.ingress=' set
egress      = '.egress=' set

set         = '{' ( sw (',' sw)*) '}'

forwarding  = '.forwarding' forward*
forward     = sw '.fwd(' sw ')'

NAT         = '0'..'9'+
ID          = ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*

STR         = '"' ~('"' | '\n' | '\r' | '\t')* '"'
STR_MULTI   = '"' ~('"' | '\t' )* '"'
COMMENT     = ('//' ~('\n'|'\r')* | '/*' (. )*? '*/')
WS          = ( ' ' | '\n' | '\r' | '\t')
```

Listing 4.5: Grammar of the topology and the initial configuration of SDNs.

Furthermore, `AdamMC` provides a parser for the concurrent overlapping updates. For examples please see the updates in the example calls of Listing 3.3.

```
result      = update EOF
update      = swUpdate | seqUpdate | parUpdate

swUpdate    = 'upd(' idi '.fwd(' sidi ('/' idi)? '))'
seqUpdate   = '[' update (SEQ update)* ']'
parUpdate   = '[' update (PAR update)* ']'

sidi        = idi | '-'
idi         = ID | INT

PAR         = '||'
SEQ         = '>>'

INT         = '0'..'9'+
ID          = ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*

COMMENT     = ('//' ~('\n'|'\r')* | '/*' (. )*? '*/')
WS          = ( ' ' | '\n' | '\r' | '\t')
```

Listing 4.6: Grammar of the concurrent overlapping updates of SDNs.

4.1.4 LTL and Flow-LTL

AdamMC also provides a parser for the temporal logics LTL and Flow-LTL. For examples please see the example calls of AdamMC of Listing 3.2 and Listing 3.1. The syntax is given by the following grammar.

```
flowLTL         = runFormula EOF

runFormula      = ltl | '(' ltl rimp runFormula ')' | runBinary |
    flowFormula
runBinary       = '(' runFormula rbin runFormula ')'
flowFormula     = forallFlows ltl

ltl             = ltlUnary | ltlBinary | tt | ff | atom
ltlUnary        = unaryOp ltl
ltlBinary       = '(' ltl binaryOp ltl ')'

atom            = ID | INT

// LTL
unaryOp         = (ltlFinally | globally | next | neg)
binaryOp        = (and | or | imp | bimp | until | weak | release )
ltlFinally      = 'F'
globally        = 'G'
next            = 'X'
neg             = 'NEG' | '!'
and             = 'AND'
or              = 'OR'
imp             = 'IMP' | '->'
bimp            = 'BIMP' | '<->'
until           = 'U'
weak            = 'W'
release         = 'R'

// FlowFormula
forallFlows     = 'A'

// RunFormula
rbin            = rand | ror
rand            = 'AND'
ror             = 'OR'
rimp            = 'IMP' | '->'

tt              = 'TRUE'
ff              = 'FALSE'

INT             = '0'..'9'+
ID              = ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_'
    )*

COMMENT         = ('//' ~('\n'|'\r')* | '/*' (. )*? '*/')
WS              = ( ' ' | '\n' | '\r' | '\t')
```

Listing 4.7: Grammar of the temporal logics Flow-LTL

Chapter 5

Contact

We appreciate your feedback on `AdamMC`. Please send any bugs, comments, or questions to:

manuel.gieseking(at)informatik.uni-oldenburg.de

Appendix A

Detailed List of Modules and Parameters

In the following we list the help dialogues of each module. That is, how to call the module, the possible and needed parameters including their explanations.

Module: pnwt2dot

Converts a Petri net with transits to a dot file. The help dialogue:

```
usage: sh adam.sh pnwt2dot [-d] -i <file> [-l <file>] [-o <file>] [-psst]
       [-v]
Converts a Petri net with transits to a dot file.
 -d,--debug            Get some debug infos.
 -i,--input <file>     The path to the input file.
 -l,--logger <file>    The path to an optional logger file. If it's not
                       set, the information will be send to the terminal.
 -o,--output <file>    The path to the output folder. If it's not given the
                       path from the input file is used.
 -psst,--silent        Makes the tool voiceless.
 -v,--verbose          Makes the tool chatty.
```

Module: pnwt2pdf

Converts a Petri net with transits to a pdf file by using Graphviz (dot has to be executable). The help dialogue:

```
usage: sh adam.sh pnwt2pdf [-d] -i <file> [-l <file>] [-o <file>] [-psst]
       [-v]
Converts a Petri net with transits to a pdf file by using Graphviz (dot
has to be executable).
 -d,--debug            Get some debug infos.
 -i,--input <file>     The path to the input file.
 -l,--logger <file>    The path to an optional logger file. If it's not
```

