NAME

nupn, NUPN – Nested-Unit Petri Nets

DESCRIPTION

Nested-Unit Petri Nets (NUPN, for short) are a theoretical model of concurrency. Technically, they extend ordinary, one-safe Petri Nets with structure and locality, by grouping sets of places into hierarchically nested "units". This semantic model is particularly suitable when translating specifications written in high-level concurrent languages (such as process calculi), as structural information can be preserved to make later analyses more efficient and scalable. For further details, refer to the [Gar19] paper listed below in the BIBLIOGRAPHY section.

In practice, the theoretical NUPN model is implemented in two different ways:

- An NUPN-specific extension of the standard PNML format has been proposed in December 2015. This extension is documented by the http://mcc.lip6.fr/nupn.php page and is supported by many model checkers, among which ITS-Tools (Paris), LTSmin (Twente), and PNMC (Toulouse). Models encoded using this extension of PNML are stored in files having the **.pnml** suffix.

- Alternately, a simple file format for representing NUPN models has been present in the CADP toolbox since July 2004 with gradual evoluations. This format was designed to be readable by humans, as well as easy to generate and parse using tools. Models encoded using this format are stored in files having the **.nupn** suffix.

The remainder of this manual page defines the latter format, which is referred to as the "NUPN format".

Conversion from PNML to NUPN can be done using the PNML2NUPN tool (see below), whereas conversion from NUPN to PNML (with NUPN-specific extension) can be achieved by invoking the **caesar.bdd**(LOCAL) tool with its **-pnml** option.

SYNTAX OF THE NUPN FORMAT

The format of *filename***.nupn** is defined by the following BNF grammar (where **<element>?** denotes zero or one ocurrences of **<element>**, where **<element>**∗ denotes zero or many occurrences of **<element>**, and where "nb" stands for "number"):

<nested-unit-petri-net> ::=
  <pragma-description>∗
  places #<nb-of-places> <min-place-nb>...<max-place-nb>\n
  <initial-marking-description>
  units #<nb-of-units> <min-unit-nb>...<max-unit-nb>\n
  root unit <root-unit-nb>\n
  <unit-description>∗\n
  transitions #<nb-of-trans> <min-trans-nb>...<max-trans-nb>\n
  <trans-description>∗\n
  <label-description>?

<pragma-description> ::= !<character-string>\n

<initial-marking-description> ::=
  initial place <init-place-nb>\n
| initial places #<nb-of-initial-places> <initial-place-list>\n

<unit-description> ::=

```
  U<unit-nb>
  #<nb-of-subplaces> <min-subplace-nb>...<max-subplace-nb>
  #<nb-of-subunits> <subunit-list>\n

<trans-description> ::=
  T<trans-nb>
  #<nb-of-input-places> <input-place-list>
  #<nb-of-output-places> <output-place-list>\n

<label-description> ::=
  labels <place-flag> <trans-flag> <unit-flag> <label-length>\n
  <place-label>*
  <trans-label>*
  <unit-label>*

<place-label> ::= p<place-nb> <label>\n
<trans-label> ::= t<trans-nb> <label>\n
<unit-label>  ::= u<unit-nb> <label>\n

<initial-place-list>   ::= <place-nb>*
<input-place-list>     ::= <place-nb>*
<output-place-list>    ::= <place-nb>*
<subunit-list>         ::= <unit-nb>*

<nb-of-places>         ::= <unsigned-integer>
<min-place-nb>         ::= <unsigned-integer>
<max-place-nb>         ::= <unsigned-integer>
<init-place-nb>        ::= <unsigned-integer>
<place-nb>             ::= <unsigned-integer>
<nb-of-units>          ::= <unsigned-integer>
<min-unit-nb>          ::= <unsigned-integer>
<max-unit-nb>          ::= <unsigned-integer>
<root-unit-nb>         ::= <unsigned-integer>
<unit-nb>              ::= <unsigned-integer>
<nb-of-trans>          ::= <unsigned-integer>
<min-trans-nb>         ::= <unsigned-integer>
<max-trans-nb>         ::= <unsigned-integer>
<nb-of-subplaces>      ::= <unsigned-integer>
<min-subplace-nb>      ::= <unsigned-integer>
<max-subplace-nb>      ::= <unsigned-integer>
<nb-of-subunits>       ::= <unsigned-integer>
<trans-nb>             ::= <unsigned-integer>
<nb-of-input-places>   ::= <unsigned-integer>
<nb-of-output-places>  ::= <unsigned-integer>
<label-length>         ::= <unsigned-integer>

<place-flag>           ::= 0 | 1
<trans-flag>           ::= 0 | 1
<unit-flag>            ::= 0 | 1

<label>                ::= <character-string>
```

