NAME

caesar.bdd – structural and behavioural analysis of Nested-Unit Petri Nets

SYNOPSIS

caesar.bdd *option* [*filename***.nupn**]

DESCRIPTION

Taking as input *filename***.nupn** (or by default the standard input), which contains a Nested-Unit Petri Net (NUPN) encoded in the **nupn**(LOCAL) format, **caesar.bdd** performs various structural or behavioural analyses, depending on the *option* specified on the command-line, and writes the corresponding results to the standard output. Error and warning messages, if any, are displayed on the standard error.

See the **nupn**(LOCAL) manual page for a detailed definition of the NUPN file format.

For performance reasons, most options of **caesar.bdd** assume that the contents of *filename***.nupn** are correct. When dealing with an unknown Nested-Unit Petri Net, it is thus advisable to first analyze its contents with **-check** before using any other option. Indeed, these other options often perform less stringent verifications than **-check**, or even no verification at all. In particular, options **-dead-places**, **-dead-transitions**, **-concurrent-places**, and **-concurrent-units** rely on an optimized encoding of markings, which is only correct if markings are safe or unit-safe, and may may produce invalid results if this is not the case.

Also, most options of **caesar.bdd** do not take into account the various pragmas that may be present in *filename***.nupn**. As a general rule, the pragmas "**`!multiple_arcs`**", "**`!multiple_initial_tokens`**", and "**`!unit_safe`**" (see the section on pragmas in the **nupn**(LOCAL) format definition) are ignored, unless explicitly stated otherwise below.

OPTIONS

-check Perform syntactic, structural and behavioural analysis of the Nested-Unit Petri Net to verify whether the stated constraints of the NUPN format are satisfied. The **-check** option performs additional checks and warns about redundant or void units. In addition to these static checks, the **-check** option also explores the marking graph using BDDs (Binary Decision Diagrams), prints the number of states reached at each iteration, and checks whether all reachable markings are one safe and unit safe; such checks are inconclusive if the marking graph is too large for being generated. The warning and error messages emitted by this option may differ depending on whether the "**`!unit_safe`**" pragma is present or absent.

-arcs Display the number of arcs in the Nested-Unit Petri Net.

-bits Display the number of bits needed to encode (using the best encoding) the markings of the Nested-Unit Petri Net.

-concurrent-places

Explore the set of reachable markings to determine the pairs of "concurrent" places, i.e., pairs of places Pi and Pj such that Pi = Pj or there exists a reachable marking containing both Pi and Pj. The exploration of the set of reachable markings may be incomplete if a timeout (see below) is specified. The output is defined as follows: Let P be the number of places in the Nested-Unit Petri Net; **caesar.bdd** writes to the standard output a list of P∗(P+1)/2 character values corresponding to the lower half of a matrix indexed according to increasing place numbers; this list of characters is displayed as a sequence of lines of increasing length; the upper half of the matrix is not displayed because it enjoys a symmetry property (see below).

