NAME

aut, AUT – simple file format for labelled transition systems

DESCRIPTION

The AUT format (where *AUT* stands for *AUTomaton*), also called the ALDEBARAN format, is a simple file format for representing labelled transition systems.

The AUT format has been present in the CADP toolbox since its origins (see the HISTORY section below for details about the chronology of the AUT format).

Because of its simplicity and because it is human readable, this format has enjoyed some popularity in academia among developers of verification and testing tools.

However, the AUT format suffers from three drawbacks:

- Because it is a text format, it is not compact enough to efficiently store very large transitions systems (both in terms of disk space and access time).
- Because the AUT format definition was kept as simple as possible, the various implementations would often diverge, for instance, regarding the maximal length for label strings, the places where white spaces are allowed, etc.
- There was a lot of redundant work across these implementations, as each developer would have to implement the same functionalities, e.g., to store labels and transitions in main memory.

To address these issues (and others), the CADP toolbox has been, since 1993, equipped with an alternative file format named BCG (see the **bcg**(LOCAL) manual page for details). Progressively, the BCG format has supplanted the AUT format in CADP, and most of the CADP tools nowadays that can read or write the AUT format do it by internal translation from/to BCG.

Thus, it is advised to use the BCG format rather than the AUT format when developing new tools. The only exception to this general principle concerns those tools developed using a programming language that can not bridge with the BCG code libraries provided by the CADP toolbox.

SPECIFICATION OF THE AUT FORMAT (2014 VERSION)

Labelled transition systems encoded in the AUT format are stored in files having the **.aut** suffix.

In an AUT file, each state is represented by a natural (i.e., unsigned) number.

Blanks (i.e., white spaces, tabulations, etc.) may appear everywhere between lexical tokens (i.e., keywords, numbers, strings, commas, parentheses, etc.), and also at the beginning or end of each line. Blanks are not significant and can be discarded, except inside quoted labels.

Line breaks are significant. Each line should be terminated with a line-feed (LF) character, with the possible exception of the last line of the file. To ensure compatibility with Windows, each line-feed character may be preceded (or not) by a carriage-return (CR) character.

The numbers of states and transitions should be at least 32-bit wide, which enables 4 billions of states and transitions. On 64-bit machines, wider numbers can be used, but the size of AUT files is likely to become prohibitive for large models (in such case, the BCG format is the option of choice).

In the AUT format, only 7-bit ASCII characters are accepted. This is currently found to be sufficient for the study of concurrent systems.

The first line of an AUT file has the following syntax:
des (<initial-state>, <number-of-transitions>, <number-of-states>)
The "**des**" keywords stands for "descriptor". The **<number-of-states>** must be greater or equal to 1, because there is always at least one state, the initial state. State numbers range between 0 and **<number-of-states> - 1**. The **<initial-state>** is always equal to 0.

The remainder of the AUT file consists of one line per transition, in an arbitrary order. Each remaining line has the following syntax:
(<from-state>, <label>, <to-state>)
where **<from-state>** and **<to-state>** are state numbers, and where <label> can take two possible forms:

- **<unquoted-label>** consists of a character string starting with a letter, and followed by zero, one, or many letters, digits, or underscore characters. To ensure backward compatibility with the version of AUT format used by the **tgv**(LOCAL) tool, the character string consisting of a single '∗' character is also accepted as a valid unquoted label.

- **<quoted-label>** consists of a character string starting with a double quote and ending with a double quote. Between these two double quotes, there can be zero, one or many printable characters; the meaning of "printable" is given by the POSIX isprint() function with locale "C" (namely, ASCII characters with decimal codes in the range from 32 to 126, bounds included). No other assumption should be made about the characters present between these two double quotes. In particular, the double quote character itself may be present, and may not necessarily be "escaped" in some way (e.g., preceded by a backslash, as in C).

Any conformant implementation of the AUT format should at least handle labels that are 5000-character long; handling labels of unbounded length is obviously better.

An unquoted label (e.g., **PUT_1**) can also be written as a quoted label (e.g., **"PUT_1"**) without changing its meaning: both labels should be considered to be identical. This rule also applies to the '∗' label.

The non-observable label (also called invisible label, or hidden label, and usually noted "tau" in concurrency theory) is noted **i** in the AUT format, following a convention set by the ISO International Standard 8807 "LOTOS". This same label can also be noted **"i"**, but the unquoted notation is better for compactness. Notice that there is no way to express an observable label that would be noted **i**; this is not a problem for languages such as LOTOS and LNT, in which **i** is a reserved keyword or a reserved gate identifier.

By default, labels are case sensitive, meaning that two labels such as **"put"** and **"PUT"** should be considered to be different.

If each label can semantically be seen as a tuple consisting of a communication port (or gate, channel, etc.) followed by zero, one, or many typed values, then it is recommended (although not mandatory) to write labels according to the syntax conventions defined in the LABEL PARSING section of the **bcg_write**(LOCAL) manual page, and to enable label parsing when processing the AUT file using the CADP tools.

When label parsing is enabled, two labels may be considered to be the same even if they are not character-wise identical. For instance, label **"PUT!false!'\x23'"** will be turned into **"PUT !FALSE !'#'"** by label parsing, and both labels will be considered to be the same.