In order to keep the NUPN format simple and easy to process using script languages (such as Awk), the rules for spacing are strict: tabulations are prohibited; multiple spaces are not allowed where a single space is; spaces are not permitted at the beginning or end of a line; empty or blank lines are forbidden.

Note 1: inserting spaces just after "!" and "#", or around "..." is not permitted by the syntax.

Note 2: in <initial-marking-description>, the first form:
  initial place <init-place-nb>\n
is a shorthand for the second form and, namely, is equivalent to:
  initial places #1 <init-place-nb>\n
The first form is an early syntax, kept for backward compatibility reasons. In the sequel, we may only consider the second form.

Note 3: unless otherwise stated, all <unsigned-integer> values are assumed to be smaller than 2^31.

STATIC SEMANTICS OF THE NUPN FORMAT

A few preliminary definitions are needed:

- Let Length (L) denote the length of a list L whose elements belong to the syntactic categories <place-nb> or <unit-nb>; the length of an empty list is zero.
- Let Unit (<place-nb>) be the <unit-nb> associated with the <unit-description> whose interval <min-subplace-nb>...<max-subplace-nb> contains <place-nb>.
- Let Sub (u1, u2) be the predicate that is true iff the unit numbered u1 is a sub-unit of the unit numbered u2, i.e., iff there exists a <unit-description> whose <unit-nb> is u2 and whose <subunit-list> contains u1.
- Let Sub* (u1, u2) be the transitive closure of Sub (u1, u2), i.e., the predicate that is true iff the unit numbered u1 is transitively included in (but not equal to) the unit numbered u2.
- Let Disjoint (u1, u2) be the predicate that is true iff both units numbered u1 and u2 are neither equal nor transitively included in each other, i.e., iff (u1 <> u2) and not Sub* (u1, u2) and not Sub* (u2, u1).

A valid NUPN file should satisfy the following constraints:

In <nested-unit-petri-net>:
1. <nb-of-places> > 0 -- a net has at least one place
2. <max-place-nb> - <min-place-nb> + 1 = <nb-of-places>
3. <nb-of-units> > 0 -- a net has at least one unit
4. <max-unit-nb> - <min-unit-nb> + 1 = <nb-of-units>
5. <min-unit-nb> <= <root-unit-nb> <= <max-unit-nb>
6. <nb-of-trans> >= 0 -- a net may have zero transition
7. <nb-of-trans> = 0 => <min-trans-nb> = 1 and <max-trans-nb> = 0
   -- the empty interval of transitions is canonically noted 1...0
8. <max-trans-nb> - <min-trans-nb> + 1 = <nb-of-trans>

In <initial-marking-description>:
9. <min-place-nb> <= <init-place-nb> <= <max-place-nb>
10. 0 <= <nb-of-initial-places> <= <nb-of-places>
11. Length (<initial-place-list>) = <nb-of-initial-places>
12. for each pair <place-nb> and <place-nb>' in <initial-place-list>
    one must have: Disjoint (Unit (<place-nb>), Unit (<place-nb>'))

-- the initial places must belong to disjoint units (and are
-- thus pairwise distinct); see discussion below about this rule

In each <unit-description>:
13. <min-unit-nb> <= <unit-nb> <= <max-unit-nb>
14. 0 <= <nb-of-subplaces> <= <nb-of-places>
15. <nb-of-subplaces> = 0 => <min-subplace-nb> = 1
and <max-subplace-nb> = 0
-- the empty interval of places is canonically noted 1...0
16. <min-place-nb> <= <min-subplace-nb> <= <max-place-nb>
17. <min-place-nb> <= <max-subplace-nb> <= <max-place-nb>
18. <max-subplace-nb> - <min-sublace-nb> + 1 = <nb-of-subplaces>
19. 0 <= <nb-of-subunits> <= <nb-of-units>
20. Length (<subunit-list>) = <nb-of-subunits>