In a first approximation, the character written at the intersection of row i and column j is either "`1`" if places Pi and pj are concurrent, "`0`" if these places are not concurrent, or "`.`" if the answer is unknown.
Actually, beyond these three values, there are six more possible values: "`=`", "`<`", and "`>`", which are equivalent to "`0`" (i.e., non-concurrent), and "`~`", "`[`", and "`]`", which are equivalent to "`.`" (i.e., unknown).
The meaning of these nine values is defined as follows. The NUPN is said to be "presumably unit safe" if either the exploration was complete and the NUPN was found to be unit safe, or the exploration was incomplete and the "`!unit_safe`" pragma is present. Line i corresponds to place Pi (belonging to unit Ui) and row j corresponds to place Pj (belonging to unit Uj). The character written at the intersection of row i and column j (with i != j, i.e., for each non-diagonal element) is equal to:
- "`=`": if the NUPN is presumably unit safe and both places belong to the same unit, i.e., Ui = Uj;
- "`<`": if the NUPN is presumably unit safe and the unit of Pi is contained in the unit of Pj, i.e., Sub* (Ui, Uj), where the Sub* predicate is specified in the **nupn**(LOCAL) format definition;
- "`>`": if the NUPN is presumably unit safe and the unit of Pi contains the unit of Pj, i.e., Sub* (Uj, Ui);
- "`1`": if none of the above applies, and a marking containing both Pi and Pj was reached;
- "`~`": if the NUPN is not presumably unit safe and both places are different and belong to the same unit, i.e., (Pi != Pj) and (Ui = Uj);
- "`[`": if the NUPN is not presumably unit safe and the unit of Pi is contained in the unit of Pj, i.e., Sub* (Ui, Uj);
- "`]`": if the NUPN is not presumably unit safe and the unit of Pi contains the unit of Pj, i.e., Sub* (Uj, Ui);
- "`0`": if none of the above applies, and the exploration was complete and no marking containing both Pi and Pj was reached;
- "`.`": if none of the above applies, and the exploration was incomplete and no marking containing both Pi and Pj was reached.
Concerning diagonal elements, the character written at the intersection of row i and column i is equal to:
- "`1`": if a marking containing Pi was reached (i.e., Pi is not a dead place);
- "`0`": if the exploration was complete and no marking containing Pi was reached (i.e., Pi is a dead place);
- "`.`": if the exploration was incomplete and no marking containing Pi was reached.
The lower half and upper half of the matrix are symmetric modulo permutations of "`<`" and "`>`", and of "`[`" and "`]`". Each line of the lower half of the matrix is compressed using the algorithm described below in the section entitled "COMPRESSION ALGORITHM".

-concurrent-units

Explore the set of reachable markings and determine the pairs of "concurrent" units, i.e., pairs of units Ui and Uj that satisfy the predicate Disjoint (Ui, Uj) specified in the **nupn**(LOCAL) format definition, and such that there exists at least one reachable marking containing one place of Ui and one place of Uj.

The output is defined as follows: Let U be the number of units in the Nested-Unit Petri Net; **caesar.bdd** writes to the standard output a list of U*(U+1)/2 character values corresponding to the lower half of a matrix indexed according to increasing unit numbers; this list of characters is displayed as a sequence of lines of increasing length; the upper half of the matrix is not displayed because it is symmetric; the character written at the intersection of row i and column j is either "`1`" if units Ui and Uj are concurrent, "`0`" if these units are not concurrent, or "`.`" if the answer is unknown because the exploration of reachable markings has been interrupted. Each line of the lower half of the matrix is compressed using the algorithm described below in the section entitled "COMPRESSION ALGORITHM".

-creator
Display the name of the creator tool specified by the **`!creator`** pragma if it is present in the Nested-Unit Petri Net, or the empty string if this pragma is absent. This pragma is specified in the "PRAGMA CREATOR" section of the **nupn**(LOCAL) format definition.

-dead-places
Explore (parts of) the set of reachable markings to determine the set of "dead" places, i.e., places that do not belong to any reachable marking. The output is defined as follows: Let P be the number of places in the Nested-Unit Petri Net; **caesar.bdd** writes to the standard output a line of P character values corresponding to each place and ordered according to increasing place numbers; the character written for each place is either "`1`" if the place is dead, "`0`" if the place is not dead, or "`.`" if the answer is unknown because the exploration of reachable markings has been interrupted. The **`!unit_safe`** pragma, if present, is used to speed up calculations. The output line is compressed using the algorithm described below in the section entitled "COMPRESSION ALGORITHM".

-dead-transitions
Explore (parts of) the set of reachable markings to determine the set of "dead" transitions, i.e., transitions that are not enabled in any reachable marking. The output is defined as follows: Let T be the number of transitions in the Nested-Unit Petri Net; **caesar.bdd** writes to the standard output a line of T character values corresponding to each transition and ordered according to increasing transition numbers; the character written for each transition is either "`1`" if the transition is dead, "`0`" if the transition is not dead (i.e., quasi-live), or "`.`" if the answer is unknown because the exploration of reachable markings has been interrupted. The **`!unit_safe`** pragma, if present, is used to speed up calculations. The output line is compressed using the algorithm described below in the section entitled "COMPRESSION ALGORITHM".