```
                          set, the information will be send to the terminal.
 -o,--output <file>       The path to the output folder. If it's not given the
                          path from the input file is used.
 -psst,--silent           Makes the tool voiceless.
 -v,--verbose             Makes the tool chatty.
```

Module: sdn2dot

Converts a Software Defined Networks topology (with an concurrent update) to a Petri net with transits and saves this to a dot file. The help dialogue:

```
usage: sh adam.sh sdn2dot [-d] -i <file> [-l <file>] [-o <file>] [-optCon]
       [-psst] [-u <cup>] [-v]
Converts a Software Defined Networks topology (with an concurrent update)
to a Petri net with transits and saves this to a dot file.
 -d,--debug                    Get some debug infos.
 -i,--input <file>             The path to the input file.
 -l,--logger <file>            The path to an optional logger file. If it's
                               not set, the information will be send to the
                               terminal.
 -o,--output <file>            The path to the output folder. If it's not
                               given the path from the input file is used.
 -optCon,--opt-connections     If set only the necessary connections of the
                               topology are added. This means only those
                               used in the initial configuration and those
                               of the update.
 -psst,--silent                Makes the tool voiceless.
 -u,--update <cup>             A concurrent update.
 -v,--verbose                  Makes the tool chatty.
```

Module: sdn2pdf

Converts a Software Defined Networks topology (with an concurrent update) to a Petri net with transits and saves this to a pdf file by using Graphviz (dot has to be executable). The help dialogue:

```
usage: sh adam.sh sdn2pdf [-d] -i <file> [-l <file>] [-o <file>] [-optCon]
       [-psst] [-u <cup>] [-v]
Converts a Software Defined Networks topology (with an concurrent update)
to a Petri net with transits and saves this to a pdf file by using
Graphviz (dot has to be executable).
 -d,--debug                    Get some debug infos.
 -i,--input <file>             The path to the input file.
 -l,--logger <file>            The path to an optional logger file. If it's
                               not set, the information will be send to the
                               terminal.
 -o,--output <file>            The path to the output folder. If it's not
                               given the path from the input file is used.
 -optCon,--opt-connections     If set only the necessary connections of the
                               topology are added. This means only those
                               used in the initial configuration and those
                               of the update.
 -psst,--silent                Makes the tool voiceless.
```

```
-u,--update <cup>             A concurrent update.
-v,--verbose                  Makes the tool chatty.
```

Module: mc_pn

Modelchecking 1-bounded Petri nets with inhibitor arcs against LTL. The help dialogue:

```
usage: sh adam.sh mc_pn [-circ] [-cp] [-cr_abc] [-cr_com <arg>] [-cr_sys
       <arg>] [-d] [-enc <arg>] -f <LTL> -i <file> [-l <file>] [-max
       <arg>] [-noF] [-o <file>] [-p <abcParameters>] [-pnml] [-pre
       <process>] [-psst] [-stats] [-v] [-veri <verifier>]
Modelchecking 1-bounded Petri nets with inhibitor arcs against LTL.
 -circ,--circuit                    Saves the created circuit of the net
                                    as PDF. Attention: this could be
                                    really huge and dot could need lots
                                    of time!
 -cp,--check-precon                 Checks preconditions like 1-bounded.
                                    Takes some time, but should be used
                                    if you are not sure that your net
                                    fulfills  all necessary
                                    preconditions!
 -cr_abc,--red_abc                  Uses abc dfraig to reduce the
                                    circuit.
 -cr_com,--red_gates_com <arg>      Reduces the number of gates of the
                                    whole cirucit. That means it reduces
                                    the output of McHyper. Possible
                                    values: RX-G | RX-G-S | DS-G |
                                    DS-G-S | DS-G-S-EXTRA | NONE.
                                    Standard: NONE.
 -cr_sys,--red_gates_sys <arg>      Reduces the number of gates of the
                                    system's circuit. Possible values: G
                                    | G-EQCOM | G-I | G-I-EQCOM |
                                    G-I-EXTRA | G-I-EXTRA-EQCOM | NONE.
                                    Standard: NONE.
 -d,--debug                         Get some debug infos.
 -enc,--encoding <arg>              Encoding of the transitions in the
                                    circuit. Possible values: logEnc |
                                    expEnc. Standard: logEnc.
 -f,--formula <LTL>                 The formula which should be checked.
 -i,--input <file>                  The path to the input file.
 -l,--logger <file>                 The path to an optional logger file.
                                    If it's not set, the information
                                    will be send to the terminal.
 -max,--maximality <arg>            States which kind of maximality
                                    should be used. Possible values:
                                    IntC (interleaving calculated within
                                    the circuit)  | IntF (interleaving
                                    added to the formula)  | ConF
                                    (concurrent added to the formula)  |
                                    NONE. Standard: IntC.
 -noF,--noFile                      Does not write the formula to a file
                                    for giving it to MCHyper. This
                                    causes problems for huge formulas.
 -o,--output <file>                 The path to the output folder. If
                                    it's not given the path from the
```