Globally to all <unit-description>s:
21. each <unit-nb> occurs once and only once after a "U"
22. the sum of all <nb-of-subplaces> is equal to <nb-of-places>
23. all intervals <min-subplace-nb>...<max-subplace-nb> form
a partition of <min-place-nb>...<max-place-nb>
24. the sum of all <nb-of-subunits> is equal to <nb-of-units> - 1
25. <root-unit-number> and all non-empty <subunit-list>s form
a partition of <min-unit-nb>...<max-unit-nb>

In each <subunit-list>:
26. <min-unit-nb> <= <unit-nb> <= <max-unit-nb>
27. <unit-nb> != <root-unit-nb> -- the root unit is not a
-- sub-unit of any other unit

In each <trans-description>:
28. <min-trans-nb> <= <trans-nb> <= <max-trans-nb>
29. 0 <= <nb-of-input-places> <= <nb-of-places>
30. Length (<input-place-list>) = <nb-of-input-places>
31. 0 <= <nb-of-output-places> <= <nb-of-places>
32. Length (<output-place-list>) = <nb-of-output-places>
33. <input-place-list> is included in <output-place-list>
=> <input-place-list> is equal to <output-place-list>
-- see discussion below about this rule

Globally to all <trans-description>s:
34. each <trans-nb> occurs once and only once after a "T"

In each <input-place-list> and each <output-place-list>:
35. <min-place-nb> <= <place-nb> <= <max-place-nb>
36. for each pair <place-nb> and <place-nb>' in the list, one
must have: Disjoint (Unit (<place-nb>), Unit (<place-nb>'))
-- the input (resp. output) places of each transition must
-- belong to disjoint units (and are thus pairwise distinct);
-- see discussion below about this rule

In <label-description>:
37. if <place-flag> = 0, there are no occurrences of <place-label>
-- places are not labelled

38. if <place-flag> = 1, there are <nb-of-places> occurrences of
    <place-label> -- all places are labelled
39. if <trans-flag> = 0, there are no occurrences of <trans-label>
    -- transitions are not labelled
40. if <trans-flag> = 1, there are <nb-of-trans> occurrences of
    <trans-label> -- all transitions are labelled
41. <nb-of-trans> = 0 => <trans-flag> = 0
42. if <unit-flag> = 0, there are no occurrences of <unit-label>
    -- units are not labelled
43. if <unit-flag> = 1, there are <nb-of-units> occurrences of
    <unit-label> -- all units are labelled

In each <place-label>:
44. <min-place-nb> <= <place-nb> <= <max-place-nb>

Globally to all <place-label>s:
45. each <place-nb> occurs once and only once after a "p"
    -- thus all <place-nb>s are pairwise distinct, but it is not
    -- required that all <character-string>s are pairwise distinct

In each <trans-label>:
46. <min-trans-nb> <= <trans-nb> <= <max-trans-nb>

Globally to all <trans-label>s:
47. each <trans-nb> occurs once and only once after a "t"
    -- thus all <trans-nb>s are pairwise distinct, but it is not
    -- required that all <character-string>s are pairwise distinct

In each <unit-label>:
48. <min-unit-nb> <= <unit-nb> <= <max-unit-nb>

Globally to all <unit-label>s:
49. each <unit-nb> occurs once and only once after a "u"
    -- thus all <unit-nb>s are pairwise distinct, but it is not
    -- required that all <character-string>s are pairwise distinct

In each <label>:
50. strlen (<character-string>) <= <label-length>

Note 1: It is possible that:
  <nb-of-units> <= <nb-of-places>
in the case, e.g., where all places are contained in one single unit. Conversely, it is also possible that:
  <nb-of-places> <= <nb-of-units>
in the case, e.g., where each place is contained in a specific unit.

Note 2: when there is a single initial place, it is often in the root unit, but this is not mandatory (this would be even impossible when the root unit has no local place).

Note 3: as stated in constraints 46, 49, and 52, the labelling of places, transitions, and units is not necessarily injective. For instance, several transitions may share the same label (e.g., the label "i" used to denote internal transitions in concurrent languages such as LOTOS and LNT).