-density
Display the density of the incidence matrice of the the Nested-Unit Petri Net, i.e., the number of arcs divided by twice the product of the number of places by the number of transitions. The density is equal to zero if there are no transitions (hence, no arcs).

-encodings
Display statistics about the number of bits required to represent the markings of the Nested-Unit Petri Net using various possible encodings, namely those encodings described in Section 6 of [Gar19].

-height Display the height of the unit tree of the Nested-Unit Petri Net. All leaf units have height one, and the root unit only increases the height if this unit is not void (i.e., has at least one local place).

-hwb Display the HWB code of the Nested-Unit Petri Net. This code has the form *height-width-bits*, where the three fields are those numbers computed by options **-height**, **-width**, and **-bits**, respectively. Fields *height* and *width* are omitted if *width* is equal to the number of places, i.e., if the NUPN is trivial.

-idle-units
Display the list of idle units (i.e., non-void units that have no local place in the initial marking, and such that all transitions having an output place in such a unit also have an input place in this unit) or the empty string if there are no such units in the Nested-Unit Petri Net. Idle units are thus a subset of dead units (i.e., non-void units having no local place in any reachable marking).

-initial-places
Display the list of places present in the initial marking.

-initial-tokens
Display the number of tokens present in the initial marking. If present, the **`!multiple_initial_tokens`** pragma is taken into account.

-initial-units
Display the list of initial units (i.e., units that contain at least one place present in the initial marking). This list is never empty. The same unit can be displayed multiple times if it contains several places in the initial marking, meaning that the Nested-Unit Petri Net is not unit-safe.

-leaf-units
Display list of leaf units (i.e., units that have no sub-unit) in the Nested-Unit Petri Net.

-max-concurrency
Display (an over-approximation of) the upper bound for the maximal number of tokens in any reachable marking. This option is expected to quickly compute a result, even if the Nested-Unit Petri Net is large; consequently, no exhaustive exploration of reachable markings is done. This option assumes that the Nested-Unit Petri Net is safe, otherwise the result displayed is undefined. The **`!unit_safe`** pragma, if present, is used to speed up calculations.

-max-place
Display the highest place number in the Nested-Unit Petri Net.

-max-transition
Display the highest transition number in the Nested-Unit Petri Net; if there is no transition, this number is equal to zero.

-max-unit
Display the highest unit number in the Nested-Unit Petri Net.

-mcc Undocumented option used to prepare or complete the model forms of the Model Checking Contest. This option computes various structural and behavioural properties of the Nested-Unit Petri Net. The **`!unit_safe`** pragma, if present, is taken into account. If the **`!multiple_arcs`** pragma or the **`!multiple_initial_tokens`** pragma is present, the **-mcc** option computes structural and behavioural properties for the original (non safe) Petri Net rather than for the NUPN itself.

-min-concurrency
Display (an under-approximation of) the lower bound for the maximal number of tokens in any reachable marking. This option is expected to quickly compute a result, even if the Nested-Unit Petri Net is large; consequently, no exhaustive exploration of reachable markings is done. This option assumes that the Nested-Unit Petri Net is safe, otherwise the result displayed is undefined. The **`!unit_safe`** pragma, if present, is used to speed up calculations.

-min-place
Display the lowest place number in the Nested-Unit Petri Net.

-min-transition
Display the lowest transition number in the Nested-Unit Petri Net; if there is no transition, this number is equal to one.

-min-unit
Display the lowest unit number in the Nested-Unit Petri Net.

-multiple-arcs
Display the arguments of the **`!multiple_arcs`** pragma if it is present, or the empty string if this pragma is absent.

-multiple-initial-tokens
Display the arguments of the **`!multiple_initial_tokens`** pragma if it is present, or the empty string if this pragma is absent.