```
                                       input file is used.
 -p,--abcParameters <abcParameters>    Parameters for the verifier /
                                       falsifier for abc. Standard: no
                                       parameters.
 -pnml                                 Allows to read the Petri net from
                                       the PNML format rather than the
                                       standard format.
 -pre,--preProc <process>              Allows to excute any pre-process of
                                       abc before the actual veri- or
                                       falsifier is started.
 -psst,--silent                        Makes the tool voiceless.
 -stats,--statistics                   Calculates and prints some
                                       statistics for the call.
 -v,--verbose                          Makes the tool chatty.
 -veri,--verifier <verifier>           The set of ver- and falsifieres
                                       which should be executed in
                                       parallel.Note that even parallel
                                       execution has some overhead. Input
                                       format: v_1 | ... | v_n with v_i
                                       from {IC3, INT, BMC, BMC2, BMC3}.
                                       Standard: IC3
```

Module: mc_pnwt

Modelchecking Petri nets with transits against FlowLTL or LTL. The help dialogue:

```
usage: sh adam.sh mc_pnwt [-app <arg>] [-circ] [-cp] [-cr_abc] [-cr_com
       <arg>] [-cr_sys <arg>] [-d] [-enc <arg>] [-f <LTL | Flow-LTL
       formula>] -i <file> [-l <file>] [-max <arg>] [-noF] [-o <file>] [-p
       <abcParameters>] [-pre <process>] [-psst] [-st <arg>] [-stats] [-t]
       [-v] [-veri <verifier>]
Modelchecking Petri nets with transits against FlowLTL  or LTL.
 -app,--approach <arg>                    Chosing the sequential or
                                          parallel approach (with or
                                          without inhibitor arcs). Possible
                                          values: seq | seqIn | par |
                                          parIn. Standard: parIn.
 -circ,--circuit                          Saves the created circuit of the
                                          net as PDF. Attention: this could
                                          be really huge and dot could need
                                          lots of time!
 -cp,--check-precon                       Checks preconditions like
                                          1-bounded. Takes some time, but
                                          should be used if you are not
                                          sure that your net fulfills  all
                                          necessary preconditions!
 -cr_abc,--red_abc                        Uses abc dfraig to reduce the
                                          circuit.
 -cr_com,--red_gates_com <arg>            Reduces the number of gates of
                                          the whole cirucit. That means it
                                          reduces the output of McHyper.
                                          Possible values: RX-G | RX-G-S |
                                          DS-G | DS-G-S | DS-G-S-EXTRA |
                                          NONE. Standard: NONE.
 -cr_sys,--red_gates_sys <arg>            Reduces the number of gates of
                                          the system's circuit. Possible
```