DYNAMIC SEMANTICS OF THE NUPN FORMAT

The dynamic semantics of Nested-Unit Petri Nets follows the same rules as Petri Nets that are ordinary (i.e., all arcs have multiplicity one) and safe (i.e., each place contains at most one token):

- Initially, only the initial places (which are defined in the <initial-marking-description>) have a token.
- A transition can fire iff all its input places have a token.
- When a transition fires, its input places lose their token and its output place get a token.

Additionally, each reachable marking M should satisfy the "unit safe" property, which is defined as follows:

(1) If M has a token in some place P, then M has no token in any other place P' local to the same unit as P, i.e., Unit (P) = Unit (P'). The particular case where P=P' prohibits having two tokens in the same place, thus implying that the net is one-safe.

(2) If M has a token in some place P local to U = Unit (P), there is no token in any place local to any other unit U' that transitively contains U or is transitively contained in U, i.e., Sub* (U, U') or Sub* (U', U). Said differently, if a unit is active, all its ancestor units and descendent units are inactive.

Conditions (1) and (2) can be summarized as follows: a marking M is unit safe iff for each pair of places P and P' marked in M, P <> P' => Disjoint (Unit (P), Unit (P')).

Deciding whether all reachable markings of a NUPN model are unit safe requires, in general, to build the graph of reachable markings. However, some structural checks allow to detect, on the basis of sufficient conditions, certain NUPN models that are likely not to be unit safe. This is the case of three aforementioned rules of the static semantics:

- Rule 12 checks that the initial marking is unit safe.
- Rule 34 warns about transitions T whose input places are a strict subset of output places, as such transitions, if fireable, would accumulate an infinite number of tokens in their output places that are not input places.
- Rule 37 checks, for each transition T, that the set of input places of T is unit safe (otherwise, T could not fire from any unit safe marking) and that the set of output places of T is unit safe too (otherwise, firing T would produce markings that would not be unit safe).

Note: it is neither required that a NUPN is connected, nor that all places can be reachable from the initial marking, nor that all transitions are quasi-live (i.e., can be fired from at least one reachable marking). However, **caesar.bdd (LOCAL)** invoked with option **-check** warns about such situations.

SUPPORTED PRAGMAS OF THE NUPN FORMAT

Pragmas provide optional information about a NUPN model. If present, pragmas occur at the very beginning of a NUPN file; they always start with the "!" character and terminate with the end of the line. The following pragmas are currently supported:

PRAGMA CREATOR

The syntax of this pragma is:

!creator <character-string>

<character-string> should contain the name of the tool that produced the NUPN file, possibly followed by the version number of this tool and (if relevant) the command-line options given to this tool when it was invoked. This pragma is mostly intended for traceability.

PRAGMA UNIT_SAFE

The syntax of this pragma is either:

```
!unit_safe
```

or:

```
!unit_safe <character-string>
```

If present, this pragma certifies that all reachable markings of the NUPN model are unit safe. In the first form, unit safeness is certified by the creator tool (if the "!creator" pragma is present). In the second form, unit safeness has been certified by another tool, the name, version number, and (if relevant) command-line options are given in <character-string>.

PRAGMA MULTIPLE_INITIAL_TOKENS

The syntax of this pragma is:

```
!multiple_initial_tokens #<nb-tokens>
#<nb-places> <min>...<max>
```

As for all pragmas, these syntactic elements must appear on one single line.

<nb-tokens>, <min>, and <max> are unsigned integers that may be greater than 2^31 but must be smaller than 2^63.

By inserting this pragma, the creator tool indicates that the NUPN model has been derived from a original Petri Net, in the initial marking of which certain places contain more than one token. In the NUPN model, the initial number of tokens in such places is implicitly reduced to one.

Thus, this pragma and the **unit_safe** pragma are mutually exclusive.

The arguments of this pragma are computed, on the original Petri Net, as follows:

- <nb-tokens> gives the sum of tokens initially contained in all the initial places (either marked with a single or multiple tokens).
- <nb-places> gives the number of places that initially contain multiple tokens.
- <min> and <max> give respectively the minimum (greater than one) and maximum number of initial tokens present in the places that initially contain multiple tokens.

