ADAM: The Input File Format

Manuel Gieseking

Carl von Ossietzky Universität Oldenburg
manuel.gieseking(at)informatik.uni-oldenburg.de

This document describes the file format of ADAM, a tool for the automatic synthesis of distributed systems with multiple concurrent processes. For further information about the tool feel free to visit `http://www.uni-oldenburg.de/csd/adam`.

ADAM uses the data structure for Petri nets and the parser provided by APT[1]. Therefore, we rehash some information about the grammar of the parser and the special case how to define Petri games in the APT file format.

In general a file in the APT file format is structured in sections labeled with *.section* with its content added below or directly behind this keyword. For the usability of writing the document, whitespaces are ignored.

The name of the file or Petri game is given by

```
.name "name"
```

Every character is allowed to be used within the string, apart from the doubled quotation marks, line breaks and tabulators.

Since the file format itself is more general and also used for transition systems in other contexts, we have to specify the type of the graph. This is done by

```
.type LPN
```

Optionally we can add a description to the given net with

```
.description "Some describing text of the Petri game"
```

This string is the only one which – different to the string mentioned above – can additionally use line breaks for defining longer descriptions.

The places are given by

```
.places
env1[env="true"]
bad0[token=2, bad="true"]
bad1[token=3, bad="true"]
...
env0[env="true"]
```

For typing a place as an environment place annotate him with [`env="true"`], all other places are typed as belonging to the system players. Do the same for bad places with [`bad="true"`].

[1] `https://github.com/CvO-Theory/apt`

The places of the Petri game aren't just divided into environment and system places, but also into groups of places stating these are the places a given token can lie on. Thus, you can annotate the places with the [`token=<number>`] key-value pair, where <number>is the number of the group this place belongs to. You only have to annotate the system places. Environment places are automatically marked as group 0. Thus, the numbers annotated must start from 1 and it is not allowed to omit some natural number between 1 and the maximum number annotated. Be aware that the partition has to be disjunct in the sense that no two places in one group can be marked at the same time. ADAM has a feature to support you by annotating the places or even completely annotate the places on its own. To use this, just don't annotate any places with [`token=<number>`] and use, for example, the *win_strat* module. If ADAM can annotate the places on its own, you can read the annotation by the colouring of the places in the resulting pdf file of the Petri game. Otherwise, ADAM prints some invariants which should support you by following the token through the net.

The identifiers of the places (like `env1`, `bad0`, etc.) are allowed to be integers or strings beginning with a letter and containing only characters (a-z, A-Z) and integers. Different to the grammar described in Listing 1.2, there are no underscores valid within the identifier of the places for Petri games. This would cause some trouble while calculating the strategy for the Petri game.

For defining the transitions of the Petri game the same rules for the identifiers, apart from the underscores, are implemented. That is, additionally the identifiers can start with an underscore and can contain some within.

```
.transitions
t1
t2
...
```

To define the connection between places and transitions the section *.flows* is used. This is done by listing the pre- and postsets of every transition separately.

```
.flows
t1: {s1} -> {s1}
```

Thus, here the place `s1` has a self loop with transition `t1`. The preset is intuitively defined in front of the arrow and the postset behind.

At the moment the Petri games in ADAM are restricted to safe Petri nets. Thus, there is no need for multi-weight edges. But since this restriction should not last for ever, we are already describing the not 1-bound case here.

Multi-weight edges are defined by a multiplier given by a natural number stating how many tokens should be used or transmitted respectively.

```
.flows
t1: {} -> {s2, 2*s1, s2, 0*s3, 3*s2}
```

The edge weight 1 need not to be given as a multiplier additionally. The multiplier 0 – an edge with weight zero – is the same as not listing the place at

all. The empty pre- or postset can be defined by an empty set {}. Furthermore, the place `s2` is listed three times in the postset of `t1`. Twice with the invisible multiplier 1 and once with the multiplier 3. This yields in an edge with weight 5 in the resulting Petri net. Remark that the multiplier is not commutative, since the identifiers of the places can also – as mentioned above – be only integers and thus the semantics would not be unique any more.