```
                                         values: G | G-EQCOM | G-I |
                                         G-I-EQCOM | G-I-EXTRA |
                                         G-I-EXTRA-EQCOM | NONE. Standard:
                                         NONE.
-d,--debug                               Get some debug infos.
-enc,--encoding <arg>                    Encoding of the transitions in
                                         the circuit. Possible values:
                                         logEnc | expEnc. Standard:
                                         logEnc.
-f,--formula <LTL | Flow-LTL formula>    The formula, either Flow-LTL or
                                         LTL, which should be checked.
-i,--input <file>                        The path to the input file.
-l,--logger <file>                       The path to an optional logger
                                         file. If it's not set, the
                                         information will be send to the
                                         terminal.
-max,--maximality <arg>                  States which kind of maximality
                                         should be used. Possible values:
                                         IntC (interleaving calculated
                                         within the circuit) | IntF
                                         (interleaving added to the
                                         formula) | ConF (concurrent
                                         added to the formula) | NONE.
                                         Standard: IntC.
-noF,--noFile                            Does not write the formula to a
                                         file for giving it to MCHyper.
                                         This causes problems for huge
                                         formulas.
-o,--output <file>                       The path to the output folder. If
                                         it's not given the path from the
                                         input file is used.
-p,--abcParameters <abcParameters>       Parameters for the verifier /
                                         falsifier for abc. Standard: no
                                         parameters.
-pre,--preProc <process>                 Allows to excute any pre-process
                                         of abc before the actual veri- or
                                         falsifier is started.
-psst,--silent                           Makes the tool voiceless.
-st,--stuck <arg>                        Different formulas for the
                                         sequential approach to prevent
                                         runs from stucking in a subnet.
                                         Possible values: GFo | GFANDNpi |
                                         ANDGFNpi | GFoANDNpi. Standard:
                                         GFANDNpi.
-stats,--statistics                      Calculates and prints some
                                         statistics for the call.
-t,--trans                               Saves the transformed net in APT
                                         format and, in the case that dot
                                         is executable, as PDF.
-v,--verbose                             Makes the tool chatty.
-veri,--verifier <verifier>              The set of ver- and falsifieres
                                         which should be executed in
                                         parallel.Note that even parallel
                                         execution has some overhead.
                                         Input format: v_1 | ... | v_n
                                         with v_i from {IC3, INT, BMC,
                                         BMC2, BMC3}. Standard: IC3
```

Module: mc_sdn

Modelchecking Software Defined Networks with concurrent updates. The help dialogue:

```
usage: sh adam.sh mc_sdn [-app <arg>] -c <property> | -f <LTL | Flow-LTL
       formula> [-circ] [-cr_abc] [-cr_com <arg>] [-cr_sys <arg>] [-d]
       [-enc <arg>]  -i <file> [-l <file>] [-max <arg>] [-noF] [-o <file>]
       [-p <abcParameters>] [-pre <process>] [-psst] [-st <arg>] [-stats]
       [-t] -u <update> [-v] [-veri <verifier>]
Modelchecking Software Defined Networks with concurrent updates.
 -app,--approach <arg>                    Chosing the sequential or
                                          parallel approach (with or
                                          without inhibitor arcs). Possible
                                          values: seq | seqIn | par |
                                          parIn. Standard: parIn.
 -c,--check <property>                    The standard property to check.
                                          Possible values: connectivity |
                                          loopFreedom | weakLoopFreedom |
                                          dropFreedom | packetCoherence
 -circ,--circuit                          Saves the created circuit of the
                                          net as PDF. Attention: this could
                                          be really huge and dot could need
                                          lots of time!
 -cr_abc,--red_abc                        Uses abc dfraig to reduce the
                                          circuit.
 -cr_com,--red_gates_com <arg>            Reduces the number of gates of
                                          the whole cirucit. That means it
                                          reduces the output of McHyper.
                                          Possible values: RX-G | RX-G-S |
                                          DS-G | DS-G-S | DS-G-S-EXTRA |
                                          NONE. Standard: NONE.
 -cr_sys,--red_gates_sys <arg>            Reduces the number of gates of
                                          the system's circuit. Possible
                                          values: G | G-EQCOM | G-I |
                                          G-I-EQCOM | G-I-EXTRA |
                                          G-I-EXTRA-EQCOM | NONE. Standard:
                                          NONE.
 -d,--debug                               Get some debug infos.
 -enc,--encoding <arg>                    Encoding of the transitions in
                                          the circuit. Possible values:
                                          logEnc | expEnc. Standard:
                                          logEnc.
 -f,--formula <LTL | Flow-LTL formula>    The formula, either Flow-LTL or
                                          LTL, which should be checked.
 -i,--input <file>                        The path to the input topology
                                          file.
 -l,--logger <file>                       The path to an optional logger
                                          file. If it's not set, the
                                          information will be send to the
                                          terminal.
 -max,--maximality <arg>                  States which kind of maximality
                                          should be used. Possible values:
                                          IntC (interleaving calculated
                                          within the circuit) | IntF
                                          (interleaving added to the
                                          formula) | ConF (concurrent
                                          added to the formula) | NONE.
```