The following constraints should hold, where <nb-of-initial-places> denotes the number of initial places given in the <initial-marking-description>:

```
<nb-of-initial-places> + <nb-places> * (<min> - 1)
<= <nb-tokens> <=
<nb-of-initial-places> + <nb-places> * (<max> - 1)
```

PRAGMA MULTIPLE_ARCS

The syntax of this pragma is:

```
!multiple_arcs
#<nb-trans-in> #<nb-trans-out> #<nb-trans-inout>
<min-in>...<max-in> <min-out>...<max-out>
<min-diff>...<max-diff>
```

As for all pragmas, these syntactic elements must appear on one single line.

By inserting this pragma, the creator tool indicates that the NUPN model has been derived from a original Petri Net that is not ordinary, i.e., in which certain input arcs (i.e., from a place to a transition) and/or output arcs (i.e., from a transition to a place) have a valuation (or "inscription", in PNML terminology) greater than one. Such arcs are called "multiple input arcs" and "multiple output arcs", respectively. In the NUPN model, there are no multiple arcs.

Thus, this pragma and the **unit_safe** pragma are mutually exclusive.

The arguments of this pragma are computed, on the original Petri Net, as follows:

- <nb-trans-in> gives the number of transitions having at least one multiple input arc and no multiple output arc.
- <nb-trans-out> gives the number of transitions having no multiple input arc and at least one multiple output arc.
- <nb-trans-inout> gives the number of transitions having at least one multiple input arc and at least one multiple output arc. At least one among <nb-trans-in>, <nb-trans-out>, and <nb-trans-inout> must be non-null.
- <min-in> and <max-in> give respectively the minimum (greater than one) and maximum valuations of multiple input arcs. In absence of such arcs (i.e., when <nb-trans-in> and <nb-trans-inout> are both null), <min-in> and <max-in> must be equal to 1 and 0, respectively.
- <min-out> and <max-out> give respectively the minimum (greater than one) and maximum valuations of multiple output arcs. In absence of such arcs (i.e., when <nb-trans-out> and <nb-trans-inout> are both null), <min-out> and <max-out> must be equal to 1 and 0, respectively.
- <min-diff> and <max-diff> give the minimum and maximum, for all transitions T, of OutVal (T) - InVal (T), where InVal (T) and OutVal (T) denote the sum of the valuations of all input arcs arriving in T (respectively, all output arcs going out of T). Each arc that is not multiple is assumed to have a valuation equal to one. <min-diff> and <max-diff> are computed on the entire set of transitions of the original Petri Net (this set is necessarily non empty if this pragma is present).

EXAMPLE OF A NUPN FILE

This is an example of a valid NUPN file:

```
!creator caesar
!unit_safe
places #7 0...6
initial place 0
units #3 0...2
root unit 0
U1 #4 1...4 #0
U2 #2 5...6 #0
U0 #1 0...0 #2 1 2
transitions #5 0...4
T0 #1 0 #2 1 5
T1 #2 3 6 #2 2 5
T2 #2 1 5 #2 4 6
T3 #1 2 #1 1
T4 #1 4 #1 3
```

HOW TO CREATE A NUPN FILE

At present, there are two ways of producing NUPN models:

- Using the **-nupn** option of the **caesar**(LOCAL) compiler, which will generate a NUPN file corresponding to the control structure of a LOTOS specification (or any other specification, for instance, an LNT specification, that can be translated to LOTOS).
- Using the PNML2NUPN translator developed by Lom Messan Hillah - see http://pnml.lip6.fr/pnml2nupn - which converts to NUPN a large subset of Petri Nets encoded in the PNML standard. Notice that PNML models containing arcs with multiple valuations (i.e., wih PNML "inscription" attributes greater than one) or initial places with more than one token (i.e., with PNML "initialMarking" attributes greater than one) cannot be easily expressed in NUPN. Also, the models generated by PNML2NUPN are unstructured (technically, each place is contained in a particular NUPN unit).

HOW TO READ A NUPN FILE

At present, there is one single CADP tool, **caesar.bdd**(LOCAL), that reads and processes NUPN files. This tool, when invoked with its **-check** option, is the reference parser and semantic checker for the NUPN format.

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.bdd(LOCAL), **caesar**(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 bugs to cadp@inria.fr