The initial marking of the Petri net is given analogously to the pre- and postsets as multisets with multipliers:

```
.initial_marking {2*s1, s2}
```

Thus, initially there are two token on place `s1` and one on `s2`. Is this section omitted, the resulting Petri net won't have any token.

The last property is the ability to add comments at any place in the file. Therefore we use the comments notations of languages like C. Thus, a comment influencing the whole line is initialized by //. A whole area of a comment can be framed by /* */, which is also possible to contain multiple lines.

```
// comment
.flows
t1: {} -> {2*s1, /*s2, */ 3*s3}
```

All in all, a complete example of a Petri game defined in the APT format can look like:

```
// The environment decides between A or B, and the
    system players should imitate this by choosing A' or
     B'. http://www.react.uni-saarland.de/publications/
    FO14.pdf
.name "SameDecision"
.type LPN
.description "Testing the same decision."

.places
Env[env="true"]
A[env="true"]
B[env="true"]
Sys[token=1]
A_[token=1]
B_[token=1]
EA[env="true"]
EB[env="true"]
qbad[token=1, bad="true"]

.transitions
t1 t2
test1 test2
t1_ t2_
```

```
tbad1 tbad2 tbad3 tbad4

.flows
t1: {Env} -> {A}
t2: {Env} -> {B}
test1: {A,Sys} -> {Sys,EA}
test2: {B,Sys} -> {Sys,EB}
t1_: {Sys} -> {A_}
t2_: {Sys} -> {B_}
// bad states
tbad1: {A_,EB} -> {EB,qbad}
tbad2: {B_,EA} -> {EA,qbad}
tbad3: {A_,B} -> {B,qbad}
tbad4: {B_,A} -> {A,qbad}

.initial_marking {Env,Sys}
```

Listing 1.1. Example of a Petri game

The whole grammar which the parser accepts is given by

```
S                = { name | type | description | places
                   | transitions | flows
                   | initial_marking | final_markings },
                   ? EOF ?;
name             = '.name', STR;
type             = '.type', ('LPN' | 'PN');
description      = '.description', (STR | STR_MULTI);
places           = '.places', place;
place            =  | idi, [opts], place;
opts             = '[', option, ']';
option           = ID, '=', STR, [',', option];
transitions      = '.transitions', transition;
transition       =  | idi, [opts], transition;
flows            = '.flows', flow;
flow             = | idi, ':', set, '->', set, flow;
set              = '{', [objs], '}';
objs             = obj, [',', objs];
obj              = (INT, '*', idi) | idi;
initial_marking  = '.initial_marking', [set];
final_markings   = '.final_markings', final_marking;
final_marking    = | set, final_marking;
idi              = ID | INT;
INT              = DIGIT, {DIGIT};
ID               = (CHAR | '_'), {CHAR | DIGIT | '_'};
DIGIT            = '0' | '1' | '2' | '3' | '4' | '5'
                 | '6' | '7' | '8' | '9';
```

```
CHAR          = 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
                | 'G' | 'H' | 'I' | 'J' | 'K' | 'L'
                | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R'
                | 'S' | 'T' | 'U' | 'V' | 'W' | 'X'
                | 'Y' | 'Z' |
                  'a' | 'b' | 'c' | 'd' | 'e' | 'f'
                | 'g' | 'h' | 'i' | 'j' | 'k' | 'l'
                | 'm' | 'n' | 'o' | 'p' | 'q' | 'r'
                | 's' | 't' | 'u' | 'v' | 'w' | 'x'
                | 'y' | 'z';
STR           = '"', {ALLCHAR - ('"' | '\n' | '\r' | '
    \t')}, '"';
STR_MULTI     = '"', {ALLCHAR - ('"' | '\t')}, '"';
COMMENT       = '//', {ALLCHAR - ('\n'|'\r')}, ['\r'],
     '\n';
              | '/*', {ALLCHAR} - '*/', '*/';
WHITESPACE    = ' ' | '\n' | '\r' | '\t';
ALLCHAR       = ? All possible symbols ?;
```

Listing 1.2. Grammar of the APT format for labelled Petri nets