```
                                        Standard: IntC.
-noF,--noFile                           Does not write the formula to a
                                        file for giving it to MCHyper.
                                        This causes problems for huge
                                        formulas.
-o,--output <file>                      The path to the output folder. If
                                        it's not given the path from the
                                        input file is used.
-p,--abcParameters <abcParameters>      Parameters for the verifier /
                                        falsifier for abc. Standard: no
                                        parameters.
-pre,--preProc <process>                Allows to excute any pre-process
                                        of abc before the actual veri- or
                                        falsifier is started.
-psst,--silent                          Makes the tool voiceless.
-st,--stuck <arg>                       Different formulas for the
                                        sequential approach to prevent
                                        runs from stucking in a subnet.
                                        Possible values: GFo | GFANDNpi |
                                        ANDGFNpi | GFoANDNpi. Standard:
                                        GFANDNpi.
-stats,--statistics                     Calculates and prints some
                                        statistics for the call.
-t,--trans                              Saves the transformed net in APT
                                        format and, in the case that dot
                                        is executable, as PDF.
-u,--update <update>                    The update of the topoloy which
                                        should be checked.
-v,--verbose                            Makes the tool chatty.
-veri,--verifier <verifier>             The set of ver- and falsifieres
                                        which should be executed in
                                        parallel.Note that even parallel
                                        execution has some overhead.
                                        Input format: v_1 | ... | v_n
                                        with v_i from {IC3, INT, BMC,
                                        BMC2, BMC3}. Standard: IC3
```

Module: gen_mc_rm_node_update

Generates a network which has an update function to detour exactly one node (the node is chosen randomly). Saves the resulting net in APT and, if dot is executable, as pdf. The help dialogue:

```
usage: sh adam.sh gen_mc_rm_node_update [-con] [-d] [-l <file>] -nb1
       <numberOf_nodes> [-npdf] -o <file> [-psst] [-v]
Generates a network which has an update function to detour exactly one
node (the node is chosen randomly). Saves the resulting net in APT and, if
dot is executable, as pdf.
 -con,--connectivity                 Adds the formula checking connectivity
                                     to the net inscription.
 -d,--debug                          Get some debug infos.
 -l,--logger <file>                  The path to an optional logger file.
                                     If it's not set, the information will
                                     be send to the terminal.
 -nb1,--nb_nodes <numberOf_nodes>    The desired number of node (>= 3).
 -npdf,--noPDF                       Does not create a pdf of the generated
```

```
                                          net.
 -o,--output <file>                       The output path where the generated
                                          Petri net with flows should be saved.
 -psst,--silent                           Makes the tool voiceless.
 -v,--verbose                             Makes the tool chatty.
```

Module: gen_mc_redundant_flow_network

Generates a network which has two ways to the output. A update function can block one of the ways. This can be done in correct or incorrect ways. Saves the resulting net in APT and, if dot is executable, as pdf. The help dialogue:

```
usage: sh adam.sh gen_mc_redundant_flow_network [-con] [-d] [-l <file>]
       -nb1 <numberOf_nodesU> -nb2 <numberOf_nodesD> [-npdf] -nv <version>
       -o <file> [-psst] [-v]
Generates a network which has two ways to the output. A update function
can block one of the ways. This can be done in correct or incorrect ways.
Saves the resulting net in APT and, if dot is executable, as pdf.
 -con,--connectivity                        Adds the formula checking
                                            connectivity to the net inscription.
 -d,--debug                                 Get some debug infos.
 -l,--logger <file>                         The path to an optional logger file.
                                            If it's not set, the information
                                            will be send to the terminal.
 -nb1,--nb_nodesU <numberOf_nodesU>         The desired number of node for the
                                            upper path (>= 1).
 -nb2,--nb_nodesD <numberOf_nodesD>         The desired number of node for the
                                            lower path (>= 1).
 -npdf,--noPDF                              Does not create a pdf of the
                                            generated net.
 -nv,--version <version>                    The desired version of the network
                                            (B - basic, U - update, M - mutex, C
                                            - correct).
 -o,--output <file>                         The output path where the generated
                                            Petri net with flows should be
                                            saved.
 -psst,--silent                             Makes the tool voiceless.
 -v,--verbose                               Makes the tool chatty.
```

Module: gen_topologie_zoo

Generates a network from the topology given by the input file. Saves the resulting net in APT and, if dot is executable, as pdf. The help dialogue:

```
usage: sh adam.sh gen_topologie_zoo [-d] -i <path> [-l <file>] [-npdf] -o
       <file> [-psst] [-v]
Generates a network from the topology given by the input file. Saves the
resulting net in APT and, if dot is executable, as pdf.
 -d,--debug              Get some debug infos.
 -i,--input <path>       The input file for the topology from which the Petri
                         game should be created.
 -l,--logger <file>      The path to an optional logger file. If it's not
                         set, the information will be send to the terminal.
```

```
-npdf,--noPDF         Does not create a pdf of the generated net.
-o,--output <file>    The output path where the generated Petri net with
                      flows should be saved.
-psst,--silent        Makes the tool voiceless.
-v,--verbose          Makes the tool chatty.
```