-permanent-units
Display the list of permanent units (i.e., units that have a local place in the initial marking, and such that all transitions having an input place in such a unit also have an output place in this unit) or the empty string if there are no such units in the Nested-Unit Petri Net. Permanent units are thus a subset of initial units.

-places Display the number of places in the Nested-Unit Petri Net.

-pnml Write to the standard output the translation in PNML (Petri Net Markup Language) the network given as input in the NUPN format. The PNML output will contain a "NUPN-toolspecific" section as defined on the http://mcc.lip6.fr/nupn.php page. If the "**`!unit_safe`**" pragma is present, it is propagated to this section. If the input network contains errors, the translation may stop prematurely, leaving an incomplete PNML file. It is therefore recommended to invoke the tool as follows:
caesar.bdd -pnml [*filename***.nupn**] > *filename***.pnml** **||** rm -f *filename***.pnml**

-redundant-units
Display the list of redundant units (i.e., units that have a single sub-unit) or the empty string if there are no such units in the Nested-Unit Petri Net.

-root-unit
Display the number of the root unit of the Nested-Unit Petri Net.

-transitions
Display the number of transitions in the Nested-Unit Petri Net.

-trivial Display "1" if the Nested-Unit Petri Net is trivial, or "0" otherwise.

-units Display the number of units in the Nested-Unit Petri Net.

-version
Display the current version number of the software.

-void-nonroot-units
Display the list of void units (i.e., units having no local place) that are different from the root unit or the empty string if there are no such units in the Nested-Unit Petri Net. If the net is correct, the empty string should be displayed.

-void-root-unit
Display the number of the root unit if this unit is void (i.e., has no local place) or the empty string otherwise.

-void-units
Display the list of void units (i.e., units having no local place, possibly including the root unit) or the empty string if there are no such units in the Nested-Unit Petri Net.

-width Display the width of the unit tree of the Nested-Unit Petri Net. This width is equal to the number of leaf units.

INTERRUPTS

caesar.bdd handles certain POSIX interrupt signals (namely, SIGINT, SIGQUIT, SIGALRM, and SIGTERM) and adapt its behaviour in consequence. Upon reception of such a signal:

- The execution of options **-check**, **-concurrent-places**, **-concurrent-units**, **-dead-places**, **-dead-transitions**, and **-mcc** stops the BDD-based exploration of reachable markings, and completes using pessimistic assumptions (i.e., the fact that parts of the marking graph have not been explored) delivering less precise, yet correct results.
- The execution of other options terminates immediately, returning the exit code 5 (see below).

USER-SPECIFIED TIMEOUTS

The environment variable **`$CAESAR_BDD_TIMEOUT`** can be used to specify a maximal duration for marking graph exploration. If this variable is set to a strictly positive integer *N*, the exploration of the marking graph will stop after *N* seconds. If this variable is set to zero, the exploration of the marking graph is disabled, meaning only the initial marking is visited. If this variable is not set (or set to an invalid value, such as "" or -1), the exploration of the marking graph is performed without timeout. Upon expiration of a timeout:

- The execution of option **-check** terminates immediately, returning the exit code 5 (see below).
- The execution of options **-concurrent-places**, **-concurrent-units**, **-dead-places**, and **-dead-transitions** stops the BDD-based exploration of reachable markings, and completes using pessimistic assumptions.
- The execution of option **-mcc** completes by computing behavioural properties using faster algorithms that rely on pessimistic assumptions.

The functionality provided by this environment variable is not available on Windows.

COMPRESSION ALGORITHM

The output of the four options **-concurrent-places**, **-concurrent-units**, **-dead-places**, and **-dead-transitions** is compressed using a simple algorithm based on run-length encoding: when a character is followed by an integer *n* greater than 3 enclosed between parentheses, it means that this character must be repeated *n* times, including its initial occurrence. For instance, the following sequence of characters:

```
0111000010110000001 1.0000000110011111011111110.10........001111
```