EXAMPLE OF AN AUT FILE

This is an example of a valid AUT file:

```
des (0, 5, 4)
(0, PUT_6, 1)
(0, "GET !true !7 !CONS (A, CONS (B, NIL))", 2)
(1, i, 3)
(2, "SEND !"hello" !"world"", 3)
(3, "DUMP ... " ... \" ...", 0)
```

The two latter transitions illustrate that no particular assumption should be made about occurrences of double quotes in labels: neither must they be systematically preceded by a backslash, nor must they occur an even number of times, etc.

HOW TO CREATE AN AUT FILE

At present, there are two CADP tools that directly create AUT files: **caesar**(LOCAL) when invoked with its **-aldebaran** option, and **bcg_io**(LOCAL).

The easiest way to produce an AUT file is to first create a BCG file using the **bcg_write**(LOCAL) application programming interface, and then invoke the **bcg_io**(LOCAL) tool to convert this BCG file into an AUT file.

Otherwise, it is easy to produce AUT files directly by writing lines to a file. The only issue may be to produce the descriptor line if the total numbers of states and/or transitions are not already known at the moment the AUT file is created. To solve this issue, **caesar**(LOCAL) writes a first line of the following form:

```
des (0, ?, ?)_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

where the _'s denote a sufficient number of white spaces, and then proceeds normally writing the subsequent lines corresponding to the transitions. When all transitions have been written, the numbers of states and transitions should be known. One can then skip back to the beginning of the file, e.g., using the ftell() POSIX primitive, and overwrite the first line with the actual numbers.

When an AUT file is generated directly, its correctness can be checked by converting it to a BCG file, then back to an AUT file using the **bcg_io**(LOCAL) tool. If the former AUT file is correct, the conversion should succeed. If label parsing is disabled during the conversion, both AUT files should be character-wise identical (with the possible exception of blanks that might have been inserted or removed outside labels). If label parsing is enabled, the conversion will additionally parse the contents of each label, check its correctness, and possibly turn some values into their normal form.

HOW TO READ AN AUT FILE

At present, there is one single CADP tool, **bcg_io**(LOCAL), that directly reads AUT files. This is a design decision to stop maintaining multiple parsers. **bcg_io**(LOCAL) is the reference parser for the AUT format. It has command-line options to enable or disable label parsing.

The easiest way to read an AUT file is to first convert it to BCG using the **bcg_io**(LOCAL) tool, and then opening the resulting BCG file using the **bcg_read**(LOCAL) application programming interface, which provides primitives and data structures for loading and accessing the labelled transition system in memory.

Otherwise, it is always possible to write a new parser for the AUT format. The simplest approach is to write it manually. There is yet an involved point: when reading an AUT file, quoted labels should not be parsed from left to right, as it is not possible, upon occurrence of a double quote character, to immediately decide whether this character marks the end of the label, or if it is just a double quote present in the middle of the label (technically, this is a shift-reduce conflict in the grammar).

The recommended way of parsing quoted labels is to read the entire transition line in a buffer and search for the leftmost and rightmost double quotes, e.g., using the POSIX functions strchr() and strrchr(). The quoted label is the sub-string of characters located between these two double quotes.

As an alternative to manual writing, a parser generator may be used, but one should keep in mind that the AUT format cannot be simply parsed from left to right because double quotes are allowed inside labels. Using a parser generator will only work if the generated parser can perform lookahead and take decisions based upon the right-hand context to resolve shift-reduce conflicts. If so, the double quote that terminates a label can be characterized as any double quote that is followed by no other double quote before the end of line.

HISTORY OF THE AUT FORMAT

An early version of the AUT format was defined in 1987, as part of the VENUS tool [Sor87] for the verification of communicating systems. Because this tool was written in Prolog, the file format for describing automata was naturally a set of Prolog clauses. For instance, the transition relation of a given automaton *A* was described as a set of clauses of the form: *A* (*state1*, *label*, *state2*).

The actual AUT format was defined in 1988, as part of the ALDEBARAN tool [Fer88] for the verification of communicating processes. The Prolog-clause style of the VENUS format was kept (for instance, on the first line), sometimes with simplifications (for instance, the automaton name *A* was dropped from each transition clause).

In 1989, Hubert Garavel extended the syntax with labels between double quotes, so as to provide support for the labelled transition systems generated from value-passing LOTOS specifications.

The AUT format remained unchanged until September 2014, when Hubert Garavel introduced the explicit possibility of having double quotes in labels, so as to support concurrent modelling languages with rich data types, in particular the LNT and PseuCo languages that can pass character string values across communication channels. The specification of the AUT format was revised and made more precise.

BIBLIOGRAPHY

[Sor87] Amelia Soriano Montes. Venus: un outil d'aide a la verification des systemes communicants. These de doctorat, Institut National Polytechnique de Grenoble, Jan. 1987.

[Fer88] Jean-Claude Fernandez. Aldebaran : un systeme de verification par reduction de processus communicants. These de doctorat, Universite Joseph Fourier (Grenoble I), May 1988.

SEE ALSO

aldebaran(LOCAL), **bcg**(LOCAL), **bcg_io**(LOCAL), **bcg_write**(LOCAL).

Additional information is available from the CADP Web page located at http://cadp.inria.fr

Directives for installation are given in files **$CADP/INSTALLATION_***.

Recent changes and improvements to this software are reported and commented in file **$CADP/HISTORY**.

BUGS

Please report bugs to cadp@inria.fr