is compressed as:

```
01110(4)10110(6)11.0(7)11001(5)01(7)0.10.(8)001(4)
```

The compression algorithm is implemented as follows in the C language:

```
#include <stdio.h>
int main ()
{
   char C, PREVIOUS = ’\0’;
   int  N, REPEAT = 0;
   while (1) {
      C = getchar ();
      if (C == PREVIOUS) {
         /* assert (C != ’\0’) && (C != EOF) && (C != ’\n’) */
         ++ REPEAT;
      } else {
         /* flush the repetition buffer, if any */
         if (REPEAT > 3) {
            printf ("(%d)", REPEAT);
         } else if (REPEAT > 0) {
            for (N = 1; N < REPEAT; ++ N)
               putchar (PREVIOUS);
         }
         if (C == EOF)
            return 0;
         putchar (C);
         if (C == ’0) {
            PREVIOUS = ’\0’;
            REPEAT = 0;
         } else {
            PREVIOUS = C;
            REPEAT = 1;
         }
      }
   }
}
```

This algorithm enjoys three nice properties: (1) it can operate on the fly (e.g., using coroutines, pipes, or data streams), meaning that it is not mandatory to generate the input entirely before starting to compress it; (2) the size (in characters) of the compressed output is always less or equal to the size of the input; (3) compressing an already compressed input has no effect.

DECOMPRESSION ALGORITHM

The compressed output of the four options **-concurrent-places**, **-concurrent-units**, **-dead-places**, and **-dead-transitions** is decompressed using a simple algorithm, implemented as follows in the C language:

```
#include <stdio.h>
int main ()
{
   char C, PREVIOUS = ’\0’;
   int  REPEAT = 0;
   while (1) {
      if (REPEAT > 0) {
         /* assert (PREVIOUS != ’\0’) && (PREVIOUS != ’\n’) */
         putchar (PREVIOUS);
         -- REPEAT;
```

```
            } else {
               C = getchar ();
               if (C == EOF)
                  return 0;
               if (C != '(') {
                  putchar (C);
                  PREVIOUS = C;
               } else {
                  scanf ("%d)", &REPEAT);
                  /* assert REPEAT > 3 */
                  -- REPEAT;
               }
            }
         }
      }
```

This algorithm can operate on the fly; for instance, one can compare two (or more) compressed files without having to decompress them entirely in advance.

EXIT STATUS

The exit status of **caesar.bdd** may take the following values:

0:	normal termination (everything is alright)
1:	memory shortage (Petri Net and/or BDDs are too large)
2:	incorrect command-line arguments for **caesar.bdd**
3:	*filename***.nupn** does not exist or is unreadable
4:	syntax or semantic error in *filename***.nupn**
5:	**caesar.bdd** terminated by the user (SIGINT or SIGQUIT) or upon timeout (SIGALRM or SIGTERM)
6:	the net given in *filename***.nupn** is not safe or not unit safe

Any other value corresponds to an unexpected error.

AUTHORS

Damien Bergamini and Hubert Garavel (INRIA Rhone-Alpes)

FILES

*filename***.nupn**	Nested-Unit Petri Net (input)

CREDITS

To perform its behavioural analyses, **caesar.bdd** uses the Binary Decision Diagram package CUDD (Release 3.0) developed by Fabio Somenzi at the University of Colorado (Boulder, CO, USA).

BIBLIOGRAPHY

[Gar19] Hubert Garavel. "Nested-Unit Petri Nets". Journal of Logical and Algebraic Methods in Programming, vol. 104, pages 60-85, April 2019. Available from http://cadp.inria.fr/publications/Garavel-19.html

[Gar15] Hubert Garavel. "Nested-Unit Petri Nets: A Structural Means to Increase Efficiency and Scalability of Verification on Elementary Nets". In R. Devillers and A. Valmari, editors, Proceedings of the 36th International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS'15), Brussels, Belgium. Lecture Notes in Computer Science, vol. 9115, Springer, 2015. Superseded by [Gar19]. Available from http://cadp.inria.fr/publications/Garavel-15-a.html

SEE ALSO

caesar(LOCAL), **nupn**(LOCAL), **nupn_info**(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 new bugs to Hubert.Garavel@inria.fr